Specification of the IP Flow Information Export (IPFIX) File Format

Abstract

   This document describes a file format for the storage of flow data
   based upon the IP Flow Information Export (IPFIX) protocol. It
   proposes a set of requirements for flat-file, binary flow data file
   formats, then specifies the IPFIX File format to meet these
   requirements based upon IPFIX Messages. This IPFIX File format is
   designed to facilitate interoperability and reusability among a wide
   variety of flow storage, processing, and analysis tools.

Status of This Memo

   This document specifies an Internet standards track protocol for the
   Internet community, and requests discussion and suggestions for
   improvements. Please refer to the current edition of the "Internet
   Official Protocol Standards" (STD 1) for the standardization state
   and status of this protocol. Distribution of this memo is unlimited.

Table of Contents

1. Introduction

   This document specifies a file format based upon IPFIX, designed to
   facilitate interoperability and reusability among a wide variety of
   flow storage, processing, and analysis tools. It begins with an
   overview of the IPFIX File format, and a quick summary of how IPFIX
   Files work in Section 3. The detailed specification of the IPFIX
   File format appears in Section 7; this section includes general
   specifications for IPFIX File Readers and IPFIX File Writers and
   specific recommendations for common situations in which they are
   used. The format makes use of the IPFIX Options mechanism for
   additional file metadata, in order to avoid requiring any protocol
   extensions, and to minimize the effort required to adapt IPFIX
   implementations to use the file format; a detailed definition of the
   Options Templates used for storage metadata appears in Section 8.
   Appendix A contains a detailed example IPFIX File.

   An advantage of file-based storage is that files can be readily
   encapsulated within each other and other data storage and
   transmission formats. The IPFIX File format leverages this to
   provide encryption, described in Section 9 and compression, described
   in Section 10. Section 11 provides specific recommendations for
   integration of IPFIX File data with other formats.

   The IPFIX File format was designed to be applicable to a wide variety
   of flow storage situations; the motivation behind its creation is
   described in Section 4. The document outlines of the set of
   requirements the format is designed to meet in Section 5, and
   explores the applicability of such a format to various specific
   application areas in Section 6. These sections are intended to give
   background on the development of IPFIX Files.

1.1. IPFIX Documents Overview

   "Specification of the IP Flow Information Export (IPFIX) Protocol for
   the Exchange of IP Traffic Flow Information" [RFC5101] and its
   associated documents define the IPFIX protocol, which provides
   network engineers and administrators with access to IP traffic flow
   information.

"Architecture for IP Flow Information Export" [RFC5470] defines the
architecture for the export of measured IP flow information out of an
IPFIX Exporting Process to an IPFIX Collecting Process, and the basic
terminology used to describe the elements of this architecture, per
the requirements defined in "Requirements for IP Flow Information
Export" [RFC3917]. [RFC5101] then covers the details of the method
for transporting IPFIX Data Records and Templates via a congestion-
aware transport protocol from an IPFIX Exporting Process to an IPFIX

Collecting Process.

"Information Model for IP Flow Information Export" [RFC5102]
describes the Information Elements used by IPFIX, including details
on Information Element naming, numbering, and data type encoding.

"IP Flow Information Export (IPFIX) Applicability" [RFC5472]
describes the various applications of the IPFIX protocol and their
use of information exported via IPFIX, and it relates the IPFIX
architecture to other measurement architectures and frameworks.

In addition, "Exporting Type Information for IP Flow Information
Export (IPFIX) Information Elements" [RFC5610] specifies a method for
encoding Information Model properties within an IPFIX Message stream.

This document references [RFC5101] and [RFC5470] for terminology,
defines IPFIX File Writer and IPFIX File Reader in terms of the IPFIX
Exporting Process and IPFIX Collecting Process definitions from
[RFC5101], and extends the IPFIX Information Model defined in
[RFC5102] to provide new Information Elements for IPFIX File
metadata. It uses the method described in [RFC5610] to support the
self-description of IPFIX Files containing enterprise-specific
Information Elements.

2. Terminology

   This section defines terminology related to the IPFIX File format.
   In addition, terms used in this document that are defined in the
   "Terminology" section of [RFC5101] are to be interpreted as defined
   there.

   IPFIX File: An IPFIX File is a serialized stream of IPFIX Messages;
      this stream may be stored on a filesystem or transported using any
      technique customarily used for files. Any IPFIX Message stream
      that would be considered valid when transported over one or more
      of the specified IPFIX transports (Stream Control Transmission
      Protocol (SCTP), TCP, or UDP) as defined in [RFC5101] is

      considered an IPFIX File. However, this document extends that
      definition with recommendations on the construction of IPFIX Files
      that meet the requirements identified in Section 5.

   IPFIX File Reader: An IPFIX File Reader is a process that reads
      IPFIX Files from a filesystem. An IPFIX File Reader operates as
      an IPFIX Collecting Process as specified in [RFC5101], except as
      modified by this document.

   IPFIX File Writer: An IPFIX File Writer is a process that writes
      IPFIX Files to a filesystem. An IPFIX File Writer operates as an
      IPFIX Exporting Process as specified in [RFC5101], except as
      modified by this document.

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

3. Design Overview

   An IPFIX File is simply a data stream containing one or more IPFIX
   Messages serialized to some filesystem. Though any set of valid
   IPFIX Messages can be serialized into an IPFIX File, the
   specification includes guidelines designed to ease storage and
   retrieval of flow data using the IPFIX File format.

   IPFIX Files contain only IPFIX Messages; any file metadata such as
   checksums or export session details are stored using Options within
   the IPFIX Message. This design is completely compatible with the
   IPFIX protocol on the wire. A schematic of a typical IPFIX File is
   shown below:

```
          +======================================+
          | IPFIX File                           |
          | +==================================+ |
          | | IPFIX Message                    | |
          | | +------------------------------+ | |
          | | | IPFIX Message Header         | | |
          | | +------------------------------+ | |
          | | +------------------------------+ | |
          | | | Options Template Set         | | |
          | | |    Options Template Record   | | |
          | | |           . . .              | | |
          | | +------------------------------+ | |
          | | +------------------------------+ | |
          | | | Template Set                 | | |
          | | |    Template Record           | | |
          | | |           . . .              | | |
          | | +------------------------------+ | |
          | +==================================+ |
          | | IPFIX Message                    | |
          | | +------------------------------+ | |
          | | | IPFIX Message Header         | | |
          | | +------------------------------+ | |
          | | +------------------------------+ | |
          | | | Data Set                     | | |
          | | |    Data Record               | | |
          | | |           . . .              | | |
          | | +------------------------------+ | |
          | | +------------------------------+ | |
          | | | Data Set                     | | |
          | | |    Data Record               | | |
          | | |           . . .              | | |
          | | +------------------------------+ | |
          | |           . . .                  | |
          | +==================================+ |
          |             . . .                    |
          +======================================+
```
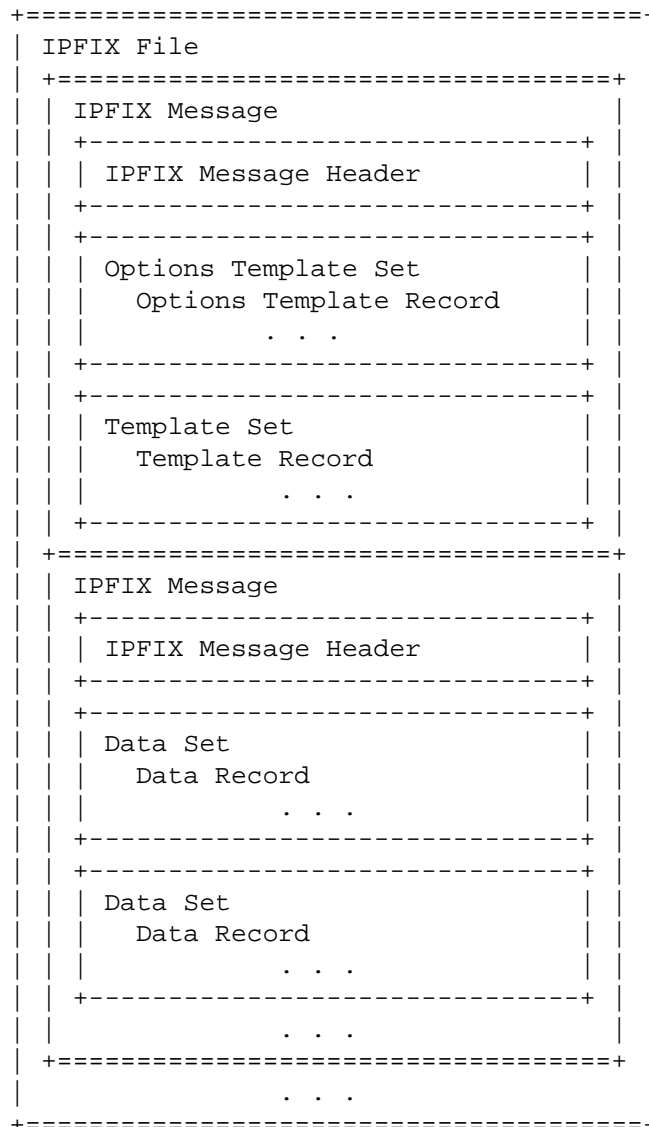
                  Figure 1: Typical File Structure

4. Motivation

   There is a wide variety of applications for the file-based storage of
   IP flow data, across a continuum of time scales. Tools used in the
   analysis of flow data and creation of analysis products often use
   files as a convenient unit of work, with an ephemeral lifetime. A
   set of flows relevant to a security investigation may be stored in a
   file for the duration of that investigation, and further exchanged
   among incident handlers via email or within an external incident

handling workflow application. Sets of flow data relevant to
Internet measurement research may be published as files, much as
libpcap [pcap] packet trace files are, to provide common datasets for
the repeatability of research efforts; these files would have
lifetimes measured in months or years. Operational flow measurement
systems also have a need for long-term, archival storage of flow
data, either as a primary flow data repository, or as a backing tier
for online storage in a relational database management system
(RDBMS).

The variety of applications of flow data, and the variety of
presently deployed storage approaches, indicates the need for a
standard approach to flow storage with applicability across the
continuum of time scales over which flow data is stored. A storage
format based around flat files would best address the variety of
storage requirements. While much work has been done on structured
storage via RDBMS, relational database systems are not a good basis
for format standardization owing to the fact that their internal data
structures are generally private to a single implementation and
subject to change for internal reasons. Also, there are a wide
variety of operations available on flat files, and external tools and
standards can be leveraged to meet file-based flow storage
requirements. Further, flow data is often not very semantically
complicated, and is managed in very high volume; therefore, an RDBMS-
based flow storage system would not benefit much from the advantages
of relational database technology.

The simplest way to create a new file format is simply to serialize
some internal data model to disk, with either textual or binary
representation of data elements, and some framing strategy for
delimiting fields and records. "Ad hoc" file formats such as this
have several important disadvantages. They impose the semantics of
the data model from which they are derived on the file format, and as
such, they are difficult to extend, describe, and standardize.

Indeed, one de facto standard for the storage of flow data is one of
these ad hoc formats. A common method of storing data collected via
Cisco NetFlow is to serialize a stream of raw NetFlow datagrams into
files. These NetFlow PDU files consist of a collection of header-
prefixed blocks (corresponding to the datagrams as received on the
wire) containing fixed-length binary flow records. NetFlow V5, V7,
and V8 data may be mixed within a given file, as the header on each
datagram defines the NetFlow version of the records following. While
this NetFlow PDU file format has all the disadvantages of an ad hoc
format, and is not extensible to data models other than that defined
by Cisco NetFlow, it is at least reasonably well understood due to
its ubiquity.

Over the past decade, XML has emerged as a new "universal"
representation format for structured data. It is intended to be
human readable; indeed, that is one reason for its rapid adoption.
However, XML has limited usefulness for representing network flow
data. Network flow data has a simple, repetitive, non-hierarchical
structure that does not benefit much from XML. An XML representation
of flow data would be an essentially flat list of the attributes and
their values for each flow record.

The XML approach to data encoding is very heavyweight when compared
to binary flow encoding. XML's use of start- and end-tags, and
plaintext encoding of the actual values, leads to significant
inefficiency in encoding size. Typical network traffic datasets can
contain millions or billions of flows per hour of traffic
represented. Any increase in storage size per record can have
dramatic impact on flow data storage and transfer sizes. While data
compression algorithms can partially remove the redundancy introduced
by XML encoding, they introduce additional overhead of their own.

A further problem is that XML processing tools require a full XML
parser. XML parsers are fully general and therefore complex,
resource-intensive, and relatively slow, introducing significant
processing time overhead for large network-flow datasets. In
contrast, parsers for typical binary flow data encodings are simply
structured, since they only need to parse a very small header and
then have complete knowledge of all following fields for the
particular flow. These can then be read in a very efficient linear
fashion.

This leads us to propose the IPFIX Message format as the basis for a
new flow data file format. The IPFIX Working Group, in defining the
IPFIX protocol, has already defined an information model and data
formatting rules for representation of flow data. Especially at
shorter time scales, when a file is a unit of data interchange, the
filesystem may be viewed as simply another IPFIX Message transport
between processes. This format is especially well suited to
representing flow data, as it was designed specifically for flow data
export; it is easily extensible, unlike ad hoc serialization, and
compact, unlike XML. In addition, IPFIX is an IETF Standards-Track
protocol for the export and collection of flow data; using a common
format for storage and analysis at the collection side allows
implementors to use substantially the same information model and data
formatting implementation for transport as well as storage.

5. Requirements

   In this section, we outline a proposed set of requirements
   [SAINT2007] for any persistent storage format for flow data. First
   and foremost, a flow data file format should support storage across
   the continuum of time scales important to flow storage applications.
   Each of the requirements enumerated in the sections below is broadly
   applicable to flow storage applications, though each may be more
   important at certain time scales. For each, we first identify the
   requirement, then explain how the IPFIX Message format addresses it,
   or briefly outline the changes that must be made in order for an
   IPFIX-based file format to meet the requirement.

5.1. Record Format Flexibility

   Due to the wide variety of flow attributes collected by different
   network flow attribute measurement systems, the ideal flow storage
   format will not impose a single data model or a specific record type
   on the flows it stores. The file format must be flexible and
   extensible; that is, it must support the definition of multiple
   record types within the file itself, and must be able to support new
   field types for data within the records in a graceful way.

   IPFIX provides record format flexibility through the use of Templates
   to describe each Data Record, through the use of an IANA Registry to
   define its Information Elements, and through the use of enterprise-
   specific Information Elements.

5.2. Self-Description

   Archived data may be read at a time in the future when any external
   reference to the meaning of the data may be lost. The ideal flow
   storage format should be self-describing; that is, a process reading
   flow data from storage should be able to properly interpret the
   stored flows without reference to anything other than standard
   sources (e.g., the standards document describing the file format) and
   the stored flow data itself.

   The IPFIX Message format is partially self-describing; that is, IPFIX
   Templates containing only IANA-assigned Information Elements can be
   completely interpreted according to the IPFIX Information Model
   without additional external data.

   However, Templates containing private information elements lack
   detailed type and semantic information; a Collecting Process
   receiving Data Records described by a Template containing enterprise-
   specific Information Elements it does not understand can only treat
   the data contained within those Information Elements as octet arrays.

To be fully self-describing, enterprise-specific Information Elements
must be additionally described via IPFIX Options according to the
Information Element Type Options Template defined in [RFC5610].

5.3. Data Compression

Regardless of the representation format, flow data describing traffic
on real networks tends to be highly compressible. Compression tends
to improve the scalability of flow collection systems, by reducing
the disk storage and I/O bandwidth requirement for a given workload.
The ideal flow storage format should support applications that wish
to leverage this fact by supporting compression of stored data.

The IPFIX Message format has no support for data compression, as the
IPFIX protocol was designed for speed and simplicity of export. Of
course, any flat file is readily compressible using a wide variety of
external data compression tools, formats, and algorithms; therefore,
this requirement can be met via encapsulation in one of these
formats. Section 10 specifies an encapsulation based on bzip2 or
gzip, to maximize interoperability.

A few simple optimizations can be made by File Writers to increase
the integrity and usability of compressed IPFIX data; these are
outlined in Section 10.3.

5.4. Indexing and Searching

Binary, record-stream-oriented file formats natively support only one
form of searching: sequential scan in file order. By choosing the
order of records in a file carefully (e.g., by flow end time), a file
can be indexed by a single key.

Beyond this, properly addressing indexing is an application-specific
problem, as it inherently involves trade-offs between storage
complexity and retrieval speed, and requirements vary widely based on
time scales and the types of queries used from site to site.
However, a generic standard flow storage format may provide limited
direct support for indexing and searching.

The ideal flow storage format will support a limited table of
contents facility noting that the records in a file contain data
relating only to certain keys or values of keys, in order to keep
multi-file search implementations from having to scan a file for data
it does not contain.

The IPFIX Message format has no direct support for indexing.
However, the technique described in "Reducing Redundancy in IP Flow
Information Export (IPFIX) and Packet Sampling (PSAMP) Reports"

   [RFC5473] can be used to describe the contents of a file in a limited
   way. Additionally, as flow data is often sorted and divided by time,
   the start and end time of the flows in a file may be declared using
   the File Time Window Options Template defined in Section 8.1.2.

5.5. Error Recovery

   When storing flow data for archival purposes, it is important to
   ensure that hardware or software faults do not introduce errors into
   the data over time. The ideal flow storage format will support the
   detection and correction of encoding-level errors in the data.

   Note that more advanced error correction is best handled at a layer
   below that addressed by this document. Error correction is a topic
   well addressed by the storage industry in general (e.g., by Redundant
   Array of Independent Disks (RAID) and other technologies). By
   specifying a flow storage format based upon files, we can leverage
   these features to meet this requirement.

   However, the ideal flow storage format will be resilient against
   errors, providing an internal facility for the detection of errors
   and the ability to isolate errors to as few data records as possible.

   Note that this requirement interacts with the choice of data
   compression or encryption algorithm. For example, the use of block
   compression algorithms can serve to isolate errors to a single
   compression block, unlike stream compressors, which may fail to
   resynchronize after a single bit error, invalidating the entire
   message stream.

   The IPFIX Message format does not support data integrity assurance.
   It is assumed that advanced error correction will be provided
   externally. Compression and encryption, if used, provide some
   allowance for detection, if not correction, of errors. For simple
   error detection support in the absence of compression or encryption,
   checksums may be attached to messages via IPFIX Options according to
   the Message Checksum Options Template defined in Section 8.1.1.

5.6. Authentication, Confidentiality, and Integrity

   Archival storage of flow data may also require assurance that no
   unauthorized entity can read or modify the stored data. Cryptography
   can be applied to this problem to ensure integrity and
   confidentiality by signing and encryption.

As with error correction, this problem has been addressed well at a
layer below that addressed by this document. We can leverage the
fact that existing cryptographic technologies work quite well on data
stored in files to meet this requirement.

Beyond support for the use of Transport Layer Security (TLS) for
transport over TCP or Datagram Transport Layer Security (DTLS) for
transport over SCTP or UDP, both of which provide transient
authentication and confidentiality, the IPFIX protocol does not
support this requirement directly. The IETF has specified the
Cryptographic Message Syntax (CMS) [RFC3852] for creating detached
signatures for integrity and authentication; Section 9 specifies a
CMS-based method for signing IPFIX Files. Confidentiality protection
is assumed to be met by methods external to this specification,
leveraging one of the many such technologies for encrypting files to
meet specific application and process requirements; however, notes on
improving archival integrity of encrypted IPFIX Files are given in
Section 9.2.

5.7. Anonymization and Obfuscation

To ensure the privacy of individuals and organizations at the
endpoints of communications represented by flow records, it is often
necessary to obfuscate or anonymize stored and exported flow data.
The ideal flow storage format will provide for a notation that a
given information element on a given record type represents
anonymized, rather than real, data.

The IPFIX protocol presently has no support for anonymization
notation. It should be noted that anonymization is one of the
requirements given for IPFIX in [RFC3917]. The decision to qualify
this requirement with 'MAY' and not 'MUST' in the requirements
document, and its subsequent lack of specification in the current
version of the IPFIX protocol, is due to the fact that anonymization
algorithms are still an open area of research, and that there
currently exist no standardized methods for anonymization.

No support is presently defined in [RFC5101] or this IPFIX-based File
format for anonymization, as anonymization notation is an area of
open work for the IPFIX Working Group.

5.8. Session Auditability and Replayability

Certain use cases for archival flow storage require the storage of
collection infrastructure details alongside the data itself. These
details include information about how and when data was received, and
where it was received from. They are useful for auditing as well as
for the replaying received data for testing purposes.

The IPFIX protocol contains no direct support for auditability and
replayability, though the IPFIX Information Model does define various
Information Elements required to represent collection infrastructure
details. These details may be stored in IPFIX Files using the Export
Session Details Options Template defined in Section 8.1.3, and the
Message Details Options Template defined in Section 8.1.4.

5.9. Performance Characteristics

   The ideal standard flow storage format will not have a significant
   negative impact on the performance of the application generating or
   processing flow data stored in the format. This is a non-functional
   requirement, but it is important to note that a standard that implies
   a significant performance penalty is unlikely to be widely
   implemented and adopted.

   An examination of the IPFIX protocol would seem to suggest that
   implementations of it are not particularly prone to slowness; indeed,
   a template-based data representation is more easily subject to
   optimization for common cases than representations that embed
   structural information directly in the data stream (e.g., XML).
   However, a full analysis of the impact of using IPFIX Messages as a
   basis for flow data storage on read/write performance will require
   more implementation experience and performance measurement.

6. Applicability

   This section describes the specific applicability of IPFIX Files to
   various use cases. IPFIX Files are particularly useful in a flow
   collection and processing infrastructure using IPFIX for flow export.
   We explore the applicability and provide guidelines for using IPFIX
   Files for the storage of flow data collected by IPFIX Collecting
   Processes and NetFlow V9 collectors, the testing of IPFIX Collecting
   Processes, and diagnostics of IPFIX Devices.

6.1. Storage of IPFIX-Collected Flow Data

   IPFIX Files can naturally be used to store flow data collected by an
   IPFIX Collecting Process; indeed, this was one of the primary initial
   motivations behind the file format described within this document.
   Using IPFIX Files as such provides a single, standard, well-
   understood encoding to be used for flow data on disk and on the wire,
   and allows IPFIX implementations to leverage substantially the same
   code for flow export and flow storage. In addition, the storage of
   single Transport Sessions in IPFIX Files is particularly important
   for network measurement research, allowing repeatability of

experiments by providing a format for the storage and exchange of
IPFIX flow trace data much as the libpcap [pcap] format is used for
experiments on packet trace data.

6.2. Storage of NetFlow-V9-Collected Flow Data

Although the IPFIX protocol is based on the Cisco NetFlow Services,
Version 9 (NetFlow V9) protocol [RFC3954], the two have diverged
since work began on IPFIX. However, since the NetFlow V9 information
model is a compatible subset of the IPFIX Information Model, it is
possible to use IPFIX Files to store collected NetFlow V9 flow data.
This approach may be particularly useful in multi-vendor, multi-
protocol collection infrastructures using both NetFlow V9 and IPFIX
to export flow data.

The applicability of IPFIX Files to this use case is outlined in
Appendix B.

6.3. Testing IPFIX Collecting Processes

IPFIX Files can be used to store IPFIX Messages for the testing of
IPFIX Collecting Processes. A variety of test cases may be stored in
IPFIX Files. First, IPFIX data collected in real network
environments and stored in an IPFIX File can be used as input to
check the behavior of new or extended implementations of IPFIX
Collectors. Furthermore, IPFIX Files can be used to validate the
operation of a given IPFIX Collecting Process in a new environment,
i.e., to test with recorded IPFIX data from the target network before
installing the Collecting Process in the network.

The IPFIX File format can also be used to store artificial, non-
compliant reference messages for specific Collecting Process test
cases. Examples for such test cases are sets of IPFIX records with
undefined Information Elements, Data Records described by missing
Templates, or incorrectly framed Messages or Data Sets.
Representative error handling test cases are defined in [RFC5471].

Furthermore, fast replay of IPFIX Messages stored in a file can be
used for stress/load tests (e.g., high rate of incoming Data Records,
large Templates with high Information Element counts), as described
in [RFC5471]. The provisioning and use of a set of reference files
for testing simplifies the performance of tests and increases the
comparability of test results.

6.4. IPFIX Device Diagnostics

   As an IPFIX File can be used to store any collection of flows, the
   format may also be used for dumping and storing various types of flow
   data for IPFIX Device diagnostics (e.g., the open flow cache of a
   Metering Process or the flow backlog of an Exporting or Collecting
   Process at the time of a process reset or crash). File-based storage
   is preferable to remote transmission in such error-recovery
   situations.

7. Detailed File Format Specification

   Any valid serialized IPFIX Message stream MUST be accepted by a File
   Reader as a valid IPFIX File. In this way, the filesystem is simply
   treated as another IPFIX transport alongside SCTP, TCP, and UDP,
   albeit a potentially high-latency transport, as the File Reader and
   File Writer do not necessarily run at the same time.

   This section specifies the detailed actions of File Readers and File
   Writers in handling IPFIX Files, and further specifies actions of
   File Writers in specific use cases. Unless otherwise specified
   herein, IPFIX File Writers MUST behave as IPFIX Exporting Processes,
   and IPFIX File Readers MUST behave as IPFIX Collecting Processes,
   where appropriate.

7.1. File Reader Specification

   An IPFIX File Reader MUST act as an IPFIX Collecting Process as
   specified in [RFC5101], except as modified by this document.

   An IPFIX File Reader MUST accept as valid any serialized IPFIX
   Message stream that would be considered valid by one or more of the
   other defined IPFIX transport layers. Practically, this means that
   the union of IPFIX Template management features supported by SCTP,
   TCP, and UDP MUST be supported in IPFIX Files. File Readers MUST:

   o accept IPFIX Messages containing Template Sets, Options Template
      Sets, and Data Sets within the same message, as with IPFIX over
      TCP or UDP;

   o accept Template Sets that define Templates already defined within
      the File, as may occur with retransmission of Templates when using
      IPFIX over UDP as described in Section 10.3.6 of [RFC5101];

   o resolve any conflict between a resent definition and a previous
      definition by assuming that the new Template replaces the old, as
      consistent with Template expiration and ID reuse when using UDP at
      the IPFIX transport protocol; and

   o accept Template Withdrawals as described in Section 8 of
      [RFC5101], provided that the Template to be withdrawn is defined,
      as is the case with IPFIX over TCP and SCTP.

   Considering the filesystem-as-transport view, in the general case, an
   IPFIX File SHOULD be treated as containing a single Transport Session
   as defined by [RFC5101]. However, some applications may benefit from
   the ability to treat a collection of IPFIX Files as a single
   Transport Session; see especially Section 7.3.3 below. A File Reader
   MAY be configurable to treat a collection of Files as a single
   Transport Session. However, a File Reader MUST NOT treat a single
   IPFIX File as containing multiple Transport Sessions.

   If an IPFIX File uses the technique described in [RFC5473] AND all of
   the non-Options Templates in the File contain the commonPropertiesId
   Information Element, a File Reader MAY assume the set of
   commonPropertiesId definitions provides a complete table of contents
   for the File for searching purposes.

7.2. File Writer Specification

   An IPFIX File Writer MUST act as an IPFIX Exporting Process as
   specified in [RFC5101], except as modified by this document. This
   section contains specifications for IPFIX File Writers in all
   situations; specifications and recommendations for specific File
   Writer use cases are found in Section 7.3 below.

   File Writers SHOULD store the Templates and Options required to
   decode the data within the File itself, unless modified by the
   requirements of a specific use case in a subsection of Section 7.3.
   In this way, a single IPFIX File generally contains a single notional
   Transport Session as defined by [RFC5101].

   File Writers SHOULD emit each Template Set or Options Template Set to
   appear in the File before any Data Set described by the Templates
   within that Set, to ensure the File Reader can decode every Data Set
   without waiting to process subsequent Templates or Options Templates.

   File Writers SHOULD emit Data Records described by Options Templates
   to appear in the File before any Data Records that depend on the
   scopes defined by those options.

   File Writers SHOULD use Template Withdrawals to withdraw Templates if
   Template IDs need to be reused. Template Withdrawals SHOULD NOT be
   used unless it is necessary to reuse Template IDs.

   File Writers SHOULD write IPFIX Messages within an IPFIX File in
   ascending Export Time order.

File Writers MAY write Data Records to an IPFIX File in any order.
However, File Writers that write flow records to an IPFIX File in
flowStartTime or flowEndTime order SHOULD be consistent in this
ordering within each File.

7.3. Specific File Writer Use Cases

The specifications in this section apply to specific situations.
Each section below extends or modifies the base File Writer
specification in Section 7.2. Considerations for collocation of a
File Writer with IPFIX Collecting Processes and Metering Processes
are given, as are specific guidelines for using IPFIX Files for
archival storage, or as documents. Also covered are the use of IPFIX
Files in the testing and diagnostics of IPFIX Devices.

7.3.1. Collocating a File Writer with a Collecting Process

When collocating a File Writer with an IPFIX Collecting Process for
archival storage of collected data in IPFIX Files as described in
Section 6.1, the following recommendations may improve the usefulness
of the stored data.

The simplest way for a File Writer to store the data collected in a
single Transport Session is to simply write the incoming IPFIX
Messages to an IPFIX File as they are collected. This approach has
several drawbacks. First, if the original Exporting Process did not
conform to the recommendations in Section 7.2 with respect to
Template and Data Record ordering, the written file can be difficult
to use later; in this case, File Writers MAY reorder records as
received in order to ensure that Templates appear before the Data
Records they describe.

A File Writer collocated with a Collecting Process that starts
writing data from a running Transport Session SHOULD write all the
Templates currently active within that Transport Session before
writing any Data Records described by them.

Also, the resulting IPFIX Files will lack information about the IPFIX
Transport Session used to export them, such as the network addresses
of the Exporting and Collecting Processes and the protocols used to
transport them. In this case, if information about the Transport
Session is required, the File Writer SHOULD store a single IPFIX
Transport Session in an IPFIX File and SHOULD record information
about the Transport Session using the Export Session Details Options
Template described in Section 8.1.3.

Additional per-Message information MAY be recorded by the File Writer
using the Message Details Options Template described in
Section 8.1.4. Per-Message information includes the time at which
each IPFIX Message was received at the Collecting Process, and can be
used to resend IPFIX Messages while keeping the original measurement
plane traffic profile.

When collocating a File Writer with a Collecting Process, the Export
Time of each Message SHOULD be the Export Time of the Message
received by the Collecting Process containing the first Data Record
in the Message. Note that File Writers storing IPFIX data collected
from an IPFIX Collecting Process using SCTP as the transport protocol
SHOULD interleave messages from multiple streams in order to preserve
Export Time order, and SHOULD reorder the written messages as
necessary to ensure that each Template Set or Options Template Set
appears in the File before any Data Set described by the Templates
within that Set. Template reordering MUST preserve the sequence of
Template Sets with Template Withdrawals in order to ensure
consistency of Templates.

Note that when adding additional information to IPFIX Messages
received from Collecting Processes (e.g., Message Checksum Options,
Message Detail Options), the File Writer SHOULD extend the length of
the Message for the additional data if possible; otherwise, the
Message SHOULD be split into two approximately equal-size Messages
aligned on a Data Set or Template Set boundary from the original
Message if possible; otherwise, the Message SHOULD be split into two
approximately equal-size Messages aligned on a Data Record boundary.
Note that, since the Maximum Segment Size (MSS) or MTU of most
network links (1500-9000 for common Ethernets) is smaller than the
maximum IPFIX Message size (65536) within an IPFIX File, it is
expected that message length extension will suffice in most
circumstances.

A File Writer collocated with a Collecting Process SHOULD NOT sign a
File as specified in Section 9.1 unless the Transport Session over
which the data was exported was protected via TLS or DTLS, and the
Collecting Process positively identified the Exporting Process by its
certificate. See Section 12.2 for more information on this issue.

7.3.2. Collocating a File Writer with a Metering Process

Note that File Writers may also be collocated directly with IPFIX
Metering Processes, for writing measured information directly to disk
without intermediate IPFIX Exporting or Collecting Processes. This
arrangement may be particularly useful when providing data to an

analysis environment with an IPFIX-File-based workflow, when testing
Metering Processes during development, or when the authentication of
a Metering Process is important.

When collocating a File Writer with a Metering Process, note that
Information Elements associated with Exporting or Collecting
Processes are meaningless, and SHOULD NOT appear in the Export
Session Details Options Template described in Section 8.1.3 or the
Message Details Options Template described in Section 8.1.4.

When collocating a File Writer with a Metering Process, the Export
Time of each Message SHOULD be the time at which the first Data
Record in the Message was received from the Metering Process.

Note that collocating a File Writer with a Metering Process is the
only way to provide positive authentication of a Metering Process
through signatures as in Section 9.1. See Section 12.2 for more
information on this issue.

7.3.3. Using IPFIX Files for Archival Storage

While in the general case File Writers should store one Transport
Session per IPFIX File, some applications storing large collections
of data over long periods of time may benefit from the ability to
treat a collection of IPFIX Files as a single Transport Session. A
File Writer MAY be configurable to write data from a single Transport
Session into multiple IPFIX Files; however, File Writers supporting
such a configuration option MUST provide a configuration option to
support one-file-per-session behavior for interoperability purposes.

File Writers using IPFIX Files for archival storage SHOULD support
compression as in Section 10.

7.3.4. Using IPFIX Files as Documents

When IPFIX Files are used as documents, to store a set of flows
relevant to query, investigation, or other common context, or for the
publication of traffic datasets relevant to network research, each
File MUST be readable as a single Transport Session, self-contained
aside from any detached signature as in Section 9.1, and making no
reference to metadata stored in separate Files, in order to ensure
interoperability.

When writing Files to be used as documents, File Writers MAY emit the
special Data Records described by Options Templates before any other
Data Records in the File in the following order to ease the
inspection and use of documents by File Readers:

   o Time Window records described by the File Time Window Options
      Template as defined in Section 8.1.2 below; followed by:

   o Information Element Type Records as described in [RFC5610];
      followed by

   o commonPropertiesId definitions as described in [RFC5473]; followed
      by

   o Export Session details records described by the Export Session
      Details Options Template as defined in Section 8.1.3 below.

   The Export Time of each Message within a File used as a document
   SHOULD be the time at which the Message was written by the File
   Writer.

   If an IPFIX File used as a document uses the technique described in
   [RFC5473] AND all of the non-Options Templates in the File contain
   the commonPropertiesId Information Element, a File Reader MAY assume
   the set of commonPropertiesId definitions provides a complete table
   of contents for the File for searching purposes.

7.3.5. Using IPFIX Files for Testing

   IPFIX Files can be used for testing IPFIX Collecting Processes in two
   ways. First, IPFIX Files can be used to store specific flow data for
   regression and stress testing of Collectors; there are no special
   considerations for IPFIX Files used in this way.

   Second, IPFIX Files are useful for storing reference messages that do
   not comply to the IPFIX protocol in order to test the error handling
   and recovery behavior of Collectors. Of course, IPFIX Files intended
   to be used in this application necessarily MAY violate any of the
   specifications in this document or in [RFC5101], and such Files MUST
   NOT be transmitted to Collecting Processes or given as input to File
   Readers not under test.

   Note that an extremely simple IPFIX Exporting Process may be crafted
   for testing purposes by simply reading an IPFIX File and transmitting
   it directly to a Collecting Process. Similarly, an extremely simple
   Collecting Process may be crafted for testing purposes by simply
   accepting connections and/or IPFIX Messages from Exporting Processes
   and writing the session's message stream to an IPFIX File.

7.3.6. Writing IPFIX Files for Device Diagnostics

   IPFIX Files can be used in the debugging of devices that use flow
   data as internal state, as a common format for the representation of
   flow tables. In such situations, the opaqueOctets information
   element can be used to store additional non-IPFIX encoded, non-flow
   information (e.g., stack backtraces, process state, etc.) within the
   IPFIX File as in Section 11.1; the IPFIX flow table information could
   also be embedded in a larger proprietary diagnostic format using
   delimiters as in Section 11.2

7.3.7. IPFIX File Manipulation

   For many applications, it may prove useful for implementations to
   provide functionality for the manipulation of IPFIX Files; for
   example, to select data from a File, to change the Templates used
   within a File, or to split or join data in Files. Any such utility
   should take special care to ensure that its output remains a valid
   IPFIX File, specifically with respect to Templates and Options, which
   are scoped to Transport Sessions.

   Any operation that splits one File into multiple Files SHOULD write
   all necessary Templates and Options to each resulting File, and
   ensure that written Options are valid for each resulting File (e.g.,
   the Time Window Options Template in Section 8.1.2). Any operation
   that joins multiple Files into a single File should do the same,
   additionally ensuring that Template IDs do not collide, through the
   use of different Observation Domain IDs or Template ID rewriting.
   Combining operations may also want to ensure any desired ordering of
   flow records is maintained.

7.4. Media Type of IPFIX Files

   The media type for IPFIX Files is application/ipfix. The
   registration information [RFC4288] for this media type is given in
   the IANA Considerations section.

8. File Format Metadata Specification

   This section defines the Options Templates used for IPFIX File
   metadata, and the Information Elements they require.

8.1. Recommended Options Templates for IPFIX Files

   The following Options Templates allow IPFIX Message streams to meet
   the requirements outlined above without extension to the message
   format or protocol. They are defined in terms of existing
   Information Elements defined in [RFC5102], the Information Elements

defined in [RFC5610], as well as Information Elements defined in
Section 8.2. IPFIX File Readers and Writers SHOULD support these
Options Templates as defined below.

In addition, IPFIX File Readers and Writers SHOULD support the
Options Templates defined in [RFC5610] in order to support self-
description of enterprise-specific Information Elements.

8.1.1. Message Checksum Options Template

The Message Checksum Options Template specifies the structure of a
Data Record for attaching an MD5 message checksum to an IPFIX
Message. An MD5 message checksum as described MAY be used if data
integrity is important to the application but file signing is not
available or desired. The described Data Record MUST appear only
once per IPFIX Message, but MAY appear anywhere within the Message.

This Options Template SHOULD contain the following Information
Elements:

```
+-------------------+--------------------------------------------+
| IE                | Description                                |
+-------------------+--------------------------------------------+
| messageScope      | A marker denoting this Option applies to the |
| [scope]           | whole IPFIX Message; content is ignored.   |
|                   | This Information Element MUST be defined as |
|                   | a Scope Field.                             |
| messageMD5Checksum | The MD5 checksum of the containing IPFIX   |
|                   | Message.                                   |
+-------------------+--------------------------------------------+
```

8.1.2. File Time Window Options Template

The File Time Window Options Template specifies the structure of a
Data Record for attaching a time window to an IPFIX File; this Data
Record is referred to as a time window record. A time window record
defines the earliest flow start time and the latest flow end time of
the flow records within a File. One and only one time window record
MAY appear within an IPFIX File if the time window information is
available; a File Writer MUST NOT write more than one time window
record to an IPFIX File. A File Writer that writes a time window
record to a File MUST NOT write any Flow with a start time before the
beginning of the window or an end time after the end of the window to
that File.

This Options Template SHOULD contain the following Information
Elements:

```
+---------------+---------------------------------------------------+
| IE            | Description                                       |
+---------------+---------------------------------------------------+
| sessionScope  | A marker denoting this Option applies to the      |
| [scope]       | whole IPFIX Transport Session (i.e., the IPFIX    |
|               | File in the common case); content is ignored.     |
|               | This Information Element MUST be defined as a      |
|               | Scope Field.                                      |
| minFlowStart* | Exactly one of minFlowStartSeconds,               |
|               | minFlowStartMilliseconds,                         |
|               | minFlowStartMicroseconds, or                      |
|               | minFlowStartNanoseconds SHOULD match the          |
|               | precision of the accompanying maxFlowEnd*         |
|               | Information Element.  The start time of the       |
|               | earliest flow in the Transport Session (i.e.,     |
|               | File).                                            |
| maxFlowEnd*   | Exactly one of maxFlowEndSeconds,                 |
|               | maxFlowEndMilliseconds, maxFlowEndMicroseconds,   |
|               | or maxFlowEndNanoseconds SHOULD match the         |
|               | precision of the accompanying minFlowStart*       |
|               | Information Element.  The end time of the latest  |
|               | flow in the Transport Session (i.e., File).       |
+---------------+---------------------------------------------------+
```

8.1.3. Export Session Details Options Template

   The Export Session Details Options Template specifies the structure
   of a Data Record for recording the details of an IPFIX Transport
   Session in an IPFIX File. It is intended for use in storing a single
   complete IPFIX Transport Session in a single IPFIX File. The
   described Data Record SHOULD appear only once in a given IPFIX File.

   This Options Template SHOULD contain at least the following
   Information Elements, subject to applicability as noted on each
   Information Element:

IE	Description
sessionScope [scope]	A marker denoting this Option applies to the whole IPFIX Transport Session (i.e., the IPFIX File in the common case); content is ignored. This Information Element MUST be defined as a Scope Field.
exporterIPv4Address	IPv4 address of the IPFIX Exporting Process from which the Messages in this Transport Session were received. Present only for Exporting Processes with an IPv4 interface. For multi-homed SCTP associations, this SHOULD be the primary path endpoint address of the Exporting Process.
exporterIPv6Address	IPv6 address of the IPFIX Exporting Process from which the Messages in this Transport Session were received. Present only for Exporting Processes with an IPv6 interface. For multi-homed SCTP associations, this SHOULD be the primary path endpoint address of the Exporting Process.
exporterTransportPort	The source port from which the Messages in this Transport Session were received.
exporterCertificate	The certificate used by the IPFIX Exporting Process from which the Messages in this Transport Session were received. Present only for Transport Sessions protected by TLS or DTLS.
collectorIPv4Address	IPv4 address of the IPFIX Collecting Process that received the Messages in this Transport Session. Present only for Collecting Processes with an IPv4 interface. For multi-homed SCTP associations, this SHOULD be the primary path endpoint address of the Collecting Process.
collectorIPv6Address	IPv6 address of the IPFIX Collecting Process that received the Messages in this Transport Session. Present only for Collecting Processes with

```
|                           | an IPv6 interface.  For multi-homed |
|                           | SCTP associations, this SHOULD be   |
|                           | the primary path endpoint address of|
|                           | the Collecting Process.             |
| collectorTransportPort    | The destination port on which the   |
|                           | Messages in this Transport Session  |
|                           | were received.                      |
| collectorTransportProtocol| The IP Protocol Identifier of the   |
|                           | transport protocol used to transport|
|                           | Messages within this Transport      |
|                           | Session.                            |
| collectorProtocolVersion  | The version of the export protocol  |
|                           | used to transport Messages within   |
|                           | this Transport Session.  Applicable |
|                           | only in mixed NetFlow V9-IPFIX      |
|                           | collection environments when storing|
|                           | NetFlow V9 data in IPFIX Messages,  |
|                           | as in Appendix B.                   |
| collectorCertificate      | The certificate used by the IPFIX   |
|                           | Collecting Process that received the|
|                           | Messages in this Transport Session. |
|                           | Present only for Transport Sessions |
|                           | protected by TLS or DTLS.           |
| minExportSeconds          | The Export Time of the first Message|
|                           | in the Transport Session.           |
| maxExportSeconds          | The Export Time of the last Message |
|                           | in the Transport Session.           |
+---------------------------+-------------------------------------+
```

8.1.4. Message Details Options Template

   The Message Details Options Template specifies the structure of a
   Data Record for attaching additional export details to an IPFIX
   Message. These details include the time at which a message was
   received and information about the export and collection
   infrastructure used to transport the Message. This Options Template
   also allows the storage of the export session metadata provided the
   Export Session Details Options Template, for storing information from
   multiple Transport Sessions in the same IPFIX File.

   This Options Template SHOULD contain at least the following
   Information Elements, subject to applicability as noted for each
   Information Element. Note that when used in conjunction with the
   Export Session Details Options Template, when storing a single
   complete IPFIX Transport Session in an IPFIX File, this Options
   Template SHOULD contain only the messageScope and

collectionTimeMilliseconds Information Elements, and the
exportSctpStreamId Information Element for Messages transported via
SCTP.

IE	Description
messageScope [scope]	A marker denoting this Option applies to the whole IPFIX message; content is ignored. This Information Element MUST be defined as a Scope Field.
collectionTimeMilliseconds	The absolute time at which this Message was received by the IPFIX Collecting Process.
exporterIPv4Address	IPv4 address of the IPFIX Exporting Process from which this Message was received. Present only for Exporting Processes with an IPv4 interface, and if this information is not available via the Export Session Details Options Template. For multi-homed SCTP associations, this SHOULD be the primary path endpoint address of the Exporting Process.
exporterIPv6Address	IPv6 address of the IPFIX Exporting Process from which this Message was received. Present only for Exporting Processes with an IPv6 interface and if this information is not available via the Export Session Details Options Template. For multi-homed SCTP associations, this SHOULD be the primary path endpoint address of the Exporting Process.
exporterTransportPort	The source port from which this Message was received. Present only if this information is not available via the Export Session Details Options Template.
exporterCertificate	The certificate used by the IPFIX Exporting Process from which this Message was received. Present only for Transport Sessions protected by TLS or DTLS.
collectorIPv4Address	IPv4 address of the IPFIX Collecting Process that received this Message.

```
|                            | Present only for Collecting      |
|                            | Processes with an IPv4 interface,|
|                            | and if this information is not   |
|                            | available via the Export Session |
|                            | Details Options Template.  For   |
|                            | multi-homed SCTP associations, this |
|                            | SHOULD be the primary path endpoint |
|                            | address of the Collecting Process. |
| collectorIPv6Address       | IPv6 address of the IPFIX Collecting |
|                            | Process that received this Message. |
|                            | Present only for Collecting      |
|                            | Processes with an IPv6 interface,|
|                            | and if this information is not   |
|                            | available via the Export Session |
|                            | Details Options Template.  For   |
|                            | multi-homed SCTP associations, this |
|                            | SHOULD be the primary path endpoint |
|                            | address of the Collecting Process. |
| collectorTransportPort     | The destination port on which this |
|                            | Message was received.  Present only |
|                            | if this information is not available |
|                            | via the Export Session Details   |
|                            | Options Template.                |
| collectorTransportProtocol | The IP Protocol Identifier of the |
|                            | transport protocol used to transport |
|                            | this Message.  Present only if this |
|                            | information is not available via the |
|                            | Export Session Details Options   |
|                            | Template.                        |
| collectorProtocolVersion   | The version of the export protocol |
|                            | used to transport this Message.  |
|                            | Present only if necessary and if |
|                            | this information is not available |
|                            | via the Export Session Details   |
|                            | Options Template.                |
| collectorCertificate       | The certificate used by the IPFIX |
|                            | Collecting Process that received |
|                            | this Message.  Present only for  |
|                            | Transport Sessions protected by TLS |
|                            | or DTLS.                         |
| exportSctpStreamId         | The SCTP stream used to transport |
|                            | this Message.  Present only if the |
|                            | Message was transported via SCTP. |
+----------------------------+------------------------------------+
```

8.2. Recommended Information Elements for IPFIX Files

   The following Information Elements are used by the Options Templates
   in Section 8.1 to allow IPFIX Message streams to meet the
   requirements outlined above without extension of the protocol. IPFIX
   File Readers and Writers SHOULD support these Information Elements as
   defined below.

   In addition, IPFIX File Readers and Writers SHOULD support the
   Information Elements defined in [RFC5610] in order to support full
   self-description of Information Elements.

8.2.1. collectionTimeMilliseconds

   Description: The absolute timestamp at which the data within the
      scope containing this Information Element was received by a
      Collecting Process. This Information Element SHOULD be bound to
      its containing IPFIX Message via IPFIX Options and the
      messageScope Information Element, as defined below.

   Abstract Data Type: dateTimeMilliseconds

   ElementId: 258

   Status: current

8.2.2. collectorCertificate

   Description: The full X.509 certificate, encoded in ASN.1 DER
      format, used by the Collector when IPFIX Messages were transmitted
      using TLS or DTLS. This Information Element SHOULD be bound to
      its containing IPFIX Transport Session via an options record and
      the sessionScope Information Element, or to its containing IPFIX
      Message via an options record and the messageScope Information
      Element.

   Abstract Data Type: octetArray

   ElementId: 274

   Status: current

8.2.3. exporterCertificate

   Description: The full X.509 certificate, encoded in ASN.1 DER
      format, used by the Collector when IPFIX Messages were transmitted
      using TLS or DTLS. This Information Element SHOULD be bound to
      its containing IPFIX Transport Session via an options record and

the sessionScope Information Element, or to its containing IPFIX
Message via an options record and the messageScope Information
Element.

   Abstract Data Type: octetArray

   ElementId: 275

   Status: current

8.2.4. exportSctpStreamId

   Description: The value of the SCTP Stream Identifier used by the
      Exporting Process for exporting IPFIX Message data. This is
      carried in the Stream Identifier field of the header of the SCTP
      DATA chunk containing the IPFIX Message(s).

   Abstract Data Type: unsigned16

   Data Type Semantics: identifier

   ElementId: 259

   Status: current

8.2.5. maxExportSeconds

   Description: The absolute Export Time of the latest IPFIX Message
      within the scope containing this Information Element. This
      Information Element SHOULD be bound to its containing IPFIX
      Transport Session via IPFIX Options and the sessionScope
      Information Element.

   Abstract Data Type: dateTimeSeconds

   ElementId: 260

   Status: current

   Units: seconds

8.2.6. maxFlowEndMicroseconds

   Description: The latest absolute timestamp of the last packet
      within any Flow within the scope containing this Information
      Element, rounded up to the microsecond if necessary. This
      Information Element SHOULD be bound to its containing IPFIX
      Transport Session via IPFIX Options and the sessionScope

Information Element. This Information Element SHOULD be used only
in Transport Sessions containing Flow Records with microsecond-
precision (or better) timestamp Information Elements.

Abstract Data Type: dateTimeMicroseconds

ElementId: 268

Status: current

Units: microseconds

8.2.7. maxFlowEndMilliseconds

Description: The latest absolute timestamp of the last packet
within any Flow within the scope containing this Information
Element, rounded up to the millisecond if necessary. This
Information Element SHOULD be bound to its containing IPFIX
Transport Session via IPFIX Options and the sessionScope
Information Element. This Information Element SHOULD be used only
in Transport Sessions containing Flow Records with millisecond-
precision (or better) timestamp Information Elements.

Abstract Data Type: dateTimeMilliseconds

ElementId: 269

Status: current

Units: milliseconds

8.2.8. maxFlowEndNanoseconds

Description: The latest absolute timestamp of the last packet
within any Flow within the scope containing this Information
Element. This Information Element SHOULD be bound to its
containing IPFIX Transport Session via IPFIX Options and the
sessionScope Information Element. This Information Element SHOULD
be used only in Transport Sessions containing Flow Records with
nanosecond-precision timestamp Information Elements.

Abstract Data Type: dateTimeNanoseconds

ElementId: 270

Status: current

Units: nanoseconds

8.2.9. maxFlowEndSeconds

   Description: The latest absolute timestamp of the last packet
      within any Flow within the scope containing this Information
      Element, rounded up to the second if necessary. This Information
      Element SHOULD be bound to its containing IPFIX Transport Session
      via IPFIX Options and the sessionScope Information Element.

   Abstract Data Type: dateTimeSeconds

   ElementId: 261

   Status: current

   Units: seconds

8.2.10. messageMD5Checksum

   Description: The MD5 checksum of the IPFIX Message containing this
      record. This Information Element SHOULD be bound to its
      containing IPFIX Message via an options record and the
      messageScope Information Element, as defined below, and SHOULD
      appear only once in a given IPFIX Message. To calculate the value
      of this Information Element, first buffer the containing IPFIX
      Message, setting the value of this Information Element to all
      zeroes. Then calculate the MD5 checksum of the resulting buffer
      as defined in [RFC1321], place the resulting value in this
      Information Element, and export the buffered message. This
      Information Element is intended as a simple checksum only;
      therefore collision resistance and algorithm agility are not
      required, and MD5 is an appropriate message digest.

   Abstract Data Type: octetArray (16 bytes)

   ElementId: 262

   Status: current

   Reference: RFC 1321, The MD5 Message-Digest Algorithm [RFC1321]

8.2.11. messageScope

   Description: The presence of this Information Element as scope in
      an Options Template signifies that the options described by the
      Template apply to the IPFIX Message that contains them. It is
      defined for general purpose message scoping of options, and
      proposed specifically to allow the attachment of checksum and
      collection information to a message via IPFIX Options. The value

of this Information Element MUST be written as 0 by the File
Writer or Exporting Process. The value of this Information
Element MUST be ignored by the File Reader or the Collecting
Process.

   Abstract Data Type: unsigned8

   ElementId: 263

   Status: current

8.2.12. minExportSeconds

   Description: The absolute Export Time of the earliest IPFIX Message
      within the scope containing this Information Element. This
      Information Element SHOULD be bound to its containing IPFIX
      Transport Session via an options record and the sessionScope
      Information Element.

   Abstract Data Type: dateTimeSeconds

   ElementId: 264

   Status: current

   Units: seconds

8.2.13. minFlowStartMicroseconds

   Description: The earliest absolute timestamp of the first packet
      within any Flow within the scope containing this Information
      Element, rounded down to the microsecond if necessary. This
      Information Element SHOULD be bound to its containing IPFIX
      Transport Session via an options record and the sessionScope
      Information Element. This Information Element SHOULD be used only
      in Transport Sessions containing Flow Records with microsecond-
      precision (or better) timestamp Information Elements.

   Abstract Data Type: dateTimeMicroseconds

   ElementId: 271

   Status: current

   Units: microseconds

8.2.14. minFlowStartMilliseconds

   Description: The earliest absolute timestamp of the first packet
      within any Flow within the scope containing this Information
      Element, rounded down to the millisecond if necessary. This
      Information Element SHOULD be bound to its containing IPFIX
      Transport Session via an options record and the sessionScope
      Information Element. This Information Element SHOULD be used only
      in Transport Sessions containing Flow Records with millisecond-
      precision (or better) timestamp Information Elements.

   Abstract Data Type: dateTimeMilliseconds

   ElementId: 272

   Status: current

   Units: milliseconds

8.2.15. minFlowStartNanoseconds

   Description: The earliest absolute timestamp of the first packet
      within any Flow within the scope containing this Information
      Element. This Information Element SHOULD be bound to its
      containing IPFIX Transport Session via an options record and the
      sessionScope Information Element. This Information Element SHOULD
      be used only in Transport Sessions containing Flow Records with
      nanosecond-precision timestamp Information Elements.

   Abstract Data Type: dateTimeNanoseconds

   ElementId: 273

   Status: current

   Units: nanoseconds

8.2.16. minFlowStartSeconds

   Description: The earliest absolute timestamp of the first packet
      within any Flow within the scope containing this Information
      Element, rounded down to the second if necessary. This
      Information Element SHOULD be bound to its containing IPFIX
      Transport Session via an options record and the sessionScope
      Information Element.

   Abstract Data Type: dateTimeSeconds

      ElementId: 265

      Status: current

      Units: seconds

8.2.17. opaqueOctets

      Description: This Information Element is used to encapsulate non-
         IPFIX data into an IPFIX Message stream, for the purpose of
         allowing a non-IPFIX data processor to store a data stream inline
         within an IPFIX File. A Collecting Process or File Writer MUST
         NOT try to interpret this binary data. This Information Element
         differs from paddingOctets as its contents are meaningful in some
         non-IPFIX context, while the contents of paddingOctets MUST be
         0x00 and are intended only for Information Element alignment.

      Abstract Data Type: octetArray

      ElementId: 266

      Status: current

8.2.18. sessionScope

      Description: The presence of this Information Element as scope in
         an Options Template signifies that the options described by the
         Template apply to the IPFIX Transport Session that contains them.
         Note that as all options are implicitly scoped to Transport
         Session and Observation Domain, this Information Element is
         equivalent to a "null" scope. It is defined for general purpose
         session scoping of options, and proposed specifically to allow the
         attachment of time window and collection information to an IPFIX
         File via IPFIX Options. The value of this Information Element
         MUST be written as 0 by the File Writer or Exporting Process. The
         value of this Information Element MUST be ignored by the File
         Reader or the Collecting Process.

      Abstract Data Type: unsigned8

      ElementId: 267

      Status: current

9. Signing and Encryption of IPFIX Files

   In order to ensure the integrity of IPFIX Files and the identity of
   IPFIX File Writers, File Writers and File Readers SHOULD provide for
   an interoperable and easily implemented method for signing IPFIX
   Files, and verifying those signatures. This section specifies method
   via CMS detached signatures.

   Note that while CMS specifies an encapsulation format that can be
   used for encryption as well as signing, no method is specified for
   encapsulation for confidentiality protection. It is assumed that
   application-specific or process-specific requirements outweigh the
   need for interoperability for encrypted files.

9.1. CMS Detached Signatures

   The Cryptographic Message Syntax (CMS) [RFC3852] defines an
   encapsulation syntax for data protection, used to digitally sign,
   authenticate, or encrypt arbitrary message content. CMS can also be
   used to create detached signatures, in which the signature is stored
   in a separate file. This arrangement maximizes interoperability, as
   File Readers that are not aware of CMS detached signatures and have
   no requirement for them can simply ignore them; the content of the
   IPFIX File itself is unchanged by the signature.

   The detached signature file for an IPFIX File SHOULD be stored,
   transported, or otherwise made available (e.g., by FTP or HTTP)
   alongside the signed IPFIX File, with the same filename as the IPFIX
   File, except that the file extension ".p7s" is added to the end,
   conforming to the naming convention in [RFC3851].

   Within the detached signature, the CMS ContentInfo type MUST always
   be present, and it MUST encapsulate the CMS SignedData content type,
   which in turn MUST NOT encapsulate the signed IPFIX File content.
   The CMS detached signature is summarized as follows:

```
ContentInfo {
  contentType          id-signedData, -- (1.2.840.113549.1.7.2)
  content              SignedData
}

SignedData {
  version              CMSVersion, -- Always set to 3
  digestAlgorithms     DigestAlgorithmIdentifiers,
  encapContentInfo     EncapsulatedContentInfo,
  certificates         CertificateSet, -- File Writer certificate(s)
  crls                 CertificateRevocationLists, -- Optional
  signerInfos          SET OF SignerInfo -- Only one signer
}

SignerInfo {
  version              CMSVersion, -- Always set to 3
  sid                  SignerIdentifier,
  digestAlgorithm      DigestAlgorithmIdentifier,
  signedAttrs          SignedAttributes,
  signatureAlgorithm   SignatureAlgorithmIdentifier,
  signature            SignatureValue,
  unsignedAttrs        UnsignedAttributes
}

EncapsulatedContentInfo {
  eContentType         id-data, -- (1.2.840.113549.1.7.1)
  eContent             OCTET STRING  -- Always absent
}
```

The details of the contents of each CMS encapsulation are detailed in
the subsections below.

9.1.1. ContentInfo

   [RFC3852] requires the outer-most encapsulation to be ContentInfo;
   the fields of ContentInfo are as follows:

   contentType: the type of the associated content. For the detached
      signature file, the encapsulated type is always SignedData, so the
      id-signedData (1.2.840.113549.1.7.2) object identifier MUST be
      present in this field.

   content: a SignedData content, detailed in Section 9.1.2.

9.1.2. SignedData

   The SignedData content type contains the signature of the IPFIX File
   and information to aid in validation; the fields of SignedData are as
   follows:

   version: MUST be 3.

   digestAlgorithms: a collection of one-way hash function identifiers.
      It MUST contain the identifier used by the File Writer to generate
      the digital signature.

   encapContentInfo: the signed content, including a content type
      identifier. Since a detached signature is being created, it does
      not encapsulate the IPFIX File. The EncapsulatedContentInfo is
      detailed in Section 9.1.4.

   certificates: a collection of certificates. It SHOULD include the
      X.509 certificate needed to validate the digital signature file.
      Certification Authority (CA) and File Writer certificates MUST
      conform to the certificate profile specified in [RFC5280].

   crls: an optional collection of certificate revocation lists (CRLs).
      It SHOULD NOT contain any CRLs; any CRLs that are present MUST
      conform to the certificate profile specified in [RFC5280].

   signerInfos: a collection of per-signer information; this identifies
      the File Writer. More than one SignerInfo MAY appear to
      facilitate transitions between keys or algorithms. The SignerInfo
      type is detailed in Section 9.1.3.

9.1.3. SignerInfo

   The SignerInfo type identifies the File Writer; the fields of
   SignerInfo are as follows:

   version: MUST be 3.

   sid: identifies the File Writer's public key. This identifier MUST
      match the value included in the subjectKeyIdentifier certificate
      extension on the File Writer's X.509 certificate.

   digestAlgorithm: identifies the one-way hash function and associated
      parameters used to generate the signature.

   signedAttrs: an optional set of attributes that are signed along
      with the content.

   digestAlgorithm: identifies the digital signature algorithm and
      associated parameters used to generate the signature.

   signature: the digital signature of the associated file.

   unsignedAttrs: an optional set of attributes that are not signed.

9.1.4. EncapsulatedContentInfo

   The EncapsulatedContentInfo structure contains a content type
   identifier. Since a detached signature is being created, it does not
   encapsulate the IPFIX File. The fields of EncapsulatedContentInfo
   are as follows:

   eContentType: an object identifier that uniquely specifies the
      content type. The content type associated with IPFIX File MUST be
      id-data (1.2.840.113549.1.7.1).

   eContent: an optional field containing the signed content. Since
      this is a detached signature, eContent MUST be absent.

9.2. Encryption Error Resilience

   Note that single bit errors in the encrypted data stream can result
   in larger errors in the decrypted stream, depending on the encryption
   scheme used.

   In applications (e.g., archival storage) in which error resilience is
   very important, File Writers SHOULD use an encryption scheme that can
   resynchronize after bit errors. A common example is a block cipher
   in CBC (Cipher Block Chaining) mode. In this case, File Writers MAY
   also use the Message Checksum Options Template to attach a checksum
   to each IPFIX Message in the IPFIX File, in order to support the
   recognition of errors in the decrypted data.

10. Compression of IPFIX Files

   Network traffic measurement data is generally highly compressible.
   IPFIX Templates tend to increase the information content per record
   by not requiring the export of irrelevant or non-present fields in
   records, and the technique described in [RFC5473] also reduces the
   export of redundant information. However, even with these
   techniques, generalized compression can decrease storage requirements
   significantly; therefore, IPFIX File Writers and File Readers SHOULD
   support compression as described in this section.

10.1. Supported Compression Formats

   IPFIX Files support two compression encapsulation formats: bzip2
   [bzip2] and gzip [RFC1952]. bzip2 provides better compression than
   gzip and, as a block compression algorithm, better error recovery
   characteristics, at the expense of slower compression. gzip is
   potentially a better choice when compression time is an issue. These
   two algorithms and encapsulation formats were chosen for ubiquity and
   ease of implementation.

   IPFIX File Readers and Writers supporting compression MUST support
   bzip2, and SHOULD support gzip.

10.2. Compression Recognition at the File Reader

   bzip2, gzip, and uncompressed IPFIX Files have distinct magic
   numbers. IPFIX File Readers SHOULD use these magic numbers to
   determine what compression, if any, is in use for an IPFIX File, and
   invoke the proper decompression. bzip2 files are identified by the
   initial three-octet string 0x42, 0x5A, 0x68 ("BZh"). gzip files are
   identified by the initial two-octet string 0x1F, 0x8B. IPFIX Files
   are identified by the initial two-octet string 0x00, 0x0A; these are
   the version bytes of the first IPFIX Message header in the File.

10.3. Compression Error Resilience

   Compression at the file level, like encryption, is not particularly
   resilient to errors; in the worst case, a single bit error in a
   stream-compressed file could result in the loss of the entire file.

   Since block compression algorithms that support the identification
   and isolation of blocks containing errors limit the impact of errors
   on the recoverability of compressed data, the use of bzip2 in
   applications where error resilience is important is RECOMMENDED.

   Since the block boundary of a block-compressed IPFIX File may fall in
   the middle of an IPFIX Message, resynchronization of an IPFIX Message
   stream by a File Reader after a compression error requires some care.
   The beginning of an IPFIX Message may be identified by its header
   signature (the Version field of the Message Header, 0x00 0x0A,
   followed by a 16-bit Message Length), but simply searching for the
   first occurrence of the Version field is insufficient, since these
   two bytes may occur in valid IPFIX Template or Data Sets.

   Therefore, we specify the following algorithm for File Readers to
   resynchronize an IPFIX Message Stream after skipping a compressed
   block containing errors:

1. Search after the error for the first occurrence of the octet
    string 0x00, 0x0A (the IPFIX Message Header Version field).

2. Treat this field as the beginning of a candidate IPFIX Message.
    Read the two bytes following the Version field as a Message
    Length, and seek to that offset from the beginning of the
    candidate IPFIX Message.

3. If the first two octets after the candidate IPFIX Message are
    0x00, 0x0A (i.e., the IPFIX Message Header Version field of the
    next message in the stream), or if the end-of-file is reached
    precisely at the end of the candidate IPFIX Message, presume that
    the candidate IPFIX Message is valid, and begin reading the IPFIX
    File from the start of the candidate IPFIX Message.

4. If not, or if the seek reaches end-of-file or another block
    containing errors before finding the end of the candidate
    message, go back to step 1, starting the search two bytes from
    the start of the candidate IPFIX Message.

The algorithm above will improperly identify a non-message as a
message approximately 1 in 2^32 times, assuming random IPFIX data.
It may be expanded to consider multiple candidate IPFIX Messages in
order to increase reliability.

In applications (e.g., archival storage) in which error resilience is
very important, File Writers SHOULD use block compression algorithms,
and MAY attempt to align IPFIX Messages within compression blocks to
ease resynchronization after errors. File Readers SHOULD use the
resynchronization algorithm above to minimize data loss due to
compression errors.

11. Recommended File Integration Strategies

   This section describes methods for integrating IPFIX File data with
   other file formats.

11.1. Encapsulation of Non-IPFIX Data in IPFIX Files

   At times, it may be useful to export or store non-IPFIX data inline
   in an IPFIX File or Message stream. To do this cleanly, this data
   must be encapsulated into IPFIX Messages so that an IPFIX File Reader
   or Collecting Process can handle it without any need to interpret it.
   At the same time, this data must not be changed during transmission
   or storage. The opaqueOctets Information Element, as defined in
   Section 8.2.17, is provided for this encapsulation.

   Processing the encapsulated non-IPFIX data is left to a separate
   processing mechanisms that can identify encapsulated non-IPFIX data
   in an IPFIX Message Stream, but need not have any other IPFIX
   handling capability, except the ability to skip over all IPFIX
   Messages that do not encapsulate non-IPFIX data.

   The Message Checksum Options Template, described in Section 8.1.1,
   may be used as a uniform mechanism to identify errors within
   encapsulated data.

   Note that this mechanism can only encapsulate data objects up to
   65,515 octets in length. If the space available in one IPFIX Message
   is not enough for the amount of data to be encapsulated, then the
   data must be broken into smaller segments that are encapsulated into
   consecutive IPFIX Messages. Any additional structuring or semantics
   of the raw data is outside the scope of IPFIX and must be implemented
   within the encapsulated binary data itself. Furthermore, the raw
   encapsulated data cannot be assumed by an IPFIX File Reader to have
   any specific format.

11.2. Encapsulation of IPFIX Files within Other File Formats

   Consequently, it may also be useful to reverse the encapsulation,
   that is, to export or store IPFIX data inline within a non-IPFIX File
   or data stream. This makes sense when the other file format is not
   compatible with the encapsulation described above in Section 11.1.
   Generally speaking, the encapsulation here will be specific to the
   format of the containing file. For example, IPFIX Files may be
   embedded in XML elements using hex or Base64 encoding, or in raw
   binary files using start and end delimiters or some form of run-
   length encoding. As there are as many potential encapsulations here
   as there are potential file formats, the specifics of each are out of
   scope for this specification.

12. Security Considerations

   The Security Considerations section of [RFC5101], on which the IPFIX
   File format is based, is largely concerned with the proper
   application of TLS and DTLS to ensure confidentiality and integrity
   when exporting IPFIX Messages. By analogy, this document specifies
   the use of CMS [RFC3852] detached signatures to provide equivalent
   integrity protection to TLS and DTLS in Section 9.1. However, aside
   from merely applying CMS for signatures, there are several security
   issues which much be considered in certain circumstances; these are
   covered in the subsections below.

12.1. Relationship between IPFIX File and Transport Encryption

   The underlying protocol used to exchange the information that will be
   stored using the format proposed in this document must as well apply
   appropriate procedures to guarantee the integrity and confidentiality
   of the exported information. Such issues are addressed in [RFC5101].
   Specifically, IPFIX Files that store data taken from an IPFIX
   Collecting Process using TLS or DTLS for transport security SHOULD be
   signed as in Section 9.1 and SHOULD be encrypted out of band; storage
   of such flow data without encryption may present a potential breach
   of confidentiality. Conversely, flow data considered sensitive
   enough to require encryption in storage that is later transmitted
   using IPFIX SHOULD be transmitted using TLS or DTLS for transport
   security.

12.2. End-to-End Assertions for IPFIX Files

   Note that while both TLS and CMS provide the ability to sign an IPFIX
   Transport Session or an IPFIX File, there exists no method for
   protecting data integrity end-to-end in the case in which a
   Collecting Process is collocated with a File Writer. The channel
   between the Exporting Process to Collecting Process using IPFIX is
   signed by the Exporting Process key and protected via TLS and DTLS,
   while the File is signed by the File Writer key and protected via
   CMS. The identity of the Exporting Process is not asserted in the
   file, and the records may be modified between the Collecting Process
   and the File Writer.

   There are two potential ways to address this issue. The first is by
   fiat, and is appropriate only when the application allows the
   Collecting-Process-to-File-Writer channel to be trusted. In this
   case, the File Writer's signature is an implicit assertion that the
   channel to the Exporting Process was protected, that the Exporting
   Process's signature was verified, and that the data was not changed
   after collection. For this to work, a File Writer collocated with a
   Collecting Process SHOULD NOT sign a File as specified in Section 9.1
   unless the Transport Session over which the data was exported was
   protected via TLS or DTLS, and the Collecting Process positively
   identified the Exporting Process by its certificate. The File Writer
   SHOULD include the Exporting Process and Collecting Process
   certificates within the File using the Export Session Detail Options
   Template in Section 8.1.3 or the Message Detail Options Template in
   Section 8.1.4 to allow for later verification.

   In situations in which the Collecting Process and/or File Writer
   cannot be trusted, end-to-end integrity can then be approximated by
   collocating the File Writer with the Metering Process, and removing
   the IPFIX protocol completely from the chain. In this case, the File

   Writer's signature is an implicit assertion that the Metering Process
   is identified and is not tampering with the information as observed
   on the wire.

   Verification of these trust relationships is out of scope for this
   document, and should be considered on a per-implementation basis.

12.3. Recommendations for Strength of Cryptography for IPFIX Files

   Note that when encrypting files for archival storage, the
   cryptographic strength is dependent on the length of time over which
   archival data is expected to be kept. Long-term storage may require
   re-application of cryptographic protection, periodically resigning
   and reencrypting files with stronger keys. In this case, it is
   recommended that the existing signed and/or encpyted data be
   encapsulated within newer, stronger protection. See [RFC4810] for a
   discussion of this issue.

13. IANA Considerations

   This document specifies the creation of several new IPFIX Information
   Elements in the IPFIX Information Element registry located at
   http://www.iana.org, as defined in Section 8.2 above. IANA has
   assigned the following Information Element numbers for their
   respective Information Elements as specified below:

   o Information Element number 258 for the collectionTimeMilliseconds
      Information Element.

   o Information Element number 274 for the collectorCertificate
      Information Element.

   o Information Element number 275 for the exporterCertificate
      Information Element.

   o Information Element number 259 for the exportSctpStreamId
      Information Element.

   o Information Element number 260 for the maxExportSeconds
      Information Element.

   o Information Element number 268 for the maxFlowEndMicroseconds
      Information Element.

   o Information Element number 269 for the maxFlowEndMilliseconds
      Information Element.

   o Information Element number 270 for the maxFlowEndNanoseconds
      Information Element.

   o Information Element number 261 for the maxFlowEndSeconds
      Information Element.

   o Information Element number 262 for the messageMD5Checksum
      Information Element.

   o Information Element number 263 for the messageScope Information
      Element.

   o Information Element number 264 for the minExportSeconds
      Information Element.

   o Information Element number 271 for the minFlowStartMicroseconds
      Information Element.

   o Information Element number 272 for the minFlowStartMilliseconds
      Information Element.

   o Information Element number 273 for the minFlowStartNanoseconds
      Information Element.

   o Information Element number 265 for the minFlowStartSeconds
      Information Element.

   o Information Element number 266 for the opaqueOctets Information
      Element.

   o Information Element number 267 for the sessionScope Information
      Element.

   IANA has created the media type application/ipfix for IPFIX data, as
   described by the following registration information:

   Type name: application

   Subtype name: ipfix

   Required parameters: none

   Optional parameters: none

   Encoding considerations: IPFIX Files are binary, and therefore must
      be encoded in non-binary contexts.

   Security considerations: See the Security Considerations
      (Section 12) of RFC 5655, and the Security Considerations of
      [RFC5101].

   Interoperability considerations: See the "Detailed Specification"
      (Section 7) of RFC 5655. The format is designed to be broadly
      interoperable, as any valid stream of IPFIX Messages over any
      transport specified in [RFC5101] MUST be recognizable as a valid
      IPFIX File.

   Published specification: RFC 5655, especially Section 7, and
      [RFC5101].

   Applications that use this media type: Various IPFIX
      implementations (see [RFC5153]) support the construction of IPFIX
      File Readers and Writers.

   Additional information:

      Magic number(s):   None, although the first two bytes of any IPFIX
         File are the first two bytes of a message header, the Version
         field, which as of [RFC5101] are always 10 in network byte
         order: 0x00, 0x0A.

      File extension(s):   .ipfix

      Macintosh file type code(s):   none

   Person & email address to contact for further information: Brian
      Trammell <brian.trammell@hitachi-eu.com> for the authors of RFC
      5655; Nevil Brownlee <n.brownlee@auckland.ac.nz> for the IPFIX
      Working Group.

   Intended usage: LIMITED USE

   Restrictions on usage: none

   Change controller: Brian Trammell <brian.trammell@hitachi-eu.com>
      for the authors of RFC 5655; Nevil Brownlee
      <n.brownlee@auckland.ac.nz> for the IPFIX Working Group.

14. Acknowledgements

   Thanks to Maurizio Molina, Tom Kosnar, and Andreas Kind for technical
   assistance with the requirements for a standard flow storage format.
   Thanks to Benoit Claise, Paul Aitken, Andrew Johnson, Gerhard Muenz,
   and Nevil Brownlee for their reviews and feedback. Thanks to Pasi
   Eronen for pointing out [RFC5485], and Russ Housley for writing it;

it specifies a detached signature format, from which Section 9.1 is
largely drawn. Thanks to the PRISM project for its support of this
work.

15. References

15.1. Normative References

   [RFC5101]   Claise, B., "Specification of the IP Flow Information
                Export (IPFIX) Protocol for the Exchange of IP Traffic
                Flow Information", RFC 5101, January 2008.

   [RFC5102]   Quittek, J., Bryant, S., Claise, B., Aitken, P., and J.
                Meyer, "Information Model for IP Flow Information
                Export", RFC 5102, January 2008.

   [RFC5610]   Boschi, E., Trammell, B., Mark, L., and T. Zseby,
                "Exporting Type Information for IP Flow Information
                Export (IPFIX) Information Elements", RFC 5610,
                July 2009.

   [RFC1321]   Rivest, R., "The MD5 Message-Digest Algorithm",
                RFC 1321, April 1992.

   [RFC1952]   Deutsch, P., Gailly, J-L., Adler, M., Deutsch, L., and
                G. Randers-Pehrson, "GZIP file format specification
                version 4.3", RFC 1952, May 1996.

   [RFC2119]   Bradner, S., "Key words for use in RFCs to Indicate
                Requirement Levels", BCP 14, RFC 2119, March 1997.

   [RFC3852]   Housley, R., "Cryptographic Message Syntax (CMS)",
                RFC 3852, July 2004.

   [RFC4810]   Wallace, C., Pordesch, U., and R. Brandner, "Long-Term
                Archive Service Requirements", RFC 4810, March 2007.

   [RFC5280]   Cooper, D., Santesson, S., Farrell, S., Boeyen, S.,
                Housley, R., and W. Polk, "Internet X.509 Public Key
                Infrastructure Certificate and Certificate Revocation
                List (CRL) Profile", RFC 5280, May 2008.

   [bzip2]   Seward, J., "bzip2 (http://www.bzip.org/)", March 2008.

15.2. Informative References

   [RFC3917]   Quittek, J., Zseby, T., Claise, B., and S. Zander,
                "Requirements for IP Flow Information Export (IPFIX)",
                RFC 3917, October 2004.

   [RFC3954]   Claise, B., "Cisco Systems NetFlow Services Export
                Version 9", RFC 3954, October 2004.

   [RFC5153]   Boschi, E., Mark, L., Quittek, J., Stiemerling, M., and
                P. Aitken, "IP Flow Information Export (IPFIX)
                Implementation Guidelines", RFC 5153, April 2008.

   [RFC5470]   Sadasivan, G., Brownlee, N., Claise, B., and J. Quittek,
                "Architecture for IP Flow Information Export", RFC 5470,
                March 2009.

   [RFC5471]   Schmoll, C., Aitken, P., and B. Claise, "Guidelines for
                IP Flow Information Export (IPFIX) Testing", RFC 5471,
                March 2009.

   [RFC5472]   Zseby, T., Boschi, E., Brownlee, N., and B. Claise, "IP
                Flow Information Export (IPFIX) Applicability",
                RFC 5472, March 2009.

   [RFC5473]   Boschi, E., Mark, L., and B. Claise, "Reducing
                Redundancy in IP Flow Information Export (IPFIX) and
                Packet Sampling (PSAMP) Reports", RFC 5473, March 2009.

   [SAINT2007] Trammell, B., Boschi, E., Mark, L., and T. Zseby,
                "Requirements for a standardized flow storage solution",
                 in Proceedings of the SAINT 2007 workshop on Internet
                Measurement Technology, Hiroshima, Japan, January 2007.

   [RFC3851]   Ramsdell, B., "Secure/Multipurpose Internet Mail
                Extensions (S/MIME) Version 3.1 Message Specification",
                RFC 3851, July 2004.

   [RFC4288]   Freed, N. and J. Klensin, "Media Type Specifications and
                Registration Procedures", BCP 13, RFC 4288,
                December 2005.

   [RFC5485]   Housley, R., "Digital Signatures on Internet-Draft
                Documents", RFC 5485, March 2009.

   [pcap]   "libpcap (http://www.tcpdump.org/)", October 2008.

Appendix A. Example IPFIX File

   In this section we will explore an example IPFIX File that
   demonstrates the various features of the IPFIX File format. This
   File contains flow records described by a single Template. This File
   also contains a File Time Window record to note the start and end
   time of the data, and an Export Session Details record to record
   collection infrastructure information. Each Message within this File
   also contains a Message Checksum record, as this File may be
   externally encrypted and/or stored as an archive. The structure of
   this File is shown in Figure 2.

```
         +================================================+
         | IPFIX Message                        seq. 0    |
         | +--------------------------------------------+ |
         | | Template Set (ID 2)              1 rec   | |
         | |    Data Tmpl. ID 256                       | |
         | +--------------------------------------------+ |
         | | Options Template Set (ID 3)      3 recs  | |
         | |    File Time Window Opt. Tmpl. ID 257      | |
         | |    Message Checksum Opt. Tmpl. ID 259      | |
         | |    Export Session Details Opt. Tmpl. ID 258| |
         | +--------------------------------------------+ |
         | | Data Set (ID 259) [Message Checksum] 1 rec | |
         | +--------------------------------------------+ |
         +================================================+
         | IPFIX Message                        seq. 1    |
         | +--------------------------------------------+ |
         | | Data Set (ID 257) [File Time Window] 1 rec | |
         | +--------------------------------------------+ |
         | | Data Set (ID 258) [Export Session]   1 rec | |
         | +--------------------------------------------+ |
         | | Data Set (ID 259) [Message Checksum] 1 rec | |
         | +--------------------------------------------+ |
         +================================================+
         | IPFIX Message                        seq. 4    |
         | +--------------------------------------------+ |
         | | Data Set (ID 256)                50 recs | |
         | |  contains flow data                        | |
         | +--------------------------------------------+ |
         | | Data Set (ID 259) [Message Checksum] 1 rec | |
         | +--------------------------------------------+ |
         +================================================+
         | IPFIX Message                        seq. 55   |
         |                      . . .                     |
```

                     Figure 2: File Example Structure

The Template describing the data records contains a flow start
timestamp, an IPv4 5-tuple, and packet and octet total counts. The
Template Set defining this is as shown in Figure 3 below:

```
                        1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |          Set ID = 2           |          Length =  40         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |       Template ID = 256       |         Field Count = 8       |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |0| flowStartSeconds    = 150 |       Field Length =  4         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |0| sourceIPv4Address    =   8 |       Field Length =  4         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |0| dest.IPv4Address     =  12 |       Field Length =  4         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |0| sourceTransportPort  =   7 |       Field Length =  2         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |0| dest.TransportPort   =  11 |       Field Length =  2         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |0| protocolIdentifier   =   4 |       Field Length =  1         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |0| octetTotalCount      =  85 |       Field Length =  4         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |0| packetTotalCount     =  86 |       Field Length =  4         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

                   Figure 3: File Example Data Template

A.1.   Example Options Templates

   This is followed by an Options Template Set containing the Options
   Templates required to read the File: the File Time Window Options
   Template (defined in Section 8.1.2 above), the Export Session Details
   Options Template (defined in Section 8.1.3 above), and the Message
   Checksum Options Template (defined in Section 8.1.1 above). This
   Options Template Set is shown in Figure 4 and Figure 5 below:

```
                        1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |           Set ID = 3          |           Length =  80        |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |        Template ID = 257      |          Field Count = 3      |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |     Scope Field Count = 1     |0| sessionScope       = 267 |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |       Field Length = 1        |0| minFlowStartSeconds  = 265 |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |       Field Length = 4        |0| maxFlowEndSeconds    = 261 |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |       Field Length = 4        |        Template ID = 259      |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |        Field Count = 2        |      Scope Field Count = 1    |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |0| messageScope       = 263 |       Field Length =  1       |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |0| messageMD5Checksum   = 262 |       Field Length = 16       |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

       Figure 4: File Example Options Templates (Time Window and Checksum)

```
                            1                   2                   3
        0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |         Template ID = 258     |           Field Count = 9     |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |     Scope Field Count = 1     |0| sessionScope      = 267 |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |        Field Length =  1      |0| exporterIPv4Address   = 130 |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |        Field Length =  4      |0| collectorIPv4Address  = 211 |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |        Field Length =  4      |0| exporterTransportPort = 217 |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |        Field Length =  2      |0| col.TransportPort     = 216 |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |        Field Length =  2      |0| col.TransportProtocol = 215 |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |        Field Length =  1      |0| col.ProtocolVersion   = 214 |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |        Field Length =  1      |0| minExportSeconds      = 264 |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |        Field Length =  4      |0| maxExportSeconds      = 260 |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |        Field Length =  4      |      set padding (2 octets)   |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   Figure 5: File Example Options Templates, Continued (Session Details)

A.2. Example Supplemental Options Data

   Following the Templates required to decode the File is the
   supplemental IPFIX Options information used to describe the File's
   contents and type information. First comes the File Time Window
   record; it notes that the File contains data from 9 October 2007
   between 00:01:13 and 23:56:27 UTC, and appears as in Figure 6:

```
                            1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |          Set ID = 257         |           Length =  13        |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     | sessionScope  |           minFlowStartSeconds
     |       0       |           2007-10-09 00:01:13 UTC           . . .
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                     |           maxFlowEndSeconds
     . . .           |           2007-10-09 23:56:27 UTC           . . .
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                     |
     . . .           |
     +-+-+-+-+-+-+-+-+
```

                 Figure 6: File Example Time Window

This is followed by information about how the data in the File was
collected, in the Export Session Details record. This record notes
that the session stored in this File was sent via SCTP from an
Exporter at 192.0.2.30 port 32769 to a Collector at 192.0.2.40 port
4739, and contains messages exported between 00:01:57 and 23:57:12
UTC on 9 October 2007; it is represented in its Data Set as in
Figure 7:

```
                          1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |           Set ID = 258        |           Length =   27       |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     | sessionScope  |           exporterIPv4Address
     |      0        |              192.0.2.30                 . . .
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                     |           collectorIPv4Address
     . . .           |              192.0.2.31                 . . .
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                     |      exporterTransportPort    |   cTPort
     . . .           |            32769              |   4739   . . .
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                     |    cTProtocol | cPVersion     |
     . . .           |       132     |    10         |          . . .
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
               minExportSeconds                      |
     . . .     2007-10-09 00:01:57 UTC               |          . . .
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
               maxExportSeconds                      |
     . . .     2007-10-09 23:57:12 UTC               |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

                 Figure 7: File Example Export Session Details

A.3. Example Message Checksum

   Each IPFIX Message within the File is completed with a Message
   Checksum record; the structure of this record within its Data Set is
   as in Figure 8:

```
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |           Set ID = 259        |           Length =   24       |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     | messageScope  |                                               |
     |      0        |                                               |
     +-+-+-+-+-+-+-+-+                                               |
     |                     messageMD5Checksum                       |
     |          (16-byte MD5 checksum of options message)           |
     |                                                              |
     |                                                              |
     |               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |               |             set padding (3 octets)           |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

                   Figure 8: File Example Message Checksum

A.4. File Example Data Set

   After the Templates and supplemental Options information comes the
   data itself. The first record of an example Data Set is shown with
   its message and set headers in Figure 9:

```
                        1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |      Version = 10             |           Length = 1296        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |            Export Time = 2007-10-09 00:01:57 UTC               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                     Sequence Number = 4                        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                   Observation Domain ID = 1                    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |        Set ID = 256           |          Length = 1254         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                        flowStartSeconds                        |
   |                    2007-10-09 00:01:13 UTC                     |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                        sourceIPv4Address                       |
   |                          192.0.2.2                            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                     destinationIPv4Address                     |
   |                          192.0.2.3                            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |      sourceTransportPort      |   destinationTransportPort     |
   |            32770              |             80                 |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | protocolId    |               totalOctetCount                  |
   |     6         |                  18000                    . . . |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                   |               totalPacketCount                  |
   . . .           |                  65                       . . . |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                   |                (49 more records)                |
   . . .           |
   +-+-+-+-+-+-+-+-+
```

                     Figure 9: File Example Data Set

A.5. Complete File Example

   Bringing together the examples above and adding message headers as
   appropriate, a hex dump of the first 317 bytes of the example File
   constructed above would appear as in the annotated Figure 10 below.

```
   0:|00 0A 00 A0 47 0A B6 E5 00 00 00 00 00 00 00 01
     [^ first message header (length 160 bytes) -->
  16:|00 02 00 28 01 00 00 08 00 96 00 04 00 08 00 04
     [^ data template set -->
  32: 00 0C 00 04 00 07 00 02 00 0B 00 02 00 04 00 01

  48: 00 55 00 04 00 56 00 04|00 03 00 50 01 01 00 03
                             [^ opt template set -->
  64: 00 01 01 0B 00 01 01 09 00 04 01 05 00 04 01 03

  80: 00 02 00 01 01 07 00 01 01 06 00 10 01 02 00 09

  96: 00 01 01 0B 00 01 00 82 00 04 00 D3 00 04 00 D9

 112: 00 02 00 D8 00 02 00 D7 00 01 00 D0 00 01 01 08

 128: 00 04 01 04 00 04 00 00|01 03 00 18 00 73 F1 12
                             [^ checksum record -->
 144: D6 C7 58 BE 44 E6 60 06 4E 78 74 AE 7D 00 00 00

 176:|00 0A 00 50 47 0A B6 E5 00 00 00 01 00 00 00 01
     [^ second message header (length 80 bytes) -->
 192:|01 01 00 0E 00 47 0A B6 B9 47 0C 07 1B 00|01 02
     [^ time window rec -> [ session detail rec ^ -->
 208: 00 1C 00 C0 00 02 1E 0C 00 02 1F 80 01 12 83 84

 224: 0A 47 0A B6 E5 47 0C 07 48 00|01 03 00 18 00 3E
          [ message checksum record ^ -->
 240: 2B 37 08 CE B2 0E 30 11 32 12 4A 5F E3 AD DB 00

 256:|00 0A 05 10 47 0A B6 E5 00 00 00 06 00 00 00 01
     [^ third message header (length 1296 bytes) -->
 272:|01 00 04 E6|47 0A B6 B9 C0 00 02 02 C0 00 02 03
     [^ set hdr ][^ first data rec -->
 288: 80 02 00 50 06 00 00 46 50 00 00 00 41
```

                    Figure 10: File Example Hex Dump

Appendix B. Applicability of IPFIX Files to NetFlow V9 Flow Storage

   As the IPFIX Message format is nearly a superset of the NetFlow V9
   packet format, IPFIX Files can be used for store NetFlow V9 data
   relatively easily. This section describes a method for doing so.
   The differences between the two protocols are outlined in
   Appendix B.1 below. A simple, lightweight, message-for-message
   translation method for transforming V9 Packets into IPFIX Messages
   for storage within IPFIX Files is described in Appendix B.2. An
   example of this translation method is given in Appendix B.3.

B.1. Comparing NetFlow V9 to IPFIX

   With a few caveats, the IPFIX protocol is a superset of the NetFlow
   V9 protocol, having evolved from it largely through a process of
   feature addition to bring it into compliance with the IPFIX
   Requirements and the needs of stakeholders within the IPFIX Working
   Group. This appendix outlines the differences between the two
   protocols. It is informative only, and presented as an exploration
   of the two protocols to motivate the usage of IPFIX Files to store
   V9-collected flow data.

B.1.1. Message Header Format

   Both NetFlow V9 and IPFIX use streams of messages prefixed by a
   message header, though the message header differs significantly
   between the two. Note that in NetFlow V9 terminology, these messages
   are called packets, and messages must be delimited by datagram
   boundaries. IPFIX does not have this constraint. The header formats
   are detailed below:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Version Number          |             Count             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           sysUpTime                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           UNIX Secs                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Source ID                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
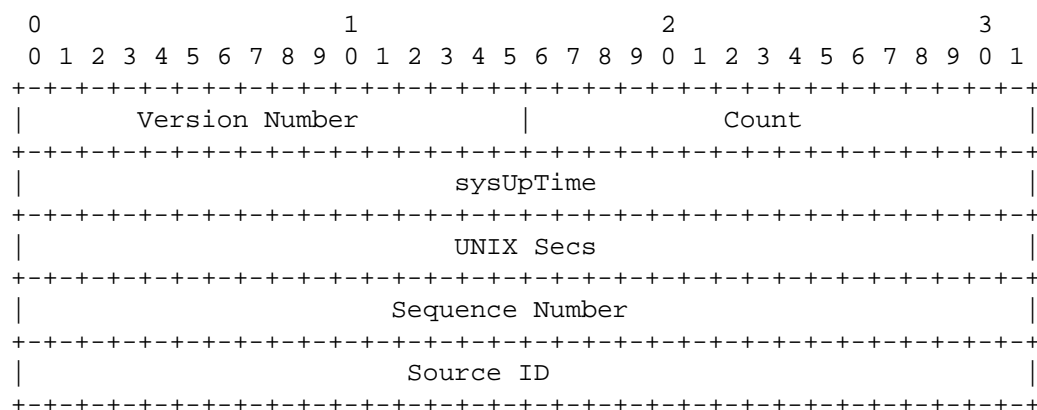
                  Figure 11: NetFlow V9 Packet Header Format

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Version Number        |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Export Time                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Sequence Number                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Observation Domain ID                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
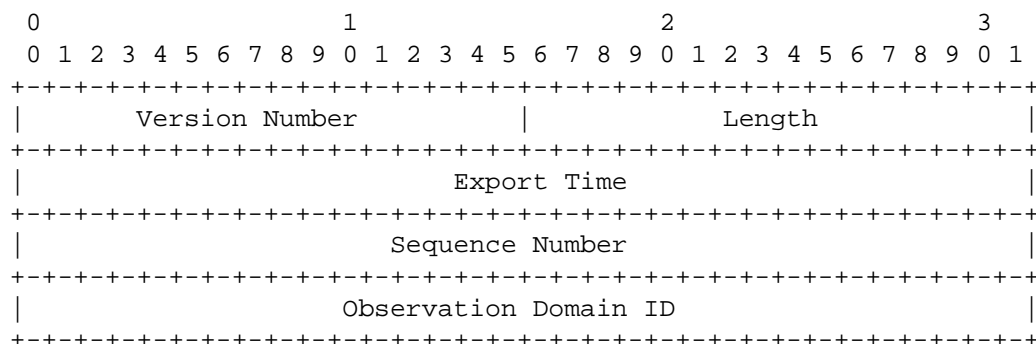
                 Figure 12: IPFIX Message Header Format

   Version Number: The IPFIX Version Number MUST be 10, while the
      NetFlow V9 Version Number MUST be 9.

   Length vs. Count: The Count field in the NetFlow V9 packet header
      counts records in the message (including Data and Template
      Records), while the Length field in the IPFIX Message Header
      counts octets in the message. Note that this implies that NetFlow
      V9 collectors must rely on datagram boundaries or some other
      external delimiter; otherwise, they must completely consume a
      message before finding its end.

   System Uptime: System uptime in milliseconds is exported in the
      NetFlow V9 packet header. This field is not present in the IPFIX
      Message Header, and must be exported using an IPFIX Option if
      required.

   Export Time: Aside from being called UNIX Secs in the NetFlow V9
      packet header specification, the export time in seconds since 1
      January 1970 at 0000 UTC appears in both NetFlow V9 and IPFIX
      message headers.

   Sequence Number: The NetFlow V9 Sequence Number counts packets,
      while the IPFIX Sequence Number counts records in Data Sets. Both
      are scoped to Observation Domain.

   Observation Domain ID: Similarly, the NetFlow V9 sourceID has
      become the IPFIX Observation Domain ID.

B.1.2. Set Header Format

   Set headers are identical between NetFlow V9 and IPFIX; that is, each
   Set (FlowSet in NetFlow V9 terminology) is prefixed by a 4-byte set
   header containing the Set ID and the length of the set in octets.

Note that the special Set IDs are different between IPFIX and NetFlow
V9. IPFIX Template Sets are identified by Set ID 2, while NetFlow V9
Template FlowSets are identified by Set ID 0. Similarly, IPFIX
Options Template Sets are identified by Set ID 3, while NetFlow V9
Options Template FlowSets are identified by Set ID 1.

Both protocols reserve Set IDs 0-255, and use Set IDs 256-65535 for
Data Sets (or FlowSets, in NetFlow V9 terminology).

B.1.3. Template Format

Template FlowSets in NetFlow V9 support a subset of functionality of
those in IPFIX. Specifically, NetFlow V9 does not have any support
for vendor-specific Information Elements as IPFIX does, so there is
no enterprise bit or facility for associating a private enterprise
number with an information element. NetFlow V9 also does not support
variable-length fields.

Options Template FlowSets in NetFlow V9 are similar to Options
Template Sets in IPFIX subject to the same caveats.

B.1.4. Information Model

The NetFlow V9 field type definitions are a compatible subset of, and
have evolved in concert with, the IPFIX Information Model. IPFIX
Information Element identifiers in the range 1-127 are defined by the
IPFIX Information Model [RFC5102] to be compatible with the
corresponding NetFlow V9 field types.

B.1.5. Template Management

NetFlow V9 has no concept of a Transport Session as in IPFIX, as
NetFlow V9 was designed with a connectionless transport in mind.
Template IDs are therefore scoped to an Exporting Process lifetime
(i.e., an Exporting Process instance between restarts). There is no
facility in NetFlow V9 as in IPFIX for Template withdrawal or
Template ID reuse. Template retransmission at the Exporter works as
in UDP-based IPFIX Exporting Processes.

B.1.6. Transport

In practice, though NetFlow V9 is designed to be transport-
independent, it is transported only over UDP. There is no facility
as in IPFIX for full connection-oriented transport without datagram
boundaries, due to the use of a record count field as opposed to a
message length field in the packet header. There is no support in
NetFlow V9 for transport layer security via TLS or DTLS.

B.2. A Method for Transforming NetFlow V9 Messages to IPFIX

   This appendix describes a method for transforming NetFlow V9 Packets
   into IPFIX Messages, which can be used to store NetFlow V9 data in
   IPFIX Files. A process transforming NetFlow V9 Packets into IPFIX
   Messages must handle the fact that NetFlow V9 Packets and IPFIX
   Messages are framed differently, that sequence numbering works
   differently, and that the NetFlow V9 field type definitions are only
   compatible with the IPFIX Information Model below Information Element
   identifier 128.

   For each incoming NetFlow V9 packet, the transformation process must:

   1. Verify that the Version field of the packet header is 9.

   2. Verify that the Sequence Number field of the packet header is
       valid.

   3. Scan the packet to:

       1. Verify that it contains no Templates with field types outside
           the range 1-127;

       2. Verify that it contains no FlowSets with Set IDs between 2
           and 255 inclusive;

       3. Verify that it contains the number of records in FlowSets,
           Template FlowSets, and Options Template FlowSets declared in
           the Count field of the packet header; and

       4. Count the number of records in Data FlowSets for calculating
           the IPFIX Sequence Number.

   4. Calculate a Sequence Number for each IPFIX Observation Domain by
       storing the last Sequence Number sent for each Observation Domain
       plus the count of records in Data FlowSets in the previous step
       to be sent as the Sequence Number for the IPFIX Message following
       this one within that Observation Domain.

   5. Generate a new IPFIX Message Header with:

       1. a Version field of 10;

       2. a Length field with the number of octets in the IPFIX
           Message, generally available by subtracting 4 from the length
           of the NetFlow V9 packet as returned from the transport layer
           (accounting for the difference in message header lengths);

      3. the Sequence Number calculated for this message by the
          Sequence Number calculation step; and

      4. Export Time and Observation Domain ID taken from the UNIX
          secs and Source ID fields of the NetFlow V9 packet header,
          respectively.

   6. Copy each FlowSet from the Netflow V9 packet to the IPFIX Message
       after the header. Replace Set ID 0 with Set ID 2 for Template
       Sets, and Set ID 1 with Set ID 3 for Options Template Sets.

   Note that this process loses system uptime information; if such
   information is required, the transformation process will have to
   export that information using IPFIX Options. This may require a more
   sophisticated transformation process structure.

B.3. NetFlow V9 Transformation Example

   The following two figures show a single NetFlow V9 packet with
   templates and the corresponding IPFIX Message, exporting a single
   flow record representing 60,303 octets sent from 192.0.2.2 to
   192.0.2.3. This would be the third packet exported in Observation
   Domain 33 from the NetFlow V9 exporter, containing records starting
   with the 12th record (packet and record sequence numbers count from
   0).

   The ** symbol in the IPFIX example shows those fields that required
   modification from the NetFlow V9 packet by the transformation
   process.

```
                            1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |            Version = 9            |            Count = 2            |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                 Uptime = 3750405 ms (1:02:30.405)               |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |   Export Time = 1171557627 epoch sec (2007-02-15 16:40:27)      |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                       Sequence Number = 2                       |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                    Observation Domain ID = 33                   |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |          Set ID = 0              |         Set Length = 20         |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |        Template ID = 256        |         Field Count = 3         |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     | IPV4_SRC_ADDR           =    8 |         Field Length = 4        |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     | IPV4_DST_ADDR           =   12 |         Field Length = 4        |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     | IN_BYTES                =    1 |         Field Length = 4        |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |         Set ID = 256            |         Set Length = 16         |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                          IPV4_SRC_ADDR                          |
     |                            192.0.2.2                            |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                          IPV4_DST_ADDR                          |
     |                            192.0.2.3                            |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                            IN_BYTES                             |
     |                              60303                              |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 13: Example NetFlow V9 Packet

```
                           1                   2                   3
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      | **       Version = 10      | **        Length = 52            |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |    Export Time = 1171557627 epoch sec (2007-02-15 16:40:27)   |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      | **                   Sequence Number = 11                     |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                   Observation Domain ID = 33                  |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      | **        Set ID = 2       |         Set Length = 20          |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |       Template ID = 256    |        Field Count  = 3          |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |0| sourceIPv4Address     =  8 |       Field Length = 4         |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |0| destinationIPv4Address = 12 |      Field Length = 4         |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |0| octetDeltaCount       =  1 |       Field Length = 4         |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |          Set ID = 256      |        Set Length = 16           |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                        sourceIPv4Address                      |
      |                           192.0.2.2                           |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                      destinationIPv4Address                   |
      |                           192.0.2.3                           |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                         octetDeltaCount                       |
      |                            60303                              |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 14: Corresponding Example IPFIX Message

Authors' Addresses

   Brian Trammell
   Hitachi Europe
   c/o ETH Zurich
   Gloriastrasse 35
   8092 Zurich
   Switzerland
   Phone: +41 44 632 70 13
   EMail: brian.trammell@hitachi-eu.com

   Elisa Boschi
   Hitachi Europe
   c/o ETH Zurich
   Gloriastrasse 35
   8092 Zurich
   Switzerland
   Phone: +41 44 632 70 57
   EMail: elisa.boschi@hitachi-eu.com

   Lutz Mark
   Fraunhofer IFAM
   Wiener Str. 12
   28359 Bremen
   Germany
   Phone: +49 421 2246206
   EMail: lutz.mark@ifam.fraunhofer.de

   Tanja Zseby
   Fraunhofer Institute for Open Communication Systems
   Kaiserin-Augusta-Allee 31
   10589 Berlin
   Germany
   Phone: +49 30 3463 7153
   EMail: tanja.zseby@fokus.fraunhofer.de

   Arno Wagner
   ETH Zurich
   Gloriastrasse 35
   8092 Zurich
   Switzerland
   Phone: +41 44 632 70 04
   EMail: arno@wagner.name