A Proposed Translation Data Model for Flow Format Interoperability

Brian Trammell
CERT Network Situational Awareness Group
Carnegie Mellon University
Pittsburgh, PA 15213, USA
bht@cert.org

Abstract

A significant technical barrier to the growth of the security-oriented network flow data analysis community is the mutual unintelligibility of raw flow and intermediate analysis data used by the proliferation of flow data analysis tools. This paper presents a proposed solution to this problem, a common event data model and a translator built around it to adapt each tool's native format to this common model.

1 Introduction

While non-technical barriers do exist to collaborative network flow data analysis across administrative domains, and these barriers are in many cases formidable, organizations finding both the desire and the political will to share data in the pursuit of a greater awareness of activities on the Internet at large are soon presented with another problem; their tools will not talk to each other.

The Internet Engineering Task Force is presently addressing the raw flow data standardization problem with IPFIX (Internet Protocol Flow Information Export) [1], a flow data format and collection architecture based upon Cisco's NetFlow version 9. This standardization effort is a start. However, it it not a complete solution to the interoperability problem. IPFIX is a wire protocol designed to efficiently generate and move flow data from an observation point (such as a router) to a collector, and as such does not address issues of short- or long-term data storage, the expression of query results containing flows or flow data summaries, or the handling of ancillary data used in flow data analysis that is not necessarily flow-oriented itself.

In addition, mandating that each existing flow collection and analysis toolchain use IPFIX natively is not really a satisfactory answer. While these toolchains will have to support the import of IPFIX flow data as the IPFIX standard is deployed in new observation points, each of these formats has evolved for a reason. If this were not the case, little specialization would have occurred beyond raw Netflow itself.

How, then, to improve the technical state of interoperability and cooperation within the network flow data analysis community? This paper presents a proposed solution to this problem. Section 2 outlines the requirements of such a solution, sections 3 and 4 propose a data model meeting these requrements, and section 5 builds a candidate design for a translator around this model.

2 Requirements Analysis

We propose the application of an event translator to this interoperability problem. The translator's design must meet the following requirements in order to be useful:

- Universality: the translator must be able to handle any type of event, event summary, or associated data, and be able to translate between any two semantically compatible formats.
- Filterability: Since the interoperation point between organizations often requires data obfuscation or sanitization, the translator must support filtering during translation.
- Ease of Translation: the translator's design must seek to minimize the development time required to add a new known format to the translator.
- Performance: though it is too much to expect a workflow with a translation step to perform as well as one using a toolchain's native format, the translator must process data quickly enough to ensure that translation does not inordinately slow down the workflow.
- Portability: the translator must be able to run on multiple POSIX (or POSIX-like) environments to

reflect the variety of environments used for flow data processing.

3 Event Data Model

The key realization leading to this proposal is that at its core, all collections of security-relevant network data, whether from flow collectors, network intrusion detection and prevention systems, host monitoring and intrusion detection systems, security information management products, etc., are made up of *events* . An event is simply an assertion that, according to some *event source* (e.g., a sensor, flow collector, etc.), something happened at some point in time, and possibly continued happening for some given duration. Therefore, every event record must have start and end timestamps and some identifier for the event source; these fields make up the *event core* data model.

Event records are made up of *key* and *value* fields. A key field is a field which defines some property of the event itself (e.g., packer header information), while a value field simply contains information about the event (e.g., counters). Each event is comprised of three or more key fields (including at least start, end, and source) mapping to zero or more value fields.

An event source is defined as a generator of a single *type* of event at a single network observation point, where type is defined by a list of key and value fields required of events of that type. This restriction is introduced to simplify processing of multiple types of data, e.g., flow events and NIDS alert events, collected at the same observation point; instead of associating a type with each event, the type is associated with the event's source.

3.1 Uniflows and Biflows

Raw flow data records contain an IP 5-tuple (source and destination IP address, source and destination port, and IP protocol), as well as packet and octet counters, and various routing information.

Flow data formats can be broadly split into two types: unidirectional (e.g., Cisco NetFlow V5), and bidirectional (e.g, Argus [1] and other pcap-based flow collection systems). We will call these *uniflows* and *biflows* for the sake of brevity. The requirement to handle both uniflows and biflows introduces some complexity into the semantics of source and destination.

For any uniflow, the meaning of source and destination are relatively straightforward; the source is the source IP address from the IP headers of the packets making up the flow, and the destination is the destination IP address from the headers. Biflows complicate the matter somewhat. The most straightforward way to assign source and destination for biflows is by the packet headers of the first packet observed, so ignoring instability at flow capture startup, the source address of a biflow corresponds to the connection initiator. However, we have seen at least one biflow format storing data differently than this; Q1Labs' QRadar [2] product presents flows by local and remote address, assuming some network perimeter under observation, and stores an additional "direction" arrow to note whether the connection is believed to have been initiated from inside or outside this perimeter. Note that this greatly confuses the definitions of "source" and "destination"; it may be better instead to refer two each address/port two-tuple as "endpoint A" and "endpoint B".

Free convertibility among these types requires that these semantics be stored for each event source, and that event sources storing biflows with an implicit perimiter must have a way of noting that perimeter.

Another difference between uniflows and biflows is that biflows have twice the counters than uniflows do; one counter of each type for packets initiated at each endpoint.

3.2 Summary Counters

It is also useful to exchange summary flow data. Retrospective traffic volume anomaly detection, for example, can be performed on flows aggregated by time bin and network. The core event data model natively supports time binning; an event with a start time at the beginning of the bin and an end time at the end of the bin represents a record for that bin.

For summarization purposes, the data model supports prefix lengths for source and destination address, as well as special "any" labels for other key fields.

TopN lists and other simple sequenced key/value counters bounded in time are another common type of summary data. The event data model supports these using multiple events, one for each place in the sequenced count list. Each of these events contains the key field and the count value, as well as a sequence number to provide order to the collection of events.

3.3 Event Classes

The event data model as described so far would seem to inherit from one another; that is, both flows and counts are types of events, and both uniflows and biflows are types of flows. Therefore, the data model can naturally be decomposed into classes in an object oriented design.

The classes comprising the event data model are illustrated in figure 1. The event data model is designed to be extensible, so new fields can be translated from one format to another, with the caveat that each class within the data model is immutable, and that extensions are

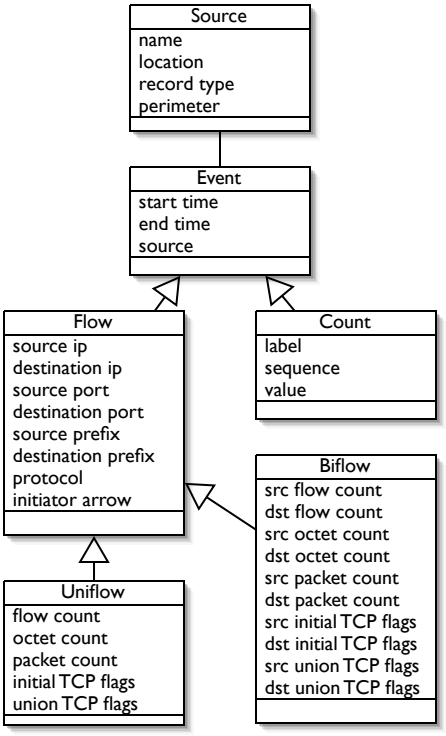

Figure 1: Event Model conceptual UML class diagram

achieved only by adding new classes which inherit from existing ones, not by adding fields to existing classes. This restriction is intended to reduce issues with data model object versioning at implementation time.

3.4 Other Event Types

Though the event core data model can handle any event placed in time and associated with an event source, this document is limited in scope to flow events and associated counters. Future revisions that deal in supporting correlations among event data types such as flow summaries and SIM events will extend the model accordingly.

3.5 Event Source Class

Event sources are handled by reference in the event data model. The one translation task presently supported by the event data model that requires information about the event source is translation between implicit-perimeter biflows and biflows without an implicit perimeter. The event source class therefore contains a perimeter field, which specifies the "local" network as a set of CIDR blocks or network address ranges.

4 Association Data Model

It is also useful to make assertions about the environment under monitoring that are not events, in order to further describe or aggregate events. An *association* is an assertion that, within a given time range, one set of key fields is known to map to another set of key and/or value fields. Associations have sources as do events; however, while association sources are only capable of generating a single type of association record, they are not associated with an observation point. An example of an association is a netblock record mapping a block address and a prefix with a block name, geographic location, and technical and administrative point-of-contact handles; the source of such an association might be a regional internet registry.

The inclusion of associations in the data model does not specifically support the translation of raw flow data from format to format, but it does support the translation of data into more aggregated or annotated formats, which may enrich the analysis products that can be derived from them.

Each association has three timestamps. The start and end timestamps define the time period during which the association is presumed to be valid; this can be used for applications such as historical DNS resolution or routing information. The third timestamp is the most recent update timestamp, which can be used to track the last time a given association was checked against its source database.

Future work will flesh out association subclasses; we hope to receive community feedback on the usefulness of the association mechanism and the types of associations presently used in aggregate analysis.

5 Candidate Translator Design

Given a data model through which to translate flows and other event data, a candidate design for a translator built around that model suggests itself. Each format will require both an input reader, to build objects in the common event data model from an input stream of a given input data format; and an output writer, to translate these common objects back into the desired output data format.

Each of these readers and writers could be written from scratch, handling their own disk I/O and other low level functions, but this is both relatively inflexible, and implies an inordinate duplication of effort. Instead, these readers and writers will be implemented in terms of I/O primitives that will handle low-level I/O. The flexibility gained by this approach could be applied to translate records from data sources other than regular files on disk, for example, from shared memory in a blackboard architecture, or directly off the wire in the case of IPFIX data

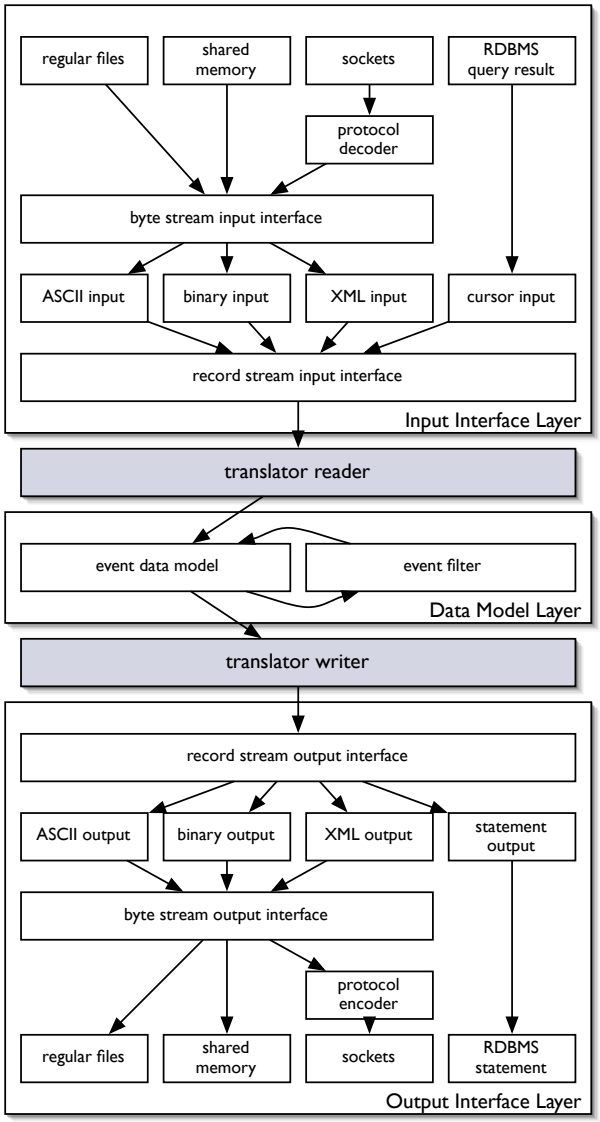

Figure 2: Candidate translator design data flow

collection.

In practice, there are limited number of ways which flow data can be represented in storage and transit, e.g. in fixed or variable length binary records, in delimited and separated ASCII records, as XML elements, as rows in a relational database. So the work of each format reader/writer author can be made less arduous still by providing an interface atop each of these fundamental formats.

Applying these principles, we are left with a data flow resembling Figure 2.

5.1 Event Model Implementation

We intend the data model to be implemented as a set of ANSI C structures, using structure containment to implement class inheritance. The choice of ANSI C was made for several reasons; most important are performance and portability.

The event data model is also designed to be implementable as a native storage format for the translator. Three implementations are planned: an XML Schema for text interchange of flow data, an IPFIX-compatible set of templated binary formats for binary interchange, and an RDBMS schema for long term storage and analysis in a relational database.

6 Conclusions and Future Directions

This paper has presented an event data model and a candidate translator design built around it, to facilitate the sharing of flow data among network security analysis communities. Core to the architecture of this translator is the realization that network security analysis tasks revolve around the manipulation of two classes of data, events and associations.

As of this writing, the ideas presented herein exist primarily on the whiteboard. The design pattern presented here has proven its usefulness in the cargogen tool released as part of AirCERT [3]; however, this tool is rather limited in its flexibility, shipping with only one input and output translator, and supporting only event data. Associations are supported after a fashion by the AirCERT AddrTree [4] RIR data source toolchain, although AddrTree does not support associations as a general superclass. These tools may be seen as ancestors of this present effort.

We plan to continue the design and implementation of this system during the summer and fall of 2005, focusing at first on flow translation, then on other types of events, finally implementing support for a variety of association subclasses.

References

[1] CLAISE, B. Ipfix protocol specification. Internet-Draft 15, Internet Engineering Task Force, May 2005. http://www.ietf.org/internet-drafts/draft-ietf-protocol-15.txt.

Notes

[1] http://www.qosient.com
[2] http://www.q1labs.com/products/prod_overview.html
[3] http://aircert.sourceforge.net
[4] http://aircert.sourceforge.net/addrtree