Intrusion Detection Working Group
draft-mansfield-curry-idmef-xmlsmi-01.txt
Expires: August 31, 2000

G. Mansfield/D. Curry
Cyber Solutions/IBM

March 1, 2000

Intrusion Detection Message Exchange Format Comparison of SMI and XML Implementations

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC 2026 [1].

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Distribution of this memo is unlimited.

This Internet-Draft expires August 31, 2000.

Abstract

The purpose of the Intrusion Detection Message Exchange Format (IDMEF) is to define data formats and exchange procedures for sharing information of interest to intrusion detection and response systems, and to the management systems which may need to interact with them. The goals and requirements of the IDMEF are described in [3].

Two implementations of the IDMEF data format have been proposed: one using the Structure of Management Information (SMI) to describe an SNMP MIB, and the other using a Document Type Definition (DTD) to describe XML documents. Both representations appear to have their good and bad traits, and deciding between them is difficult.

To arrive at an informed decision, the working group tasked the authors to identify and analyze the pros and cons of both approaches, and present the results in the form of an Internet-Draft.

TABLE OF CONTENTS

1. Conventions used in this document 4

2. Methods for communicating intrusion detection alert data 4
2.1 Tell-only 4
2.2 Tell-and-ask 5

3. Overview of proposed implementations 7
3.1 SMI 7
3.2 XML 8

4. Comparison criteria 10
4.1 Representation issues 10
4.1.1 Naming 10
4.1.1.1 SMI 11
4.1.1.2 XML 12
4.1.2 Data model 12
4.1.2.1 SMI 12
4.1.2.2 XML 13
4.1.3 Data format 14
4.1.3.1 SMI 14
4.1.3.2 XML 14
4.2 Operational issues 14
4.2.1 Bits on the wire 15
4.2.1.1 SMI 15
4.2.1.2 XML 16
4.2.2 Load on the CPU 17
4.2.2.1 SMI 17
4.2.2.2 XML 17
4.3 Implementation issues 18
4.3.1 Size of code 18
4.3.1.1 SMI 18
4.3.1.2 XML 19
4.3.2 Availability of code 19
4.3.2.1 SMI 20
4.3.2.2 XML 20
4.4 End point issues 20
4.4.1 Data display aspects 20
4.4.1.1 SMI 20
4.4.1.2 XML 20
4.4.2 Data transfer aspects 21
4.4.2.1 SMI 21

4.4.2.2 XML 21
4.5 Deployment issues 22
4.5.1 SMI 22
4.5.2 XML 22
4.6 Transport issues 22
4.6.1 TCP/UDP 22
4.6.1.1 SMI 22

4.6.1.2 XML 23
4.6.2 Intrusion Alert Protocol (IAP) 23
4.6.2.1 SMI 23
4.6.2.2 XML 23

5. Selected Implementation 23
5.1 Selection Rationale 23

6. Security Considerations 24

7. References 24

8. Authors' Addresses 25

1. Conventions used in this document

The key words "MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT," "RECOMMENDED," "MAY," and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [2].

2. Methods for communicating intrusion detection alert data

The requirements document defines the intrusion detection model that we are assuming: One or more sensors monitor some number of data sources for signs of intrusions, and report their observations to one or more analyzers. When an analyzer determines somehow that these observations represent a suspicious event, it sends an alert to one or more managers. A manager may respond to an alert by notifying an operator, investigating further by exchanging data with its peers, querying the source of the alert, or communicating the event to a higher level manager.

The format of the alert sent by the analyzer to the manager, and the method of communicating it, are what the IDMEF proposes to standardize.

In discussions within the working group, two different modes of operation have been suggested for communicating alert data between analyzers and managers. It should be noted that the IDMEF concerns itself only with the format of the alert, and not the design of the system that delivers them, or the protocols used to do so. However, it is important to have an idea of the communications mode(s) that will be used by intrusion detection systems when choosing an intrusion detection alert format, because some formats may support certain modes better than others.

2.1 Tell-only

This mode provides for a unidirectional communications flow, from an analyzer to one or more managers, as shown in Figure 1. (Managers may also pass the alert to other managers, in a hierarchical arrangement.)

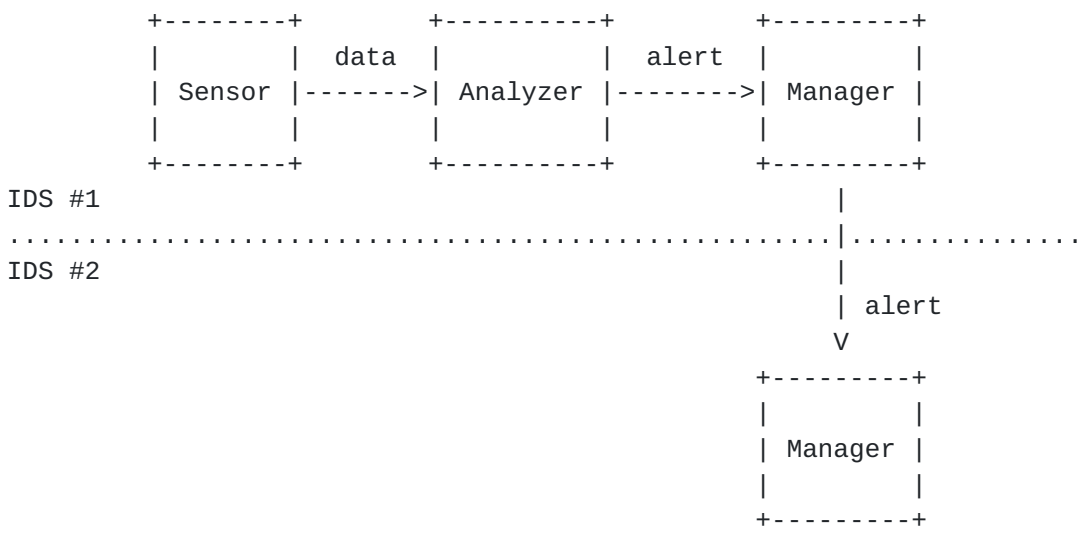

Figure 1. Tell-only mode.

When an analyzer detects an event (or some sequence of events) that must be communicated to a manager, it uses an alert message to do this. The analyzer places all the available information about the event(s) into the alert, and sends it to the manager. Once the alert has been sent to the manager, the analyzer generally "forgets" about both the alert and the events that led up to it.

The principal advantage to this mode is that analyzers can be kept both simple and small, allowing them to be deployed with less impact on performance, and using smaller and less expensive hardware. The principal disadvantage is that a manager is "stuck" with whatever the analyzer sends it -- this may be not enough information, or it may be too much. The latter case can be particularly problematic; the possibility exists for a poorly configured analyzer to inundate a manager with messages the manager has no use for but cannot ignore.

The IDMEF requirements document assumes a tell-only mode for the communication of IDMEF messages (Section 3.1, paragraph 4). Most intrusion detection products on the market today implement the tell-only model.

2.2 Tell-and-ask

This mode provides for bidirectional communication -- from an analyzer to one or more managers and also from the managers to the analyzers, for alert data exchange, as shown in Figure 2.

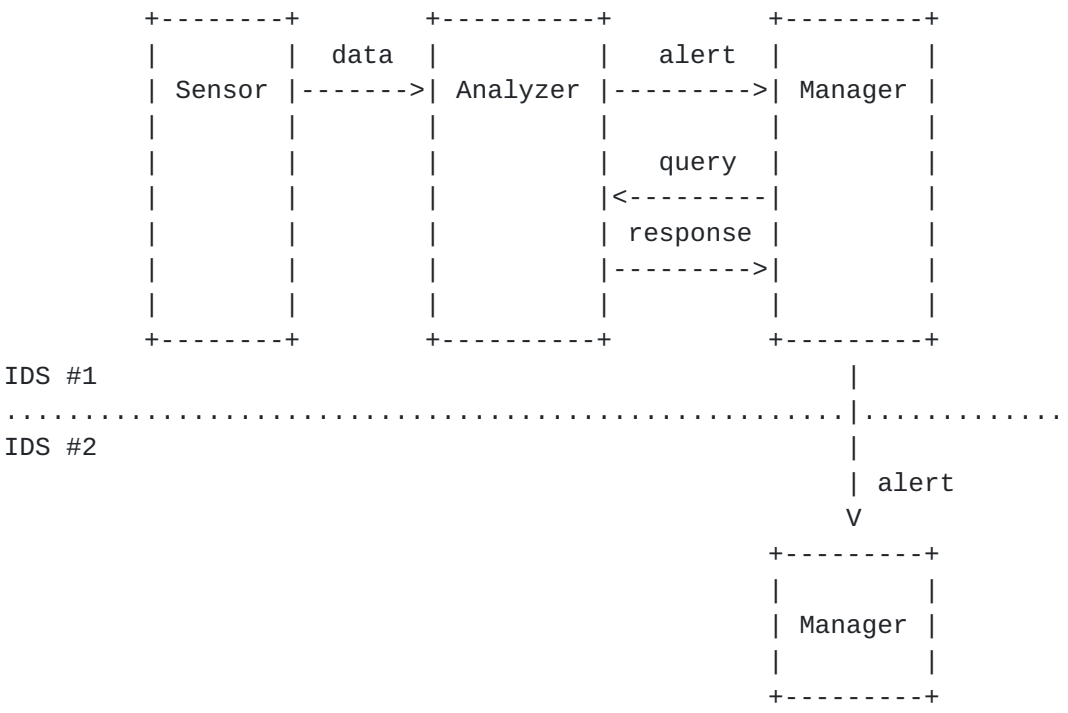

Figure 2. Tell-and-ask mode.

As with the tell-only mode, when an analyzer detects an event (or some sequence of events) that must be communicated to a manager, it uses an alert message to do this. However, instead of sending all the available event information along with the alert, the analyzer may choose to send only the "vital" data about the alert (type, time, priority, etc.). The decision about which information is considered "vital" and which is not will depend on the local circumstances and configuration.

If the manager receiving the alert is "interested" in the events that caused it to be generated, it can (either automatically or under the control of an operator) contact the analyzer and request additional information. The manager might ask for:

- more information on the alert pointers provided as URLs in the alerts from the agents (using appropriate protocols such as HTTP, FTP, etc.);

- more host-related information on the circumstances under which the intrusion/attack was detected -- this may involve fetching further information from the various MIBs of the entity that originated the notification; or

- more network-related information on the circumstances under which the intrusion/attack was detected -- this may involve fetching

further information from the various MIBs of the relevant network entities in the network.

The principal advantage of this mode of operation is its flexibility.

Alerts can be made smaller. This addresses the inundation problem described above. Moreover, since the query facility is available the analyzers may choose to retain some or all relevant information, e.g., analysis-logs, packet-dumps, etc. for some period of time. It may also provide any information it has and in which the manager is interested. The principal disadvantage of this mode is that analyzers will be relatively more complex compared to those operating in the tell-only mode. They must store historical data for some "reasonable" period of time, either in memory or on disk, and be able to retrieve that data when asked, all the while still performing their primary function, detection of intrusions.

The IDMEF requirements document indicates that the IDMEF data format may support the tell-and-ask communications mode (Section 3.1, paragraph 4).

A Remote Monitoring (RMON) device, a fairly popular network managent agent, can be configured to function as an IDS (with limited functionality) and is an example of an IDS that operates in the tell-and-ask mode.

3. Overview of proposed implementations

Two implementations of the IDMEF data format have been proposed: one using the Structure of Management Information (SMI) to describe an SNMP MIB, and the other using a Document Type Definition (DTD) to describe XML documents. The implementations are presented briefly in this section; for more detail on either implementation consult the relevant Internet-Drafts [4, 5].

3.1 SMI

Network management involves monitoring and manipulating information about the elements to be managed. These bits and pieces of information are abstracted as Managed Objects (MO). A detailed introduction to the current SNMP Management Framework can be found in [7].

Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. Objects in the MIB are defined using the mechanisms defined in the SMI.

Each resource to be managed is represented as an object in the MIB. The MIB is a database structure in the form of a tree in which each node has a sub-identifier. The identifier of the node is a concatenation of all the sub-identifiers starting at the root down to the node itself.

The Structure of Management Information (SMI) defines the general framework within which a MIB may be defined. It identifies the

datatypes that can be used in the MIB and specifies how resources within the MIB can be represented and named. A restricted subset of ASN.1 is used for this purpose. The types defined in the SMI are

- Simple types: INTEGER, OCTET STRING, OBJECT IDENTIFIER
- Structure types: SEQUENCE and SEQUENCE OF

Thus, in effect, only scalars and two dimensional arrays of scalars can be represented (though only scalars can be manipulated).

Every object has a unique name, its Object Identifier (OID) - the identifier of the corresponding node in the MIB tree. Further, there is an indexing mechanism whereby the information related to a set of similar objects (for example, interfaces or routes) may be represented and accessed as rows in a table.

Thus the operational status of an interface will have the name

.1.3.6.1.2.2.8

And if the index for the interface that we are interested in is 2 then we would "get" the value of the object named

.1.3.6.1.2.2.8.2

The MO definition of the operational status object will also tell us (and the Network Management Station) that the object is of type INTEGER.

The proposed SMI-MIB implementation of IDMEF works well in the tell-and-ask mode. The effective naming scheme and potentially fine granularity of the named objects makes it an ideal candidate for sending a small alert message and then respond to queries from the manager. Nevertheless, the SMI-MIB also works in the tell-only mode. But it should be noted that large messages are generally frowned upon by managers/operators as they consume network and system resources, and, when being transported over UDP using an SNMP-like application protocol, large messages will have to be IP-fragmented or broken into multiple messages. Over TCP, this problem can be avoided.

3.2 XML

The Extensible Markup Language (XML) is a text markup syntax defined by the World Wide Web Consortium (W3C). It is gaining widespread attention as a language for representing and exchanging documents and data on the Internet. XML is currently being used in a variety of projects, including the Text Encoding Initiative, Microsoft Channel Definition Format, Wireless Application Protocol (WAP) Wireless

Markup Language, Chemical Markup Language, Weather Observation Markup Format, Open Financial Exchange, OpenMLS (real estate), Mathematical Markup Language, Electronic Data Interchange (several projects), News

Industry Text Format, and a variety of others. For a comprehensive list, of XML applications, see

http://www.oasis-open.org/cover/xml.html#xml-osd

XML is a metalanguage -- a language for describing other languages -- that enables an application (such as IDMEF) to define its own markup. XML allows the definition of a customized markup language specific to an application. This differs from HTML, for example, in which a fixed set of markup tags with preset meanings must be "adapted" to uses for which they were not intended. Both XML and HTML use tags (identifiers delimited by '<' and '>') and attributes (of the form "name='value'"). But where "<p>" always means "paragraph" in HTML, it may mean "paragraph," "person," "price," or have no meaning at all in an XML application.

Each XML application defines the tags and attributes it needs in an XML Document Type Definition (DTD). Tags are defined to identify the semantic elements of the marked-up data (e.g., paragraphs, tables, figures, section headings, footnotes, chapters, titles, etc.) The DTD also specifies how the semantic elements of the data relate to each other (e.g., a chapter may only have one title, sections may only occur inside chapters, second-level headings must follow a first-level heading, and so on).

The XML IDMEF DTD defines the tags and attributes needed to identify the various elements of an intrusion detection alert, as set forth in the data model defined by Debar, Huang, and Donahoo [6]. A complete "document" (alert) might look like:

```
<IDMEF-Message version="0.2">
  <Alert id="1234567890" severity="75" confidence="100"
         method="knowledge">
    <Time offset="-5">
      <date>1999/10/21</date>
      <time>08:12:32</time>
    </Time>
    <Sender id="0" refid="2345678901" seq="1"></Sender>
    <Name type="bugtraqid">33</Name>
    <signature>loadmodule forking shell</signature>
    <Source id="5678901234">
      <User id="7890123456" type="osuser">
        <uid>13243</uid>
        <name>joe</name>
      </User>
      <Process id="6789012345">
        <name>loadmodule</name>
        <path>/usr/openwin/bin</path>
```

```
      </Process>
    </Source>
    <Target id="3456789012">
      <Host id="4567890123">
```

```
          <name>machine.domain.com</name>
          <Address type="ipv4">123.45.67.89</Address>
        </Host>
      </Target>
    </Alert>
  </IDMEF-Message>
```

The proposed XML implementation of the IDMEF allows the analyzers and managers to use either the tell-only or tell-and-ask communications modes. However, tell-and-ask functionality is limited to specific queries and responses, and does not implement a general query interface.

4. Comparison criteria

The authors have identified a variety of criteria against which the two implementations should be evaluated. These are presented below together with descriptions of how each criterion is met by the individual implementations.

4.1 Representation issues

Representation issues are those factors of the implementations that involve how intrusion detection alerts are represented: naming of alerts, fields in alerts, and alert-related information.

4.1.1 Naming

Naming is important to enable querying for object values. Examples include

- what is the complete list of hosts that were the target of a scan?

- how many TCP-SYNs were sent to host foo.bar.com from network aaa.bbb.ccc.ddd during the attack?

There are three characteristics of naming that are important to us: flexibility, granularity, and ease of use:

- How flexible is the naming scheme used by the implementation? For example, can objects in lists and tables be named even when the length/size of the list or table is not known?

- How granular is the naming scheme used by each implementation? Can individual objects in lists and tables be named? Can entire lists and tables be named?

- How easy is it to use the naming scheme in the above cases?

4.1.1.1 SMI

In the rest of this document, we will cite the example of SNMP as the application protocol that is used to access/exchange SMI-MIB objects. The rationale behind the choice of this example is that SNMP has been around for some time, is reasonably well understood among a larger community, is widely deployed and there is a lot of operational experience. Moreover the application protocol to handle these SMI-MIB objects, if it is not SNMP itself, is likely to be SNMP-like. Nevertheless, it should be noted that as of now SNMP is NOT the chosen application protocol and the authors have no intention of implying that it is.

Management involves getting (reading) and setting (writing) the values of (elementary) objects thus SMI supports and requires all objects of interest to be named. If an object is not explicitly or implicitly named (defined) it cannot be accessed. (Note: defining an object does not mean that it will be accessible.)

The naming offered by SMI is flexible to the extent that all objects of interest can (have to be) named. For data structures only two-dimensional tables are allowed. (Nesting is not allowed.) Tables are indexed and indices may be composed of one or more managed objects. The number of rows in a table do not need to be known before hand. The indexing mechanism takes care of that.

A management application generally finds out the indexes of the rows (which represent e.g. interfaces/routes) before hand using the powerful "get-next" protocol operation (the responder returns the lexicographically next element in the MIB to the originator) so that when it wants to monitor a particular resource it knows the name that needs to be used.

There is no limit to the granularity. MIBs can be defined to any level of granularity. For example an MO can well be defined for the nth bit of the mth packet of a certain protocol that passed through the interface of index i.

Individual objects can be accessed with ease in a table.

Entire lists and tables cannot be accessed, essentially the access is for each elementary (leaf) MO. However there are efficient mechanisms (SNMPget bulk) for fetching the MOs in a table or a subtree.

The naming scheme requires that both managers and agents know the names. MIBs are published for that purpose. There are mechanisms to know all the object instances that may be resident in the MIB of a

device, i.e. for discovering names/objects in a MIB (SNMPget next). This allows the manager to know all the manageable resources that are present at the other end.

4.1.1.2 XML

The XML IDMEF DTD specifies that each semantic element of an alert will be individually tagged or contained in an attribute. As shown in the example in the previous section, tags within an alert are organized in a more or less hierarchical structure. There are no limits to the depth or breadth of the hierarchy (other than those imposed by the IDMEF data model itself).

The Document Object Model (DOM), defined by the W3C, is a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of XML documents. Through the DOM, an application that processes XML IDMEF messages can treat an XML document (IDMEF alert) as a "database," and extract individual elements from it based on tag or attribute name.

The DOM allows on-the-fly manipulation of individual XML IDMEF messages by an application. The DOM does not, however, replace relational or object-oriented database management systems. An application that has a need to process historical alert data (e.g., for correlation or analysis purposes) would presumably use an RDBMS or OODBMS for this purpose. The XML IDMEF DTD's requirement that each semantic element of an alert be tagged or contained in an attribute, and the forthcoming XML Schema standard will make the conversion between the XML IDMEF message format and the database record format simple.

4.1.2 Data model

Debar, Huang, and Donahoo [6] have proposed that intrusion detection events in the IDMEF be represented by a class hierarchy. The working group has adopted their model as the one that should be followed by any data format implementation.

In this section, we examine how well the implementations match the data model, and any significant differences between the implementation and the model.

4.1.2.1 SMI

The Alert-MIB fits the data model neatly. Moreover, the messages themselves are indexed by a unique message-id. The manager uses this message-id to obtain further information about the event that caused the message e.g., the destinations that were targeted by the

attack and/or the packet-trace that contained the signature of the attack. [Note - there are deviations from the details given in data model document particularly related to data structures and data

representations. These details are under discussion in the WG.]

4.1.2.2 XML

The XML DTD is based on the IDMEF data model, but does not follow it exactly. The data model was completely rewritten after the IDWG meeting in November, 1999. It is therefore a first pass, and its authors have not had the opportunity to address the first round of comments. Accordingly, the XML DTD diverges from the data model in places where its author disagrees with the data model's approach. The DTD also extends the data model in several areas.

The significant differences between the IDMEF data model and the XML DTD are:

- The DTD does not support the "priority" and "impact" attributes of an alert; the author believes they overlap with the "severity" attribute and are therefore unnecessary. The DTD also does not support the "success" attribute, as the author sees no way for an analyzer to generate a meaningful value for it.

- The DTD does not support the "confidence" attribute of sources and targets; the author believes it is too complex to spread this out into multiple confidence ratings. It also does not support the "spoofed" attribute, since there is no way to accurately determine this (except in trivial cases).

- In addition to the alert generation time (required by the data model), the DTD provides the event detection time (needed for correlation) and the sender current time (needed to perform clock synchronization).

- The DTD adds a "Portlist" type to the "Service" description in the data model, to support providing comma-separated lists of ports and hyphenated ranges.

- The DTD dispenses with the "Analyzer" part of the data model that tries to associate analyzers and data sources, and replaces it with a more general "Sender." Senders are identified by "Host" and/or "Process" data, and they may be daisy-chained in the alert to show the entire forwarding path taken by the alert.

- The DTD merges the "AdditionalData" portion of the "ExtendedAlert" in to the "Alert" itself, and dispenses with "ExtendedAlert." The author believes this is more generally useful, allowing extra data to be provided for any kind of alert, instead of having two similar but separate alert hierarchies.

- The DTD provides a "Heartbeat" type of message for analyzers (and other components) to use to indicate their state of well-being to a manager.

- The DTD provides "Query" and "Response" messages for managers and analyzers to use to exchange data about sources, targets, hosts, users, processes, services, and networks. The "Query" allows a manager to ask an analyzer to fill in the data about something it gave only a reference id for, and the "Response" allows the analyzer to provide an answer to the query.

Aside from the above, the XML DTD is very close to the data model. It can be easily modified to bring it into closer "compliance" if that is the will of the group; none of these changes has any major impact on the overall design of the format.

4.1.3 Data format

What data representation format or formats are used by the implementation? For example, is everything "ASCII text" or "binary"? In addition to impacting ease-of-use, data format may affect the overall performance of an implementation.

4.1.3.1 SMI

The syntax of the managed objects are defined in the MIB - the data types used are INTEGER, OCTETSTRING, NULL and OBJECTIDENTIFIER. The encoding of the data for communication purposes is according to the Basic Encoding Rules (BER) associated with ASN.1.

4.1.3.2 XML

The XML standard specifies that all XML documents (and therefore all XML IDMEF messages) be encoded in either the UTF-8 or UTF-16 encodings of ISO 10646 (Unicode).

All data in an XML IDMEF message will be encoded as text; there will be no "binary" content. Numbers will be encoded as their formatted output equivalents (e.g., the number 123.45 would be represented by the six characters, '1,' '2,' '3,' '.,' '4,' and '5').

XML is capable of representing non-printable "binary" data, although the representation is not very efficient. Any arbitrary value can be encoded as decimal one-byte quantites (e.g., "<") or hexadecimal two-byte quantities (e.g., "<").

4.2 Operational issues

Operational issues are those factors of system and network operation that will be impacted by use of the implementation: network bandwidth consumed, memory and processor resources used, etc.

4.2.1 Bits on the wire

Intrusion detection systems are capable of sending many, many alerts in a very short period of time. Each of these alerts consumes some bandwidth on the network; if the number of alerts is too high, the network could become swamped, impacting not only the delivery of alerts, but all other applications using the network as well.

4.2.1.1 SMI

First lets look at the number of bytes that goes on the wire when we do a get for say the number of bytes that came in (ifInOctets) through an interface (Index = 2). The "get" PDU containing the variable name iso.org.dod.internet.mgmt.mib-2.interfaces.ifTable. ifEntry.ifOperatstus.1 and a null binding will contain the

```
UDP protocol header            [8 bytes]
Application protocol header    [31 bytes (6 byte community string)]
Payload: Variable Bindings     [22 bytes]
                          ---------------
                          TOTAL 61 bytes (+ IP Header).
```

Its response would be the get-response PDU containing the variable name and an integer (64bit counter) binding would contain

```
UDP protocol header            [8 bytes]
Application protocol header    [31 bytes (6 byte community string)]
Payload: Variable Bindings     [26 bytes]
                          ---------------
                          TOTAL 65 bytes (+ IP Header).
```

In case we want to do this with TCP then we will have to add the 12 bytes of excess length the TCP header has.

The point to note is that the average length of an MO name or an MOID is 10 bytes. When it is BER encoded and sent down the wire it is roughly 15 bytes.

For a complete alert containing the following information:

```
Alert.version                          = 1
Alert.priority                         = 1
Alert.confidence                       = 100
Alert.severity                         = 100
Alert.name                             = bugtraqid.33
Alert.signature                        = loadmodule forking shell
```

```
   Alert.method                          = 1
   Alert.time.date                       = 1999/10/21
   Alert.time.time                       = 08:12:32
```

```
Alert.analyzer.ident                    = 12345678
Alert.target[0].host.name               = machine.domain.com
Alert.target[0].host.address.type       = 11
Alert.target[0].host.address.value      = 123.234.345.456
Alert.source[0].process.name            = /usr/openwin/bin/loadmodule
Alert.source[0].user.name               = joe
Alert.source[0].user.uid                = 13243
```

The payload will be roughly:

```
SNMP Protocol Header (v1)                    =     31 bytes
+ [ 15(MOIDs)     * 15 (bytes/MOID)    ]     =    225 bytes
+ [ 10 (Integers) *  6 (bytes/Integer) ]     =     60 bytes
+ [  5 strings Length ={12,24,18,27}   ]     =     89 bytes
     2 bytes overhead per string
-------------------------------------------------------------
    Total                                    =    405 bytes
```

The SNMP application level protocol does not offer any means of compressibility. However if the transport offers compressibility this will certainly reduce the number of bits on the wire. But there is probably no straightforward way of saying of whether the SNMP payload is "more compressible" than the HTTP payload

4.2.1.2 XML

The XML IDMEF encoding of a complete alert containing the following information:

```
Alert.version                           = 1
Alert.priority                          = 1
Alert.confidence                        = 100
Alert.severity                          = 100
Alert.name                              = bugtraqid.33
Alert.signature                         = loadmodule forking shell
Alert.method                            = 1
Alert.time.date                         = 1999/10/21
Alert.time.time                         = 08:12:32
Alert.analyzer.ident                    = 12345678
Alert.target[0].host.name               = machine.domain.com
Alert.target[0].host.address.type       = 11
Alert.target[0].host.address.value      = 123.234.345.456
Alert.source[0].process.name            = /usr/openwin/bin/loadmodule
Alert.source[0].user.name               = joe
Alert.source[0].user.uid                = 13243
```

is shown in Section 2.2. When formatted as it would be sent over an XML IDMEF message channel (no newlines or indentation), the encoding

of this alert required 657 bytes.

XML, because of the open-tag/close-tag syntax, has a relatively high

overhead percentage. For the example above, XML tagging makes up about 66% of the total message.

However, XML is also readily compressed. Using Lempel-Ziv coding, the example above compresses to 407 bytes, a savings of 39%. By sending multiple alerts in the same message, compression results can be improved still further; savings of 80-90% are easily achieved.

4.2.2 Load on the CPU

Encoding data on the analyzer for transmission to the manager will put additional load on the analyzer's processor. Likewise, decoding and parsing the data on the manager will put additional load on the manager's processor. Depending on the processing resources needed, this additional load may impact the ability of the analyzer/manager to perform its other tasks.

4.2.2.1 SMI

There is an associated load with encoding data in the SNMP PDUs at the analyzer and then in decoding SNMP PDUs at the manager. The encoding is done in BER. However, the load is probably unavoidable if the data transferred is to be used for computational purposes.

For example, the integer 389360048 is represented as

```
41 04 17 35 29 B0
```

Where the first byte (41) represents the datatype, the second byte (04) represents the length in bytes of the following contents, the remaining 4 bytes (17 35 29 B0) represent the contents the hexadecimal representation of 389360048.

Incidentally if a plain ASCII text representation was used, the string would just be represented by

```
63 68 69 63 66 60 60 64 68
```

The time taken to parse an "average" SMI IDMEF message using a publicly available SNMP software package on a 300MHz CPU machine is roughly 60 microseconds.

4.2.2.2 XML

The load placed on an analyzer to generate XML IDMEF messages is minimal. The message itself is simply character string data, usually

generated with a formatted output function such as printf() from C/C++. Only basic control structures are needed to create the format of the message (e.g., "if value known then print else don't").

The load placed on a manager to process XML IDMEF messages is somewhat higher, since the manager must parse (and optionally validate) the XML document that represents an IDMEF message.

The following times to parse an "average" XML IDMEF message were measured on a 360 MHz UltraSPARC II with 128MB of memory:

Package	Language	Validation	Time
IBM XML4J 3.0 SAXCount	Java	On	47-52 ms
IBM XML4J 3.0 SAXCount	Java	Off	75-80 ms
IBM XML4C 2.3.1 SAXCount	C++	On	30-33 ms
IBM XML4C 2.3.1 SAXCount	C++	Off	27-29 ms

Timings for other processors, other parser implementations, and other IDMEF messages will of course vary.

4.3 Implementation issues

Implementation issues are those factors of the implementations that will impact vendors and authors of intrusion detection systems - i.e., how easy will it be to add support for IDMEF to existing and new intrusion detection systems?

4.3.1 Size of code

Adding support for IDMEF will require adding code to both the analyzer and the manager.

4.3.1.1 SMI

The parser of SMI-MIB objects is essentially an ASN.1 parser. With reference to a publicly available implementation the size of the parser written in C is Source code = 55Kbytes, object code = 75 Kbytes.

Considering the size of an analyzer, the minimum functionality that analyzers should have are

- send alerts
- receive (subsequent) queries
- reply to the queries

SNMP-agents can be very slim. They are meant to be slim and lightweight.

A simple agent program with no MIB support would be 1.4MB.

With MIBII support (i.e. with instrumentation for the MIBII objects - which means the manager can it questions about the number of TCP connections that are open and expect a reply) the agent size will be 2.5MB.

An SNMP agent that just receives alerts is 1.0MB.

The code required to send queries, receive responses, process/parse the responses and display them on the console in the (comparatively ugly but called "pretty print") form like interfaces.ifTable.ifEntry. ifInOctets.1 = 389360048 is roughly 1.0MB.

4.3.1.2 XML

Code size on the analyzer to generate XML IDMEF messages is so small as to be insignificant. 1-2 KB of string storage space (for the tag and attribute names), and a few hundred bytes of control structures.

Code size on the manager to parse the XML IDMEF message is also not large. The binary object or script file sizes for several freely available XML parsers is shown below:

Package	Language	Validating	Program Size
Expat 1.1	C	No	128 KB
IBM XML4J 3.0 (includes multiple parsers and document object model support)	Java	Yes	865 KB
IBM XML4C 2.3.1	C++	Yes	20 KB
TclXML 1.2	Tcl	No	58 KB
xmlproc 0.61	Python	Yes	156 KB
XP 0.5	Java	No	166 KB

The manager will also need space for the Java, Tcl, or Python runtime environments, if those are used, plus memory for the parser's data structures.

4.3.2 Availability of code

Availability of code addresses how easy it is for a vendor or author

of intrusion detection systems to obtain code to implement the IDMEF.

4.3.2.1 SMI

The code is widely available and freely over a large range of platforms.

The Perl version offers a complete and compact package which is of relatively small size (52K in tar gzipped form) but of course it will require the full Perl interpreter for its execution. Nevertheless the Perl package has made the package available over a very large range of platforms.

4.3.2.2 XML

Software tools for processing XML documents are widely available, in both commercial and open source forms. A variety of tools and APIs for parsing and/or validating XML are available in Java, C, C++, Tcl, Perl, Python, and GNU Emacs Lisp. For a comprehensive list of both commercial and open source XML tools, see

http://www.oasis-open.org/cover/publicSW.html#xmlTools

4.4 End point issues

End point issues are those factors of the implementations that impact how alert data will be handled once it reaches the manager.

4.4.1 Data display aspects

Data display aspects are those features of the data format that impact how data can be displayed to users, on graphical displays as well as in printed documents.

4.4.1.1 SMI

The MIB representation does not aid or hinder display at the end point.

There are a couple of MIB browsers (Perl, TCL. JAVA, ... based), but none (that I know of) directly interface with a web browser. The SMI->HTML translation, though not an impossibility, has not been attempted (not to my knowledge).

4.4.1.2 XML

One of the principal advantages of semantic tagging, such as that

used by XML, is that the same document can be used for a variety of applications without having to translate it to another format.

The Extensible Stylesheet Language (XSL), defined by the W3C, is a language for expressing stylesheets. Given a class of structured documents or data files in XML, designers use an XSL stylesheet to express their intentions about how that structured content should be presented; that is, how the source content should be styled, laid out and paginated onto some presentation medium such as a window in a Web browser or a set of physical pages in a book, report, pamphlet, or memo.

To display an XML IDMEF message on a graphical display, all that is needed is a viewing program (such as a web browser) that supports XML, and a style sheet that tells the program how to display the content of the various tags. The Microsoft Internet Explorer and Mozilla (not Netscape) web browsers have support for XML and XSL. There are also several free and commercial XML browsing tools available.

To display an XML IDMEF message on the printed page, all that is needed is a formatting program that supports XML, and a style sheet (different from the browser style sheet) that tells the program how to print the content of the various tags.

4.4.2 Data transfer aspects

Data transfer aspects are those features of the data format that impact how efficiently the implementation can transfer data.

4.4.2.1 SMI

SMI-MIBs are not good for representing bulk data (and SNMP is not good for transferring bulk data).

4.4.2.2 XML

The amount of data to be transferred is not a problem for XML, since the DTD just defines tags that identify that markup -- everything is treated as a simple stream of bytes. In situations where most of the data elements are small, however, XML may impose a lot of overhead, resulting in messages that require more bytes to represent the data tags than to represent the data itself. This overhead can be partially offset by XML's easy compressibility, but only if compression is available.

XML is primarily intended to represent "printable" data (UTF-8 or UTF-16). Although it is capable of representing arbitrary "binary"

data, its method for doing so is both cumbersome and inefficient, and should be avoided if at all possible.

4.5 Deployment issues

Deployment issues address how easy it will be to actually "get IDMEF out there" once it has been standardized (and adopted by vendors). These issues affect existing deployed systems (which may have to be upgraded or replaced), and existing products (which may have to be modified).

4.5.1 SMI

Most network devices have an SNMP agent in them, and thus the code to generate and handle SMI compliant (SNMP) messages is already there.

4.5.2 XML

To the authors' knowledge, XML is not currently supported by any existing intrusion detection products, commercial or otherwise.

However, some existing products already make use of the Java runtime environment to implement their management console functionality; this may make the integration of a Java-based XML parser somewhat easier than starting from scratch.

4.6 Transport issues

Transport issues are those factors of the implementations that impact how IDMEF messages can be transmitted via the network.

4.6.1 TCP/UDP

Can TCP-based protocols, UDP-based protocols, or both be used?

4.6.1.1 SMI

SNMP goes over UDP. This is looked upon as a major plus as when the network is in trouble the UDP packets have a better chance of getting through with least additional damage. Probably ID falls in the same category as far as network troubles are concerned.

The negative aspects of UDP is that it is unreliable. However by using Confirmed alerts (Informs) in which the receiver acknowledges the receipt can be (is) used to get around this problem.

SNMP may also be sent over TCP Experimental versions are out.

4.6.1.2 XML

XML IDMEF messages, since they are simply a stream of bytes, can be sent over TCP without problems. Indeed, a virtual circuit is the preferred transport method.

XML IDMEF messages can, in general, be sent over UDP too. However, it is possible that messages containing long lists of hosts or services could exceed the UDP maximum packet size, making it a less desirable transport mechanism.

4.6.2 Intrusion Alert Protocol (IAP)

The Intrusion Alert Protocol (IAP) has been selected by the working group as the protocol to be used for sending and receiving IDMEF alerts. IAP is an HTTP-like protocol over TCP that uses the Transport Layer Security protocol (a superset of the Secure Sockets Layer, SSL) for security and authentication.

4.6.2.1 SMI

SNMP PDUs will have no problem being transported over IAP.

4.6.2.2 XML

As XML is simply a stream of bytes, it can be transported over the IAP without problems.

5. Selected Implementation

On Februrary 1-2, 2000, an interim meeting of the Intrusion Detection Working Group was held at Harvey Mudd College. At this meeting, a tentative decision was made to use the XML IDMEF implementation. This recommendation will be put forth to the working group (via the mailing list), with strong comments requested. A final vote on the matter is tentatively scheduled for the March, 2000 IETF meeting in Adelaide, Australia.

5.1 Selection Rationale

The following points (in no particular order) make up the rationale for selecting the XML implementation over the SMI implementation:

- The IDMEF must support both network-based and host-based intrusion detection systems. While both XML and SMI make sense in network-

based systems, XML is much more "natural" in host-based systems. This is especially true when considering the representation of

text-based log formats such as UNIX "syslog."

- XML is more easily extended to other, related activities such as the command and control of intrusion detection systems. While such activities are outside the scope of the IDWG, selection of a data format that is compatible with them is viewed as beneficial.

- The SMI implementation of the tell-only mode is awkward (i.e., when all data must be communicated via trap/inform). The idea of trap-directed polling (tell-and-ask) for intrusion detection alerts is not attractive to the group.

- Some of the vendor representatives to the group indicated that they could/would not support the idea of querying their sensors for more data (i.e., tell-and-ask mode). While this idea is interesting from a research and correlation perspective, the vendors are more concerned with fast, lightweight sensors.

- XML is attractive to these same vendors, since it requires only a basic "print" function to produce XML-formatted IDMEF messages on the sensor.

- The Intrusion Alert Protocol (IAP) [8], adopted by the group, is an HTTP-like protocol. XML, because it is some sense "designed" for protocols of this type, is a "natural" for use with IAP.

6. Security Considerations

This Internet-Draft compares two data formats that have been proposed for the exchange of security-related data between security product implementations. There are no security considerations directly applicable to the format of this data. There may, however, be security considerations associated with the transport protocol chosen to move this data between communicating entities.

7. References

[1] Bradner, S., "The Internet Standards Process -- Revision 3," BCP 9, RFC 2026, October, 1996.

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

[3] Wood, M., "Intrusion Detection Message Exchange Requirements," draft-ietf-idwg-requirements-02.txt, October 21, 1999, work in progress.

[4] Mansfield, G. and D. Gupta, "Intrusion Detection Message MIB," draft-glenn-id-notification-mib-02.txt, January 29, 2000, work in progress.

[5] Curry, D, "Intrusion Detection Message Exchange Format Extensible Markup Language (XML) Implementation," draft-curry-idmef-xml-02.txt, March, 2000, work in progress.

[6] Debar, H., M. Huang, and D. Donahoo, "Intrusion Detection Exchange Format Data Model," draft-ietf-idwg-data-model-01.txt, January 21, 2000, work in progress.

[7] Case, J., R. Mundy, D. Partain, and B. Stewart, "Introduction to Version 3 of the Internet-standard Network Management Framework," RFC 2570, April, 1999.

[8] Gupta, D., "IAP: Intrusion Alert Protocol," draft-ietf-idwg-iap-01.txt, January 27, 2000, work in progress.

8. Authors' Addresses

Glenn Mansfield
Cyber Solutions, Inc.
6-6-3 Minami Yoshinari
Aoba-ku, Sendai 989-3204
Japan
Phone: +81 22-303-4012
Email: glenn@cysols.com

David A. Curry
Internet Security Systems
148 Madison Avenue, 11th Floor
New York, NY 10016
USA
Phone: +1 212 592-5525
Email: davy@iss.net

Full Copyright Statement

copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.