Intrusion Detection Working Group D. Curry/H. Debar
draft-ietf-idwg-idmef-xml-03.txt ISS/France Telecom
Expires: August 13, 2001 February 14, 2001

Intrusion Detection Message Exchange Format
Data Model and Extensible Markup Language (XML)
Document Type Definition

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.

1. 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].

This Internet-Draft describes a data model to represent information exported by intrusion detection systems, and explains the rationale for using this model. An implementation of the data model in the Extensible Markup Language (XML) is presented, an XML Document Type Definition is developed, and examples are provided.

TABLE OF CONTENTS

2. 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].

An "IDMEF-compliant application" is a program or program component, such as an analyzer or manager, that reads and/or writes messages in the format specified by this memo.

An "IDMEF document" is a message that adheres to the requirements specified by this memo, and that is exchanged by two or more IDMEF applications. "IDMEF message" is another term for an "IDMEF document."

3. Introduction

The Intrusion Detection Message Exchange Format (IDMEF) [3] is intended to be a standard data format that automated intrusion detection systems can use to report alerts about events that they deem suspicious. The development of this standard format will enable interoperability among commercial, open source, and research systems, allowing users to mix-and-match the deployment of these systems according to their strong and weak points to obtain an optimal implementation.

The most obvious place to implement the IDMEF is in the data channel between an intrusion detection analyzer (or "sensor") and the manager (or "console") to which it sends alarms. But there are other places where the IDMEF can be useful:

+ a single database system that could store the results from a variety of intrusion detection products would make it possible for data analysis and reporting activities to be performed on "the whole picture" instead of just a part of it;

+ an event correlation system that could accept alerts from a variety of intrusion detection products would be capable of performing more sophisticated cross-correlation and cross-confirmation calculations than one that is limited to a single product;

+ a graphical user interface that could display alerts from a variety of intrusion detection products would enable the user to monitor all of the products from a single screen, and require him or her to learn only one interface, instead of several; and

+ a common data exchange format would make it easier for different organizations (users, vendors, response teams, law enforcement) to not only exchange data, but also communicate about it.

The diversity of uses for the IDMEF needs to be considered when selecting its method of implementation.

3.1 About the IDMEF Data Model

The IDMEF data model is an object-oriented representation of the alert data sent to intrusion detection managers by intrusion detection analyzers.

3.1.1 Problems Addressed by the Data Model

The data model addresses several problems associated with representing intrusion detection alert data:

+ Alert information is inherently heterogeneous. Some alerts are defined with very little information, such as origin, destination, name, and time of the event. Other alerts provide much more information, such as ports or services, processes, user information, and so on. The data model that represents this information must be flexible to accomodate different needs.

  An object-oriented model is naturally extensible via aggregation and subclassing. If an implementation of the data model extends it with new classes, either by aggregation or subclassing, an implementation that does not understand these extensions will still be able to understand the subset of information that is defined by the data model. Subclassing and aggregation provide extensibility while preserving the consistency of the model.

+ Intrusion detection environments are different. Some analyzers detect attacks by analyzing network traffic; others use operating system logs or application audit trail information. Alerts for the same attack, sent by analyzers with different information sources, will not contain the same information.

  The data model defines support classes that accommodate the differences in data sources among analyzers. In particular, the notion of source and target for the alert are represented by the combination of Node, Process, Service, and User classes.

+ Analyzer capabilities are different. Depending on the environment, one may install a lightweight analyzer that provides little information in its alerts, or a more complex analyzer that will have a greater impact on the running system but provide more detailed alert information. The data model must allow for conversion to formats used by tools other than intrusion detection analyzers, for the purpose of further processing the alert information.

  The data model defines extensions to the basic schema that allow

  carrying both simple and complex alerts. Extensions are accomplished through subclassing or association of new classes.

+ Operating environments are different. Depending on the kind of network or operating system used, attacks will be observed and reported with different characteristics. The data model should accommodate these differences.

  Significant flexibility in reporting is provided by the Node and Service support classes. If additional information must be reported, subclasses may be defined that extend the data model with additional attributes.

+ Commercial vendor objectives are different. For various reasons, vendors may wish to deliver more or less information about certain types of attacks.

  The object-oriented approach allows this flexibility while the subclassing rules preserve the integrity of the model.

3.1.2 Data Model Design Goals

The data model was designed to provide a standard representation of alerts in an unambiguous fashion, and to permit the relationship between simple and complex alerts to be described.

3.1.2.1 Representing Events

The goal of the data model is to provide a standard representation of the information that an intrusion detection analyzer reports when it detects an occurence of some unusual event(s). These alerts may be simple or complex, depending on the capabilities of the analyzer that creates them.

3.1.2.2 Content-Driven

The design of the data model is content-driven. This means that new objects are introduced to accomodate additional content, not semantic differences between alerts. This is an important goal, as the task of classifying and naming computer vulnerabilities is both extremely difficult and very subjective.

The data model must be unambiguous. This means that while we allow analyzers to be more or less precise than one another (i.e., one analyzer may report more information about an event than another), we do not allow them to produce contradictory information in two alerts describing the same event (i.e., the common subset of information reported by both analyzers must be identical and inserted in the same placeholders within the alert data structure). Of course, it is

always possible to insert all "interesting" informaton about an event in extension fields of the alert instead of in the fields where it belongs; however, such practice reduces interoperability and should be avoided whenever possible.

3.1.2.3 Relationship Between Alerts

Intrusion detection alerts can be transmitted at several levels. This Internet-Draft applies to the entire range, from very simple alerts (e.g., those alerts that are the result of a single action or operation in the system, such as a failed login report) to very complex ones (e.g., the aggregation of several events causing an alert to be generated).

As such, the data model must provide a way to describe the relationship between simple and complex alerts.

3.2 About the IDMEF XML Implementation

Two implementations of the IDMEF were originally proposed to the IDWG: 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.

These proposed implementations were reviewed by the IDWG at its September 1999 and February 2000 meetings; it was decided at the February meeting that the XML solution was best at fulfilling the IDWG requirements. A comparison of the two proposals, and a rationale for this decision, are presented in [4].

3.2.1 The Extensible Markup Language

The Extensible Markup Language (XML) [5] is a simplified version of the Standard Generalized Markup Language (SGML), a syntax for specifying text markup defined by the ISO 8879 standard. XML is gaining widespread attention as a language for representing and exchanging documents and data on the Internet, and as the solution to most of the problems inherent in HyperText Markup Language (HTML). XML was published as a recommendation by the World Wide Web Consortium (W3C) on February 10, 1998.

XML is a metalanguage -- a language for describing other languages -- that enables an application to define its own markup. XML allows the definition of customized markup languages for different types of documents and different applications. This differs from HTML, in which there is a fixed set of identifiers with preset meanings that must be "adapted" for specialized uses. Both XML and HTML use elements (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 "platypus" in XML, or it might have no meaning at all, depending on the particular application.

NOTE: XML provides both a syntax for declaring document markup and structure (i.e., defining elements and attributes, specifying the order in which they appear, and so on) and a syntax for using that markup in documents. Because markup declarations look radically different from markup, many people are confused as to which syntax is called XML. The answer is that they both are, because they are actually both part of the same language.

For clarity in this document, we will use the terms "XML" and "XML documents" when speaking in the general case, and the term "IDMEF markup" when speaking specifically of the elements (tags) and attributes that describe IDMEF messages.

The publication of XML was followed by the publication of a second recommendation [6] by the World Wide Web Consortium, defining the use of namespaces in XML documents. An XML namespace is a collection of names, identified by a Universal Resource Identifier (URI) [7]. When using namespaces, each tag is identified with the namespace it comes from, allowing tags from different namespaces with the same names to occur in the same document. For example, a single document could contain both "usa:football" and "europe:football" tags, each with different meanings.

In anticipation of the widespread use of XML namespaces, this memo includes the definition of the URI to be used to identify the IDMEF namespace [8].

3.2.2 Rationale for Implementing IDMEF in XML

XML-based applications are being used or developed for a wide variety of purposes, including electronic data interchange in a variety of fields, financial data interchange, electronic business cards, calendar and scheduling, enterprise software distribution, web "push" technology, and markup languages for chemistry, mathematics, music, molecular dynamics, astronomy, book and periodical publishing, web publishing, weather observations, real estate transactions, and many others.

XML's flexibility makes it a good choice for these applications; that same flexibility makes it a good choice for implementing the IDMEF as well. Other, more specific reasons for choosing XML to implement the IDMEF are:

+ XML allows a custom language to be developed specifically for the purpose of describing intrusion detection alerts. It also defines a standard way to extend this language, either for later revisions of this document ("standard" extensions), or for vendor-specific

use ("non-standard" extensions).

+ Software tools for processing XML documents are widely available, in both commercial and open source forms. Numerous tools and APIs for parsing and/or validating XML are available in a variety of languages, including Java, C, C++, Tcl, Perl, Python, and GNU Emacs Lisp. Widespread access to tools will make adoption of the IDMEF by product developers easier, and hopefully, faster.

+ XML meets IDMEF Requirement 5.1, that message formats support full internationalization and localization. The XML standard requires support for both the UTF-8 and UTF-16 encodings of ISO/IEC 10646 (Universal Multiple-Octet Coded Character Set, "UCS") and Unicode, making all XML applications (and therefore all IDMEF-compliant applications) compatible with these common character encodings.

  XML also provides support for specifying, on a per-element basis, the language in which the element's content is written, making IDMEF easy to adapt to "Natural Language Support" versions of a product.

+ XML meets IDMEF Requirement 5.2, that message formats must support filtering and aggregation. XML's integration with XSL, a style language, allows messages to be combined, discarded, and rearranged.

+ Ongoing XML development projects, in the W3C and elsewhere, will provide object-oriented extensions, database support, and other useful features. If implemented in XML, the IDMEF immediately gains these features as well.

+ XML is free, with no license, no license fees, and no royalties.

4. Notational Conventions and Formatting Issues

This document uses three notations: Unified Modeling Language to describe the data model, XML to describe the markup used in IDMEF documents, and IDMEF markup to represent the documents themselves.

This section describes these notations in sufficient detail that readers unfamiliar with them can understand the document. Note, however, that these descriptions are not comprehensive; they only cover the components of the notations used by the data model and document format.

This section also explains several document formatting issues that apply to XML and IDMEF documents, including formats for particular data types, special character and whitespace processing, character sets, and languages.

4.1 Unified Modeling Language

The IDMEF data model is described using the Unified Modeling Language (UML) [9]. UML provides a simple framework to represent entities and their relationships. UML defines entities as classes. In this document, we have identified several classes and their associated attributes. The symbols used in this document to represent classes and attributes are shown in Figure 4.1.

```
+-------------+
| Class Name  |     <----- Name of class
+-------------+
| Attribute 1 |     <----- Name of first attribute
| ...         |
| Attribute N |     <----- Name of nth attribute
+-------------+
```

Figure 4.1 - Symbols representing classes and attributes

Note that attributes for a class may not appear in all diagrams in which the class is used.

4.1.1 Relationships

The IDMEF model currently uses only two of the relationship types defined by UML: inheritance and aggregation.

4.1.1.1 Inheritance Relationship

Inheritance denotes a superclass/subclass type of relationship where the subclass inherits all the attributes, operations, and

```
                 +-------------+
                 | Publication |
                 +-------------+
                 | publisher   |
                 | pubDate     |
                 +-------------+
                       /_\
                        |
              +---------+--------+
              |                  |
         +----------+      +----------+
         | Magazine |      |   Book   |
         +----------+      +----------+
         | name     |      | title    |
         |          |      | author   |
         +----------+      +----------+
```

Figure 4.2 - Inheritance relationships

relationships of the superclass. This type of relationship is also called a "is-a" or "kind-of" relationship. Subclasses may have additional attributes or operations that apply only to the subclass, and not to the superclass.

In this document, inheritance is denoted by the /_\ symbol. In Figure 4.2 above, we are showing that Book and Magazine are two types of Publication. Book inherits all the attributes of Publication, plus all of its own attributes (thus, it has four attributes in total); as does Magazine (giving it three attributes in total).

4.1.1.2 Aggregation Relationship

Aggregation is a form of association in which the whole is related to its parts. This type of relationship is also referred to as a "part-of" relationship. In this case, the aggregate class contains all of its own attributes and as many of the attributes associated with its parts as required and specified by the occurrence indicators (see Section 4.1.2).

```
+----------+
|   Book   |
+----------+       0..1 +--------------+
| title    |<>----------| Preface      |
| author   |            +--------------+
|          |       1..* +--------------+
|          |<>----------| Chapter      |
|          |            +--------------+
|          |       0..* +--------------+
|          |<>----------| Appendix     |
|          |            +--------------+
|          |       0..1 +--------------+
|          |<>----------| Bibliography |
|          |            +--------------+
|          |            +--------------+
|          |<>----------| Index        |
|          |            +--------------+
+----------+
```

Figure 4.3 - Aggregation relationships

In this document, the symbol <> is used to indicate aggregation. It is placed at the end of the association line closest to the aggregate (whole) class. In Figure 4.3 above, we are showing that a Book is made up of pieces called Preface, Chapter, Appendix, Bibliography, and Index.

4.1.2 Occurrence Indicators

Occurence indicators show the number of objects within a class that are linked to one another by an aggregation relationship. They are placed at the end of the association line closest to the part they refer to. Occurence indicators, as used in this document, are:

```
    n     exactly "n" (left blank if n=1)
 0..*     zero or more
 1..*     one or more
 0..1     zero or one (i.e., "optional")
 n..m     between "n" and "m" (inclusive)
```

In Figure 4.3 above, the Book:

+ may have no Preface or one Preface;
+ must have at least one Chapter, but may have more;
+ may have any number of Appendixes; and
+ must have exactly one Index.

4.2 XML Document Type Definitions

XML Document Type Definitions (DTDs) are used to declare the markup for a document. This includes the different pieces of information the document will contain (the elements), characteristics of that information (the attributes), and the relationship between the pieces (the content model).

Section 9 of this document contains the complete IDMEF DTD.

4.2.2 Element Declarations

Elements are the main part of a document's markup; they define the names of the pieces of the document, and the content model for those pieces.

```
<!ELEMENT Book (
    Preface, Chapter, Appendix, Bibliography, Index
  )>
```

In this example, the "Book" element is defined to consist of exactly one Preface, one Chapter, one Appendix, one Bibliography, and one Index. Furthermore, these parts must appear in this order (e.g., the Index cannot come before the Bibliography).

The XML document associated with this DTD might look like this:

```
<Book>
  <Preface>
    ...
  </Preface>
  <Chapter>
    ...
```



```
    </Chapter>
    <Appendix>
      ...
    </Appendix>
    <Index>
      ...
    </Index>
  </Book>
```

NOTE: XML is for the most part a free-format language; the line breaks and indentation used in the examples are for the purpose of improving readability only.

4.2.2.1 Occurrence Indicators

In the example above, Book must contain exactly one of each part -- it cannot have more than one Chapter, the Preface is not optional, and so on. This is not a very good representation of real-life books.

XML provides occurrence indicators to make it possible to represent more complex content models. The occurrence indicators are:

```
   ?         the content may appear either once or not at all
   *         the content may appear one or more times or not at all
   +         the content must appear at least once, and may appear
             more than once
   [none]    the content must appear exactly once
```

Occurence indicators allow us to revise our Book content model

```
<!ELEMENT Book (
    Preface?, Chapter+, Appendix*, Bibliography?, Index
  )>
```

Now a Book may contain an optional Preface, one or more Chapters, any number of Appendixes, an optional Bibliography, and an Index. The parts must still occur in this order.

4.2.2.2 Alternative Content and Grouping

To allow the creation of arbitrarily complex content models, XML also provides:

+ alternatives, specified with the '|' character
+ parentheses, to permit grouping of elements
+ occurence indicators may also be used on parenthesized groups

For example:


```
<!ELEMENT x (a, (b | c | d), e)* >
```

would allow all of the following:

```
<x>         <x>         <x>         <x>         <x>
  <a/>        <a/>        <a/>        <a/>      </x>
  <b/>        <d/>        <c/>        <b/>
  <e/>        <e/>        <e/>        <e/>
</x>        </x>          <a/>        <a/>
                          <c/>        <c/>
                          <e/>        <e/>
                        </x>          <a/>
                                      <d/>
                                      <e/>
                                    </x>
```

The example above also introduces the "<tag/>" notation; this is used in XML to denote empty content. It is more or less equivalent to "<tag></tag>" (the differences are beyond the scope of this document).

4.2.2.3 Element Content

An XML document has a tree structure. One element at the top is the parent of all other elements (e.g., Book), there are some number of other elements all with parents and children, and then at the bottom of the tree, there are some number of elements that have no children. These are the elements that contain the document content.

XML DTDs do not support data types such as integer, real, string, and so on (more on this later). However, they do require some indication of the type(s) of content that an element will contain. There are several types available, but only two are used in the IDMEF:

PCDATA
: An XML processor will find only text (parsed character data) in this element, no tags or entity references (see Section 4.2.4). This is the content type for all but one of the elements at the bottom of the IDMEF document tree.

ANY
: The element may contain anything -- text, other tags, entity references, etc. This is the content type for the AdditionalData element (see Section 5.2.4.5).

4.2.3 Attribute Declarations

Attributes allow data to be associated with an element. The decision to put data in an attribute or a child element is mostly one of style, although consideration should be given to the type and

quantity of data as well. Attributes are, generally, used for small, atomic data and elements are used for large or composite data.

Attributes are declared with their name, their content type, and their attribute type, as shown below:

```
<!ATTLIST Book
    title         CDATA               #REQUIRED
    author        CDATA               #REQUIRED
  >
```

The declaration above defines two attributes of the Book element, title and author. Both may contain character data, and both are required. These might be given as follows in an XML document:

```
<Book title="The Cat in the Hat" author="Dr. Seuss">
```

4.2.3.1 Attribute Types

There are four attribute types:

#REQUIRED
: The attribute is required, and has no default value. The XML document must specify a value for it.

#IMPLIED
: The attribute is optional, and has no default value.

#FIXED [value]
: The attribute must always have the default value "[value]." It is an error to specify the attribute with any other value. When an XML processor encounters an omitted attribute, it will behave as though it were present with the declared default value.

[value]
: The attribute is optional, and has a default value of "[value]." When an XML processor encounters an omitted attribute, it will behave as though it were present with the default value.

4.2.3.2 Attribute Content

There are a variety of attribute content types defined, but only two are used in the IDMEF:

CDATA
: An attribute of this type contains character data (text); tags and entity references (see Section 4.2.4) are not processed.

[values]
: An attribute may also be declared with a list of acceptable

values; this functions somewhat like an enumerated type. For example:

```
<!ATTLIST Person
   gender     "unknown|male|female"   "unknown"
  >
```

The gender attribute may have one of three values; if a Person tag appears without a gender attribute, the XML processor will behave as though it did have one, with value "unknown."

4.2.4 Entity Declarations

Entities allow symbols to be defined that will be replaced with other text when processed. There are two types of entities, "general" and "parameter." General entities are for use within XML document content; for example:

```
<!ENTITY idmef "Intrusion Detection Message Exchange Format">
```

Entities are referenced by bracketing them with the characters '&' and ';' -- whenever "&idmef;" appears in the XML document from the example above, it will be replaced with the text "Intrusion Detection Message Exchange Format". General entities (and a special case of them called character references) are used extensively in handling special characters (see Sections 4.3.2.1 and 4.3.2.2).

Paramter entities are for use within DTDs (they are not recognized in document content), and are declared and referenced in a slightly different way. The declaration includes a '%' symbol before the entity name, and they are referenced by bracketing them with the characters '%' (instead of '&') and ';'. For example, attributes that must appear on every element are declared in a parameter entity:

```
<!ENTITY % attlist.global     "
    xmlns           CDATA     #FIXED     'urn:iana:xml:ns:idmef'
    xmlns:idmef     CDATA     #FIXED     'urn:iana:xml:ns:idmef'
    xml:space       (default | preserve) 'default'
    xml:lang        NMTOKEN   #IMPLIED
  ">
```

and then referenced in each attribute list declaration:

```
<!ATTLIST IDMEF-Message
    %attlist.global;
  >
<!ATTLIST Alert
    %attlist.global;
  >
```


4.3 XML Documents

This section describes a number of XML document formatting rules; these rules apply to IDMEF documents as well.

4.3.1 The Document Prolog

The "prolog" of an XML document, that part that precedes anything else, consists of the XML declaration and the document type declaration.

4.3.1.1 XML Declaration

Every XML document (and therefore every IDMEF document) starts with an XML declaration. The XML declaration specifies the version of XML being used; it may also specify the character encoding being used.

The XML declaration looks like:

```
<?xml version="1.0" ?>
```

If a character encoding is specified, the declaration looks like:

```
<?xml version="1.0" encoding="charset" ?>
```

where "charset" is the name of the character encoding in use (see Section 4.3.2). If no encoding is specified, UTF-8 is assumed.

IDMEF documents being exchanged between IDMEF-compliant applications MUST begin with an XML declaration, and MUST specify the XML version in use. Specification of the encoding in use is RECOMMENDED.

IDMEF-compliant applications MAY choose to omit the XML declaration internally to conserve space, adding it only when the message is sent to another destination (e.g., a web browser). This practice is NOT RECOMMENDED unless it can be accomplished without loss of each message's version and encoding information.

4.3.1.2 IDMEF DTD Formal Public Identifier

The formal public identifier (FPI) for the IDMEF Document Type Definition described in this memo is:

"-//IETF//DTD RFCxxxx IDMEF v0.3//EN"

NOTE: The "RFCxxxx" text in the FPI value will be replaced with the actual RFC number, if this memo is published as an RFC.

This FPI MUST be used in the document type declaration within an XML

document referencing the IDMEF DTD defined by this memo, as shown in the following section.

4.3.1.3 IDMEF DTD Document Type Declaration

The document type declaration for an XML document referencing the IDMEF DTD defined by this memo will usually be specified in one of the following ways:

```
<!DOCTYPE IDMEF-Message PUBLIC
    "-//IETF//DTD RFCxxxx IDMEF v0.3//EN">
```

The last component of the document type declaration is the formal public identifier (FPI) specified in the previous section.

```
<!DOCTYPE IDMEF-Message SYSTEM
    "/some/path/to/the/idmef-message.dtd">
```

The last component of the document type declaration is a URI that points to a copy of the Document Type Definition.

In order to be valid (see Section 7.1), an XML document must contain a document type declaration. However, this represents significant overhead to an IDMEF-compliant application, both in the bandwidth it consumes as well as the requirements it places on the XML processor (not only to parse the declaration itself, but also to parse the DTD it references).

Implementors MAY decide, therefore, to have analyzers and managers agree out-of-band on the particular document type definition they will be using to exchange messages (the standard one as defined here, or one with extensions), and then omit the document type declaration from IDMEF messages. The method for negotiating this agreement is outside the scope of this document. Note that great care must be taken in negotiating any such agreements, as the manager may have to accept messages from many different analyzers, each using a DTD with a different set of extensions.

4.3.2 Character Data Processing in XML and IDMEF

A document's XML declaration (see Section 4.3.1.1) specifies the character encoding to be used in the document, as follows:

```
<?xml version="1.0" encoding="charset" ?>
```

where "charset" is the name of the character encoding, as registered with the Internet Assigned Numbers Authority (IANA), see [10].

The XML standard requires that XML processors support the UTF-8 and UTF-16 encodings of ISO/IEC 10646 (UCS) and Unicode, making all XML

applications (and therefore, all IDMEF-compliant applications) compatible with these common character encodings. The XML standard also permits other character encodings to be used (e.g., UTF-7, UTF-8, UTF-32). However, support for these encodings is not guaranteed to be present in all XML applications.

For portability reasons, IDMEF-compliant applications SHOULD NOT use, and IDMEF messages SHOULD NOT be encoded in, character encodings other than UTF-8 and UTF-16. Consistent with the XML standard, if no encoding is specified for an IDMEF message, UTF-8 is assumed.

NOTE: The ASCII character set is a subset of the UTF-8 encoding, and therefore may be used to encode IDMEF messages.

Per the XML standard, IDMEF documents encoded in UTF-16 MUST begin with the Byte Order Mark described by ISO/IEC 10646 Annex E and Unicode Appendix B (the "ZERO WIDTH NO-BREAK SPACE" character, #xFEFF).

4.3.2.1 Character Entity References

Within XML documents, certain characters have special meanings in some contexts. To include the actual character itself in one of these contexts, a special escape sequence, called an entity reference, must be used.

The characters that sometimes need to be escaped, and their entity references, are:

Character	Entity Reference
&	&
<	<
>	>
"	"
'	'

It is RECOMMENDED that IDMEF-compliant applications use the entity reference form whenever writing these characters in data, to avoid any possibility of misinterpretation.

4.3.2.2 Character Code References

Any character defined by the ISO/IEC 10646 and Unicode standards may be included in an XML document by the use of a character reference. A character reference is started with the characters '&' and '#', and ended with the character ';'. Between these characters, the character code for the character inserted.

If the character code is preceded by an 'x' it is interpreted in hexadecimal (base 16), otherwise, it is interpreted in decimal (base

10). For instance, the ampersand (&) is encoded as & or & and the less-than sign (<) is encoded as < or <.

Any one-, two-, or four-byte character specified in the ISO/IEC 10646 and Unicode standards can be included in a document using this technique.

4.3.2.3 White Space Processing

XML preserves white space by default. The XML processor passes all white space characters to the application unchanged. This is much different from HTML (and SGML), in which, although the space/no space distinction is meaningful, the one space/many spaces distinction is not.

XML allows elements to identify the importance of white space in their content by using the "xml:space" attribute:

<tag xml:space="action">

where "action" is either "default" or "preserve."

If "action" is "preserve," the application MUST treat all white space in the element's content as significant. If "action" is "default," the application is free to do whatever it normally would with white space in the element's content.

The intent declared with the "xml:space" attribute is considered to apply to all attributes and content of the element where it is specified (including sub-elements), unless overridden with an instance of "xml:space" on another element within that content.

All IDMEF elements support the "xml:space" attribute.

4.3.3 Languages in XML and IDMEF

XML allows elements to identify the language their content is written in by using the "xml:lang" attribute:

<tag xml:lang="langcode">

where "langcode" is a language tag as described in RFC 3066 [11].

The intent declared with the "xml:lang" attribute is considered to apply to all attributes and content of the element where it is specified (including sub-elements), unless overridden with an instance of "xml:lang" on another element within that content.

IDMEF-compliant applications SHOULD specify the language in which their contents are encoded; in general this can be done by specifying

the "xml:lang" attribute for the top-level element and letting all other elements "inherit" that definition.

If no language is specified for an IDMEF message, English SHALL be assumed.

All IDMEF tags support the "xml:lang" attribute.

4.3.4 Inheritance and Aggregation

XML DTDs do not support inheritance as used by the IDMEF data model (i.e., there is no support for "kind-of" relationships). This does not present a major problem in practice; aggregation relationships have been used instead to implement these relationships with little loss of functionality.

As a note of interest, XML Schemas, currently being developed by the W3C, will provide support for inheritance, as well as stronger data typing and other useful features. Future versions of the IDMEF will probably use XML Schemas instead of DTDs; this is not currently possible because the XML Schema Recommendation has not been finalized.

4.4 IDMEF Data Types

Within an XML IDMEF message, all data will be expressed as "text" (as opposed to "binary"), since XML is a text formatting language. We provide typing information for the attributes of the classes in the data model however, to convey to the reader the type of data the model expects for each attribute.

Each data type in the model has specific formatting requirements in an XML IDMEF message; these requirements are set forth in this section.

4.4.1 Integers

Integer attributes are represented by the INTEGER data type. Integer data MUST be encoded in Base 10 or Base 16.

Base 10 integer encoding uses the digits '0' through '9' and an optional sign ('+' or '-'). For example, "123", "-456".

Base 16 integer encoding uses the digits '0' through '9' and 'a' through 'f' (or their upper case equivalents), and is preceded by the characters "0x". For example, "0x1a2b".

4.4.2 Real Numbers

Real (floating-point) attributes are represented by the REAL data type. Real data MUST be encoded in Base 10.

Real encoding is that of the POSIX "strtod" library function: an optional sign ('+' or '-') followed by a non-empty string of decimal digits, optionally containing a radix character, then an optional exponent part. An exponent part consists of an 'e' or 'E', followed by an optional sign, followed by one or more decimal digits. For example, "123.45e02", "-567,89e-03".

IDMEF-compliant applications MUST support both the '.' and ',' radix characters.

4.4.3 Characters and Strings

Single-character attributes are represented by the CHARACTER data type. Multi-character attributes of known length are represented by the STRING data type.

Character and string data have no special formatting requirements, other than the need to occasionally use character references (see Sections 4.3.2.1 and 4.3.2.2) to represent special characters.

4.4.4 Bytes

Binary data is represented by the BYTE (and BYTE[]) data type.

Binary data MUST be encoded in its entirety using character code references (see Section 4.3.2.2).

4.4.5 Enumerated Types

Enumerated types are represented by the ENUM data type, and consist of an ordered list of acceptable values. Each value has a rank (number) and a representing keyword.

Within IDMEF XML messages, the enumerated type keywords are used as attribute values, and the ranks are ignored. However, those IDMEF-compliant applications that choose to represent these values internally in a numeric format MUST use the rank values identified in this memo.

4.4.6 Date-Time Strings

Date-time strings are represented by the DATETIME data type. Each date-time string identifies a particular instant in time; ranges are

not supported.

Date-time strings are formatted according to a subset of ISO 8601:2000 [12], as show below. Section references in parentheses refer to sections of the ISO 8601:2000 standard.

1. Dates MUST be formatted as follows:

   YYYY-MM-DD

   where YYYY is the four- digit year, MM is the two-digit month (01-12), and DD is the two- digit day (01-31). (Section 5.2.1.1, "Complete representation -- Extended format.")

2. Times MUST be formatted as follows:

   hh:mm:ss

   where hh is the two-digit hour (00-24), mm is the two-digit minute (00-59), and ss is the two-digit second (00-60). (Section 5.3.1.1, "Complete representation -- Extended format.")

   Note that midnight has two representations, 00:00:00 and 24:00:00. Both representations MUST be supported by IDMEF-compliant applications, however, the 00:00:00 representation SHOULD be used whenever possible.

   Note also that this format accounts for leap seconds. Positive leap seconds are inserted between 23:59:59Z and 24:00:00Z and are represented as 23:59:60Z. Negative leap seconds are achieved by the omission of 23:59:59Z. IDMEF-compliant applications MUST support leap seconds.

3. Times MAY be formatted to include a decimal fraction of seconds, as follows:

   hh:mm:ss.ss or
   hh:mm:ss,ss

   As many digits as necessary may follow the decimal sign (at least one digit must follow the decimal sign). Decimal fractions of hours and minutes are not supported. (Section 5.3.1.3, "Representation of decimal fractions.")

   IDMEF-compliant applications MUST support the use of both decimal signs ('.' and ',').

   Note that the number of digits in the fraction part does not imply anything about accuracy -- i.e., "00.100000", "00,1000" and "00.1" are all equivalent.

4. Times MUST be formatted to include (a) an indication that the time

is in Coordinated Universal Time (UTC), or (b) an indication of the difference between the specified time and Coordinated Universal Time.

a. Times in UTC MUST be formatted by appending the letter 'Z' to the time string as follows:

```
hh:mm:ssZ
hh:mm:ss.ssZ
hh:mm:ss,ssZ
```

(Section 5.3.3, "Coordinated Universal Time (UTC) -- Extended format.")

b. If the time is ahead of or equal to UTC, a '+' sign is appended to the time string; if the time is behind UTC, a '-' sign is appended. Following the sign, the number of hours and minutes representing the different from UTC is appended, as follows:

```
hh:mm:ss+hh:mm
hh:mm:ss-hh:mm
hh:mm:ss.ss+hh:mm
hh:mm:ss.ss-hh:mm
hh:mm:ss,ss+hh:mm
hh:mm:ss,ss-hh:mm
```

The difference from UTC MUST be specified in both hours and minutes, even if the minutes component is 0. A "difference" of "+00:00" is equivalent to UTC. (Section 5.3.4.2, "Local time and the difference with Coordinated Universal Time -- Extended Format.")

5. Date-time strings are created by joing the date and time strings with the letter 'T', as shown below:

```
YYYY-MM-DDThh:mm:ssZ
YYYY-MM-DDThh:mm:ss.ssZ
YYYY-MM-DDThh:mm:ss,ssZ
YYYY-MM-DDThh:mm:ss+hh:mm
YYYY-MM-DDThh:mm:ss-hh:mm
YYYY-MM-DDThh:mm:ss.ss+hh:mm
YYYY-MM-DDThh:mm:ss.ss-hh:mm
YYYY-MM-DDThh:mm:ss,ss+hh:mm
YYYY-MM-DDThh:mm:ss,ss-hh:mm
```

(Section 5.4.1, "Complete representation -- Extended format.")

In summary, IDMEF date-time strings MUST adhere to one of the nine templates identified in Paragraph 5, above.

4.4.7 NTP Timestamps

NTP timestamps are represented by the NTPSTAMP data type, and are described in detail in [13] and [14]. An NTP timestamp is a 64-bit unsigned fixed-point number. The integer part is in the first 32 bits, and the fraction part is in the last 32 bits.

Within IDMEF messages, NTP timestamps MUST be encoded as two 32-bit hexadecimal values, separated by a period ('.'). For example, "0x12345678.0x87654321".

See also Section 7.4 for more information on NTP timestamps.

4.4.8 Port Lists

Port lists are represented by the PORTLIST data type, and consist of a comma-separated list of numbers (individual integers) and ranges (N-M means ports N through M, inclusive). Any combination of numbers and ranges may be used in a single list. For example, "5-25,37,42,43,53,69-119,123-514".

4.4.9 Unique Identifiers

There are two types of unique identifiers used in this specification. Both types are represented by STRING data types.

These identifiers are implemented as attributes on the relevant XML elements, and must have unique values as follows:

1. The Analyzer class' (Section 5.2.4.1) "analyzerid" attribute, if specified, MUST have a value that is unique across all analyzers in the intrusion detection environment.

   The "analyzerid" attribute is not required to be globally unique, only unique within the intrusion detection environment of which the analyzer is a member. It is permissible for two analyzers, in different intrusion detection environments, to have the same value for "analyzerid".

   The default value is "0", which indicates that the analyzer cannot generate unique identifiers.

2. The Alert, Heartbeat, Source, Target, Node, User, Process, Service, Address, and UserId classes' (Sections 5.2.2, 5.2.3, 5.2.4.3, 5.2.4.4, 5.2.6.1, 5.2.6.2, 5.2.6.3, 5.2.6.4, 5.2.6.1.1, and 5.2.6.2.1) "ident" attribute, if specified, MUST have a value that is unique across all messages sent by the individual analyzer.

   The "ident" attribute value MUST be unique for each particular

combination of data identifying an object, not for each object. Objects may have more than one ident value associated with them. For example, an identification of a host by name would have one value, while an identification of that host by address would have another value, and an identification of that host by both name and address would have still another value. Furthermore, different analyzers may produce different values for the same information.

The "ident" attribute by itself provides a unique identifier only among all the "ident" values sent by a particular analyzer. But when combined with the "analyzerid" value for the analyzer, a value that is unique across the intrusion detection environment is created. Again, there is no requirement for global uniqueness.

The default value is "0", which indicates that the analyzer cannot generate unique identifiers.

The specification of methods for creating the unique values contained in these attributes is outside the scope of this document.

5. The IDMEF Data Model and XML DTD

In this section, the individual components of the IDMEF data model are explained in detail. UML diagrams of the model are provided to show how the components are related to each other, and relevant sections of the XML DTD are presented to show how the model is translated into XML.

5.1 Data Model Overview

The relationship between the principal components of the data model is shown in Figure 5.1 on the following page (occurrence indicators and attributes are omitted).

The top-level class for all IDMEF messages is IDMEF-Message; each type of message is a subclass of this top-level class. There are presently two types of messages defined; Alerts and Heartbeats. Within each message, subclasses of the message class are used to provide the detailed information carried in the message.

It is important to note that the data model does not specify how an alert should be classified or identified. For example, a port scan may be identified by one analyzer as a single attack against multiple targets, while another analyzer might identify it as multiple attacks from a single source. However, once an analyzer has determined the type of alert it plans to send, the data model dictates how that alert should be formatted.


```
                           +---------------+
                           | IDMEF-Message |
                           +---------------+
                                  /_\
                                   |
      +----------------------------+-------+
      |                                    |
  +-------+   +----------------+      +-----------+   +----------------+
  | Alert |<>-|    Analyzer    |      | Heartbeat |<>-|    Analyzer    |
  +-------+   +----------------+      +-----------+   +----------------+
  |       |   +----------------+      |           |   +----------------+
  |       |<>-|   CreateTime   |      |           |<>-|   CreateTime   |
  |       |   +----------------+      |           |   +----------------+
  |       |   +----------------+      |           |   +----------------+
  |       |<>-|   DetectTime   |      |           |<>-| AdditionalData |
  |       |   +----------------+      +-----------+   +----------------+
  |       |   +----------------+
  |       |<>-|  AnalyzerTime  |
  |       |   +----------------+
  |       |   +---------+   +----------+
  |       |<>-| Source  |<>-|   Node   |
  |       |   +---------+   +----------+
  |       |   |         |   +----------+
  |       |   |         |<>-|   User   |
  |       |   |         |   +----------+
  |       |   |         |   +----------+
  |       |   |         |<>-| Process  |
  |       |   |         |   +----------+
  |       |   |         |   +----------+
  |       |   |         |<>-| Service  |
  |       |   +---------+   +----------+
  |       |   +---------+   +----------+
  |       |<>-| Target  |<>-|   Node   |
  |       |   +---------+   +----------+
  |       |   |         |   +----------+
  |       |   |         |<>-|   User   |
  |       |   |         |   +----------+
  |       |   |         |   +----------+
  |       |   |         |<>-| Process  |
  |       |   |         |   +----------+
  |       |   |         |   +----------+
  |       |   |         |<>-| Service  |
  |       |   +---------+   +----------+
  |       |   +----------------+
  |       |<>-| Classification |
  |       |   +----------------+
  |       |   +----------------+
  |       |<>-| AdditionalData |
  +-------+   +----------------+
```

Figure 5.1 - Data model overview

5.2 The Message Classes

The individual classes are described in the following sections.

5.2.1 The IDMEF-Message Class

All IDMEF messages are members of the IDMEF-Message class; it is the top-level class of the IDMEF data model, as well as the IDMEF DTD. There are currently two types (subclasses) of IDMEF-Message: Alert and Heartbeat.

Because DTDs do not support subclassing (see Section 4.3.4), the inheritance relationship between IDMEF-Message and the Alert and Heartbeat subclasses shown in Figure 5.1 has been replaced with an aggregate relationship. This is declared in the IDMEF DTD as follows:

```
<!ENTITY % attlist.idmef                  "
    version               CDATA                   #FIXED    '0.3'
  ">
<!ELEMENT IDMEF-Message                   (
    (Alert | Heartbeat)*
  )>
<!ATTLIST IDMEF-Message
    %attlist.idmef;
  >
```

The IDMEF-Message class has a single attribute:

version
The version of the IDMEF-Message specification (this document) this message conforms to. Applications specifying a value for this attribute MUST specify the value "0.3".

5.2.2 The Alert Class

Generally, every time an analyzer detects an event that it has been configured to look for, it sends an Alert message to its manager(s). Depending on the analyzer, an Alert message may correspond to a single detected event, or multiple detected events. Alerts occur asynchronously in response to outside events.

An Alert message is composed of several aggregate classes, as shown in Figure 5.2. The aggregate classes themselves are described in Sections 5.2.4 and 5.2.5.

The aggregate classes that make up Alert are:

Analyzer
Exactly one. Identification information for the analyzer that


```
            +--------------+
            |    Alert     |
            +--------------+            +------------------+
            | STRING ident |<>----------|     Analyzer     |
            | ENUM impact  |            +------------------+
            |              |            +------------------+
            |              |<>----------|    CreateTime    |
            |              |            +------------------+
            |              |       0..1 +------------------+
            |              |<>----------|    DetectTime    |
            |              |            +------------------+
            |              |       0..1 +------------------+
            |              |<>----------|   AnalyzerTime   |
            |              |            +------------------+
            |              |       0..* +------------------+
            |              |<>----------|      Source      |
            |              |            +------------------+
            |              |       0..* +------------------+
            |              |<>----------|      Target      |
            |              |            +------------------+
            |              |       1..* +------------------+
            |              |<>----------|  Classification  |
            |              |            +------------------+
            |              |       0..* +------------------+
            |              |<>----------|  AdditionalData  |
            |              |            +------------------+
            +--------------+
                   /_\
                    |
                    +----+------------+-------------+
                         |            |             |
              +-------------------+   |   +-------------------+
              |     ToolAlert     |   |   | CorrelationAlert  |
              +-------------------+   |   +-------------------+
                                      |
                             +-------------------+
                             |   OverflowAlert   |
                             +-------------------+
```

Figure 5.2 - The Alert Class

originated the alert.

CreateTime
Exactly one. The time the alert was created. Of the three times that may be provided with an Alert, this is the only one that is required.

DetectTime
Zero or one. The time the event(s) leading up to the alert was detected. In the case of more than one event, the time the first event was detected. In some circumstances, this may not be the

same value as CreateTime.

AnalyzerTime
Zero or one. The current time on the analyzer (see Section 7.3).

Source
Zero or more. The source(s) of the event(s) leading up to the alert.

Target
Zero or more. The target(s) of the event(s) leading up to the alert.

Classification
One or more. The "name" of the alert, or other information allowing the manager to determine what it is.

AdditionalData
Zero or more. Information included by the analyzer that does not fit into the data model. This may be an atomic piece of data, or a large amount of data provided through an extension to the IDMEF (see Section 6).

Because DTDs do not support subclassing (see Section 4.3.4), the inheritance relationship between Alert and the ToolAlert, CorrelationAlert, and OverflowAlert subclasses shown in Figure 5.2 has been replaced with an aggregate relationship.

Alert is represented in the XML DTD as follows:

```
<!ENTITY % attvals.impact                 "
    ( unknown | bad-unknown | not-suspicious | attempted-admin |
      successful-admin | attempted-dos | successful-dos |
      attempted-recon | successful-recon-limited |
      successful-recon-largescale | attempted-user |
      successful-user )
  ">
<!ELEMENT Alert                           (
    Analyzer, CreateTime, DetectTime?, AnalyzerTime?, Source*,
    Target*, Classification+, ToolAlert?, OverflowAlert?,
    CorrelationAlert?, AdditionalData*
  )>
<!ATTLIST Alert
    ident                 ID                      #IMPLIED
    impact                %attvals.impact;        'unknown'
  >
```

The Alert class has two attributes:

ident
Optional. A unique identifier for the alert, see Section 4.4.9.

impact
Optional. The evaluated impact of the event(s) leading up to the alert on the target. The permitted values for this attribute are shown below. The default value is "unknown".

Rank	Keyword	Description
0	unknown	Event's impact is unknown or cannot be determined
1	bad-unknown	Event's impact is unknown or cannot be determined, but is usually undesirable
2	not-suspicious	Event is not suspicious in any way
3	attempted-admin	Attempt to obtain administrator (super-user) privileges
4	successful-admin	Successful compromise of administrator privileges
5	attempted-dos	Attempted denial of service
6	successful-dos	Successful denial of service
7	attempted-recon	Attempted reconnaissance probe
8	successful-recon-limited	Successful reconnaissance probe; limited scope (e.g., one target)
9	successful-recon-largescale	Successful reconnaissance probe; large scope (e.g., many targets)
10	attempted-user	Attempt to obtain user-level privileges
11	successful-user	Successful compromise of user-level privileges

5.2.2.1 The ToolAlert Class

The ToolAlert class carries additional information related to the use of attack tools or malevolent programs such as Trojan horses, and can be used by the analyzer when it is able to identify these tools. It is intended to group one or more previously-sent alerts together, to say "these alerts were all the result of someone using this tool."

The ToolAlert class is composed of three aggregate classes, as shown in Figure 5.3.

The aggregate classes that make up ToolAlert are:

name
Exactly one. STRING. The reason for grouping the alerts together, for example, the name of a particular tool.

command
Zero or one. STRING. The command or operation that the tool was asked to perform, for example, a BackOrifice ping.

alertident
One or more. STRING. The list of alert identifiers that are related to this alert. Because alert identifiers are only unique

```
          +------------------+
          |      Alert       |
          +------------------+
                  /_\
                   |
          +------------------+
          |    ToolAlert     |
          +------------------+            +-------------------+
          |                  |<>----------|       name        |
          |                  |            +-------------------+
          |                  |       0..1 +-------------------+
          |                  |<>----------|      command      |
          |                  |            +-------------------+
          |                  |       1..* +-------------------+
          |                  |<>----------|    alertident     |
          |                  |            +-------------------+
          |                  |            | STRING analyzerid |
          |                  |            +-------------------+
          +------------------+
```

Figure 5.3 - The ToolAlert Class

across the alerts sent by a single analyzer, the optional "analyzerid" attribute of "alertident" should be used to identify the analyzer that a particular alert came from. If the "analyzerid" is not provided, the alert is assumed to have come from the same analyzer that is sending the ToolAlert.

This is represented in the XML DTD as follows:

```
<!ELEMENT ToolAlert                     (
    name, command?, alertident+
  )>
<!ELEMENT alertident          (#PCDATA) >
<!ATTLIST alertident
    analyzerid          CDATA                   #IMPLIED
  >
```

5.2.2.2 The CorrelationAlert Class

The CorrelationAlert class carries additional information related to the correlation of alert information. It is intended to group one or more previously-sent alerts together, to say "these alerts are all related."

The CorrelationAlert class is composed of two aggregate classes, as shown in Figure 5.4.

The aggregate classes that make up CorrelationAlert are:


```
          +------------------+
          |      Alert       |
          +------------------+
                  /_\
                   |
          +------------------+
          | CorrelationAlert |
          +------------------+            +-------------------+
          |                  |<>----------|       name        |
          |                  |            +-------------------+
          |                  |       1..* +-------------------+
          |                  |<>----------|    alertident     |
          |                  |            +-------------------+
          |                  |            | STRING analyzerid |
          |                  |            +-------------------+
          +------------------+
```

Figure 5.4 - The CorrelationAlert Class

name
Exactly one. STRING. The reason for grouping the alerts together, for example, a particular correlation method.

alertident
One or more. STRING. The list of alert identifiers that are related to this alert. Because alert identifiers are only unique across the alerts sent by a single analyzer, the optional "analyzerid" attribute of "alertident" should be used to identify the analyzer that a particular alert came from. If the "analyzerid" is not provided, the alert is assumed to have come from the same analyzer that is sending the CorrelationAlert.

This is represented in the XML DTD as follows.

```
<!ELEMENT CorrelationAlert              (
    name, alertident+
  )>
<!ELEMENT alertident          (#PCDATA) >
<!ATTLIST alertident
    analyzerid          CDATA                   #IMPLIED
  >
```

5.2.2.3 The OverflowAlert Class

The OverflowAlert carries additional information related to buffer overflow attacks. It is intended to enable an analyzer to provide the details of the overflow attack itself.

The OverflowAlert class is composed of three aggregate classes, as shown in Figure 5.5.


```
               +------------------+
               |      Alert       |
               +------------------+
                       /_\
                        |
               +------------------+
               |  OverflowAlert   |
               +------------------+            +---------+
               |                  |<>----------| program |
               |                  |            +---------+
               |                  |       0..1 +---------+
               |                  |<>----------| size    |
               |                  |            +---------+
               |                  |       0..1 +---------+
               |                  |<>----------| buffer  |
               |                  |            +---------+
               +------------------+
```

Figure 5.5 - The OverflowAlert Class

The aggregate classes that make up OverflowAlert are:

program
Exactly one. STRING. The program that the overflow attack attempted to run (note: this is not the program that was attacked).

size
Zero or one. INTEGER. The size, in bytes, of the overflow (i.e., the number of bytes the attacker sent).

buffer
Zero or one. BYTE[]. Some or all of the overflow data itself (dependent on how much the analyzer can capture).

This is represented in the XML DTD as follows:

```
<!ELEMENT OverflowAlert                 (
    program, size?, buffer?
  )>
```

5.2.3 The Heartbeat Class

Analyzers use Heartbeat messages to indicate their current status to managers. Heartbeats are intended to be sent in a regular period, say every ten minutes or every hour. The receipt of a Heartbeat message from an analyzer indicates to the manager that the analyzer is up and running; lack of a Heartbeat message (or more likely, lack of some number of consecutive Heartbeat messages) indicates that the analyzer or its network connection has failed.

All managers MUST support the receipt of Heartbeat messages; however, the use of these messages by analyzers is OPTIONAL. Developers of manager software SHOULD permit the software to be configured on a per-analyzer basis to use/not use Heartbeat messages.

A Heartbeat message is composed of several aggregate classes, as shown in Figure 5.6. The aggregate classes themselves are described in Sections 5.2.4 and 5.2.5.

```
            +--------------+
            |  Heartbeat   |
            +--------------+            +------------------+
            | STRING ident |<>----------|     Analyzer     |
            |              |            +------------------+
            |              |            +------------------+
            |              |<>----------|    CreateTime    |
            |              |            +------------------+
            |              |       0..1 +------------------+
            |              |<>----------|   AnalyzerTime   |
            |              |            +------------------+
            |              |       0..* +------------------+
            |              |<>----------|  AdditionalData  |
            |              |            +------------------+
            +--------------+
```

Figure 5.6 - The Heartbeat Class

The aggregate classes that make up Heartbeat are:

Analyzer
Exactly one. Identification information for the analyzer that originated the heartbeat.

CreateTime
Exactly one. The time the heartbeat was created.

AnalyzerTime
Zero or one. The current time on the analyzer (see Section 7.3).

AdditionalData
Zero or more. Information included by the analyzer that does not fit into the data model. This may be an atomic piece of data, or a large amount of data provided through an extension to the IDMEF (see Section 6).

This is represented in the XML DTD as follows:

```
<!ELEMENT Heartbeat                     (
    Analyzer, CreateTime, AnalyzerTime?, AdditionalData*
  )>
<!ATTLIST Heartbeat
    ident               CDATA                   '0'
```

```
      >
```

The Heartbeat class has one attribute:

ident
   Optional. A unique identifier for the heartbeat, see Section 4.4.9.

5.2.4 The Core Classes

The core classes -- Analyzer, Source, Target, Classification, and AdditionalData -- are the main parts of Alerts and Heartbeats, as shown in Figure 5.7.

```
+-----------+            +----------------+
| Heartbeat |    +-------|    Analyzer    |
+-----------+    |       +----------------+
|           |<>--+--+
+-----------+    |  |  0..* +----------------+
                 |  +-------| AdditionalData |
                 |          +----------------+
+-----------+    |
|   Alert   |    |     0..* +----------------+
+-----------+    |  +-------|     Source     |
|           |<>--+  |       +----------------+
|           |       |  0..* +----------------+
|           |       +-------|     Target     |
|           |       |       +----------------+
|           |<>-----+
+-----------+       |  1..* +----------------+
                    +-------| Classification |
                    |       +----------------+
                    |  0..* +----------------+
                    +-------| AdditionalData |
                            +----------------+
```

Figure 5.7 - The Core Classes

5.2.4.1 The Analyzer Class

The Analyzer class identifies the analyzer from which the alert or heartbeat message originates. Only one analyzer may be encoded for each alert or heartbeat, and that MUST be the analyzer at which the alert or heartbeat originated. Although the IDMEF data model does not prevent the use of hierarchical intrusion detection systems (where alerts get relayed up the tree), it does not provide any way to record the identity of the "relay" analyzers along the path from the originating analyzer to the manager that ultimately receives the alert.

The Analyzer class is composed of two aggregate classes, as shown in Figure 5.8.

```
+-------------------+
|     Analyzer      |
+-------------------+       0..1 +---------+
| STRING analyzerid |<>----------|  Node   |
|                   |            +---------+
|                   |       0..1 +---------+
|                   |<>----------| Process |
|                   |            +---------+
+-------------------+
```

Figure 5.8 - The Analyzer Class

The aggregate classes that make up Analyzer are:

Node
   Zero or one. Information about the host or device on which the analyzer resides (network address, network name, etc.).

Process
   Zero or one. Information about the process in which the analyzer is executing.

This is represented in the XML DTD as follows:

```
<!ELEMENT Analyzer                         (
    Node?, Process?
  )>
<!ATTLIST Analyzer
    analyzerid           CDATA                   '0'
  >
```

The Analyzer class has one attribute:

analyzerid
   Optional (but see below). A unique identifier for the analyzer, see Section 4.4.9.

   This attribute is only "partially" optional. If the analyzer makes use of the "ident" attributes on other classes to provide unique identifiers for those objects, then it MUST also provide a valid "analyzerid" attribute. This requirement is dictated by the uniqueness requirements of the "ident" attribute (they are unique only within the context of a particular "analyzerid"). If the analyzer does not make use of the "ident" attributes however, it may also omit the "analyzerid" attribute.

5.2.4.2 The Classification Class

The Classification class provides the "name" of an alert, or other information allowing the manager to determine what it is (for example, to decide whether or not to display the alert on-screen, what color to display it in, etc.).

The Classification class is composed of two aggregate classes, as shown in Figure 5.9.

```
+----------------+
| Classification |
+----------------+            +------+
| STRING origin  |<>----------| name |
|                |            +------+
|                |            +------+
|                |<>----------| url  |
|                |            +------+
+----------------+
```

Figure 5.9 - The Classification Class

The aggregate classes that make up Classification are:

name
   Exactly one. STRING. The name of the alert, from one of the origins listed below.

url
   Exactly one. STRING. A URL at which the manager (or the human operator of the manager) can find additional information about the alert. The URL may include an in-depth description of the attack, appropriate countermeasures, or other information deemed relevant by the vendor.

This is represented in the XML DTD as follows:

```
<!ENTITY % attvals.origin                 "
    ( unknown | bugtraqid | cve | vendor-specific )
  ">
<!ELEMENT Classification                  (
    name, url
  )>
<!ATTLIST Classification
    origin               %attvals.origin;        'unknown'
  >
```

The Classification class has one attribute:

origin
   Required. The source from which the name of the alert originates. The permitted values for this attribute are shown below. The

default value is "unknown".

Rank	Keyword	Description
0	unknown	Origin of the name is not known
1	bugtraqid	The SecurityFocus.com ("Bugtraq") vulnerability database identifier (http://www.securityfocus.com/vdb)
2	cve	The Common Vulnerabilities and Exposures (CVE) name (http://www.cve.mitre.org/)
3	vendor-specific	A vendor-specific name (and hence, URL); this can be used to provide product-specific information

5.2.4.3 The Source Class

The Source class contains information about the possible source(s) of the event(s) that generated an alert. An event may have more than one source (e.g., in a distributed denial of service attack).

The Source class is composed of four aggregate classes, as shown in Figure 5.10.

```
+------------------+
|      Source      |
+------------------+       0..1 +---------+
| STRING ident     |<>----------|  Node   |
| ENUM spoofed     |            +---------+
| STRING interface |       0..1 +---------+
|                  |<>----------|  User   |
|                  |            +---------+
|                  |       0..1 +---------+
|                  |<>----------| Process |
|                  |            +---------+
|                  |       0..1 +---------+
|                  |<>----------| Service |
|                  |            +---------+
+------------------+
```

Figure 5.10 - The Source Class

The aggregate classes that make up Source are:

Node
   Zero or one. Information about the host or device that is causing the events (network address, network name, etc.).

User
   Zero or one. Information about the user that is causing the event(s).

Process
Zero or one. Information about the process that is causing the event(s).

Service
Zero or one. Information about the network service involved in the event(s).

This is represented in the XML DTD as follows:

```
<!ENTITY % attvals.yesno                "
    ( unknown | yes | no )
  ">
<!ELEMENT Source                        (
    Node?, User?, Process?, Service?
  )>
<!ATTLIST Source
    ident               CDATA                   '0'
    spoofed             %attvals.yesno;         'unknown'
    interface           CDATA                   #IMPLIED
  >
```

The Source class has three attributes:

ident
Optional. A unique identifier for this source, see Section 4.4.9.

spoofed
Optional. An indication of whether the source is, as far as the analyzer can determine, a decoy. The permitted values for this attribute are shown below. The default value is "unknown".

Rank	Keyword	Description
0	unknown	Accuracy of source information unknown
1	yes	Source is believed to be a decoy
2	no	Source is believed to be "real"

interface
Optional. May be used by a network-based analyzer with multiple interfaces to indicate which interface this source was seen on.

5.2.4.4 The Target Class

The Target class contains information about the possible target(s) of the event(s) that generated an alert. An event may have more than one target (e.g., in the case of a port sweep).

The Target class is composed of four aggregate classes, as shown in Figure 5.11.


```
+------------------+
|      Target      |
+------------------+       0..1 +----------+
| STRING ident     |<>----------|   Node   |
| ENUM decoy       |            +----------+
| STRING interface |       0..1 +----------+
|                  |<>----------|   User   |
|                  |            +----------+
|                  |       0..1 +----------+
|                  |<>----------| Process  |
|                  |            +----------+
|                  |       0..1 +----------+
|                  |<>----------| Service  |
|                  |            +----------+
+------------------+
```

Figure 5.11 - The Target Class

The aggregate classes that make up Target are:

Node
Zero or one. Information about the host or device that is receiving the events (network address, network name, etc.).

User
Zero or one. Information about the user that is receiving the event(s).

Process
Zero or one. Information about the process that is receiving the event(s).

Service
Zero or one. Information about the network service involved in the event(s).

This is represented in the XML DTD as follows:

```
<!ENTITY % attvals.yesno                "
    ( unknown | yes | no )
  ">
<!ELEMENT Target                        (
    Node?, User?, Process?, Service?
  )>
<!ATTLIST Target
    ident               CDATA                   '0'
    decoy               %attvals.yesno;         'unknown'
    interface           CDATA                   #IMPLIED
  >
```

The Target class has three attributes:

ident
Optional. A unique identifier for this target, see Section 4.4.9.

decoy
Optional. An indication of whether the target is, as far as the analyzer can determine, a decoy. The permitted values for this attribute are shown below. The default value is "unknown".

Rank	Keyword	Description
0	unknown	Accuracy of target information unknown
1	yes	Target is believed to be a decoy
2	no	Target is believed to be "real"

interface
Optional. May be used by a network-based analyzer with multiple interfaces to indicate which interface this target was seen on.

5.2.4.5 The AdditionalData Class

The AdditionalData class is used to provide information that cannot be represented by the data model. AdditionalData can be used to provide atomic data (integers, strings, etc.) in cases where only small amounts of additional information need to be sent; it can also be used to extend the data model and the DTD to support the tranmission of complex data (such as packet headers). Detailed instructions for extending the data model and the DTD are provided in Section 6.

The AdditionalData element is declared in the XML DTD as follows:

```
<!ENTITY % attvals.adtype               "
    ( boolean | byte | character | date-time | integer |
      ntpstamp | portlist | real | string | xml )
  ">
<!ELEMENT AdditionalData            ANY >
<!ATTLIST AdditionalData
    type                %attvals.adtype;        'string'
    meaning             CDATA                   #IMPLIED
  >
```

The AdditionalData class has two attributes:

type
Required. The type of data included in the element content. The permitted values for this attribute are shown below. The default value is "string".

Rank	Keyword	Description
0	boolean	The element contains a boolean value,

		i.e., the strings "true" or "false"
1	byte	The element content is a single 8-bit byte (see Section 4.4.4)
2	character	The element content is a single character (see Section 4.4.3)
3	date-time	The element content is a date-time string (see Section 4.4.6)
4	integer	The element content is an integer (see Section 4.4.1)
5	ntpstamp	The element content is an NTP timestamp (see Section 4.4.7)
6	portlist	The element content is a list of ports (see Section 4.4.8)
7	real	The element content is a real number (see Section 4.4.2)
8	string	The element content is a string (see Section 4.4.3)
9	xml	The element content is XML-tagged data (see Section 6.2)

meaning
Optional. A string describing the meaning of the element content. These values will be vendor/implementation dependent; the method for ensuring that managers understand the strings sent by analyzer is outside the scope of this specification.

5.2.5 The Time Classes

The data model provides three classes for representing time. These classes are aggregates of the Alert and Heartbeat classes.

5.2.5.1 The CreateTime Class

The CreateTime class is used to indicate the date and time the alert or heartbeat was created by the analyzer. It is represented in the XML DTD as follows:

```
<!ELEMENT CreateTime            (#PCDATA) >
<!ATTLIST CreateTime
    ntpstamp            CDATA                   #REQUIRED
  >
```

The DATETIME format of the <CreateTime> element content is described in Section 4.4.6.

The CreateTime class has one attribute:

ntpstamp
Required. The NTP timestamp representing the same date and time as the element content. The NTPSTAMP format of this attribute's

value is described in Section 4.4.7.

If the date and time represented by the element content and the NTP timestamp differ (should "never" happen), the value in the NTP timestamp MUST be used.

5.2.5.2 The DetectTime Class

The DetectTime class is used to indicate the date and time the event(s) producing an alert was detected by the analyzer. In the case of more than one event, the time the first event was detected. (This may or may not be the same time as CreateTime; analyzers are not required to send alerts immediately upon detection). It is represented in the XML DTD as follows:

```
<!ELEMENT DetectTime           (#PCDATA) >
<!ATTLIST DetectTime
    ntpstamp              CDATA                  #REQUIRED
  >
```

The DATETIME format of the <DetectTime> element content is described in Section 4.4.6.

The DetectTime class has one attribute:

ntpstamp
Required. The NTP timestamp representing the same date and time as the element content. The NTPSTAMP format of this attribute's value is described in Section 4.4.7.

If the date and time represented by the element content and the NTP timestamp differ (should "never" happen), the value in the NTP timestamp MUST be used.

5.2.5.3 The AnalyzerTime Class

The AnalyzerTime class is used to indicate the current date and time on the analyzer. Its values should be filled in as late as possible in the message transmission process, ideally immediately before placing the message "on the wire." It is represented in the XML DTD as follows:

```
<!ELEMENT AnalyzerTime         (#PCDATA) >
<!ATTLIST AnalyzerTime
    ntpstamp              CDATA                  #REQUIRED
  >
```

The DATETIME format of the <AnalyzerTime> element content is described in Section 4.4.6.

The AnalyzerTime class has one attribute:

ntpstamp
Required. The NTP timestamp representing the same date and time as the element content. The NTPSTAMP format of this attribute's value is described in Section 4.4.7.

If the date and time represented by the element content and the NTP timestamp differ (should "never" happen), the value in the NTP timestamp MUST be used.

The use of <AnalyzerTime> to perform rudimentary time synchronization between analyzers and managers is discussed in Section 7.3.

5.2.6 The Support Classes

The support classes make up the major parts of the core classes, and are shared between them.

5.2.6.1 The Node Class

The Node class is used to identify hosts and other network devices (routers, switches, etc.).

The Node class is composed of three aggregate classes, as shown in Figure 5.12.

```
+---------------+
|     Node      |
+---------------+       0..1 +----------+
| STRING ident  |<>----------| location |
| ENUM category |            +----------+
|               |       0..1 +----------+
|               |<>----------|   name   |
|               |            +----------+
|               |       0..* +----------+
|               |<>----------| Address  |
|               |            +----------+
+---------------+
```

Figure 5.12 - The Node Class

The aggregate classes that make up Node are:

location
Zero or one. STRING. The location of the equipment.

name
Zero or one. STRING. The name of the equipment. This information MUST be provided if no Address information is given.

Address
Zero or more. The network or hardware address of the equipment. Unless a name (above) is provided, at least one address must be specified.

This is represented in the XML DTD as follows:

```
<!ENTITY % attvals.nodecat              "
    ( unknown | ads | afs | coda | dfs | dns | kerberos | nds |
      nis | nisplus | nt | wfw )
  ">
<!ELEMENT Node                          (
    location?, (name | Address), Address*
  )>
<!ATTLIST Node
    ident                 CDATA                  '0'
    category              %attvals.nodecat;      'unknown'
  >
```

The Node class has two attributes:

ident
Optional. A unique identifier for the node, see Section 4.4.9.

category
Optional. The "domain" to which the name information belongs, if relevant. The permitted values for this attribute are shown below. The default value is "unknown".

Rank	Keyword	Description
0	unknown	Domain unknown or not relevant
1	ads	Windows 2000 Advanced Directory Services
2	afs	Andrew File System (Transarc)
3	coda	Coda Distributed File System
4	dfs	Distributed File System (IBM)
5	dns	Domain Name System
6	kerberos	Kerberos realm
7	nds	Novell Directory Services
8	nis	Network Information Services (Sun)
9	nisplus	Network Information Services Plus (Sun)
10	nt	Windows NT domain
11	wfw	Windows for Workgroups

5.2.6.1.1 The Address Class

The Address class is used to represent network, hardware, and application addresses.

The Address class is composed of two aggregate classes, as shown in Figure 5.13.


```
+------------------+
|     Address      |
+------------------+            +---------+
| STRING ident     |<>----------| address |
| ENUM category    |            +---------+
| STRING vlan-name |       0..1 +---------+
| INTEGER vlan-num |<>----------| netmask |
|                  |            +---------+
+------------------+
```

Figure 5.13 - The Address Class

The aggregate classes that make up Address are:

address
Exactly one. STRING. The address information. The format of this data is governed by the category attribute.

netmask
Zero or one. STRING. The network mask for the address, if appropriate.

This is represented in the XML DTD as follows:

```
<!ENTITY % attvals.addrcat              "
    ( unknown | atm | e-mail | lotus-notes | mac | sna | vm |
      ipv4-addr | ipv4-addr-hex | ipv4-net | ipv4-net-mask |
      ipv6-addr | ipv6-addr-hex | ipv6-net | ipv6-net-mask )
  ">
<!ELEMENT Address                       (
    address, netmask?
  )>
<!ATTLIST Address
    ident                 ID                     #IMPLIED
    category              %attvals.addrcat;      'unknown'
    vlan-name             CDATA                  #IMPLIED
    vlan-num              CDATA                  #IMPLIED
  >
```

The Address class has four attributes:

ident
Optional. A unique identifier for the address, see Section 4.4.9.

category
Optional. The type of address represented. The permitted values for this attribute are shown below. The default value is "unknown".

Rank	Keyword	Description
0	unknown	Address type unknown

1	atm	Asynchronous Transfer Mode network address
2	e-mail	Electronic mail address (RFC 822)
3	lotus-notes	Lotus Notes e-mail address
4	mac	Media Access Control (MAC) address
5	sna	IBM Shared Network Architecture (SNA) address
6	vm	IBM VM ("PROFS") e-mail address
7	ipv4-addr	IPv4 host address in dotted-decimal notation (a.b.c.d)
8	ipv4-addr-hex	IPv4 host address in hexadecimal notation
9	ipv4-net	IPv4 network address in dotted-decimal notation, slash, significant bits (a.b.c.d/nn)
10	ipv4-net-mask	IPv4 network address in dotted-decimal notation, slash, network mask in dotted-decimal notation (a.b.c.d/w.x.y.z)
11	ipv6-addr	IPv6 host address
12	ipv6-addr-hex	IPv6 host address in hexadecimal notation
13	ipv6-net	IPv6 network address, slash, significant bits
14	ipv6-net-mask	IPv6 network address, slash, network mask

vlan-name
Optional. The name of the Virtual LAN to which the address belongs.

vlan-num
Optional. The number of the Virtual LAN to which the address belongs.

5.2.6.2 The User Class

The User class is used to describe users. It is primarily used as a "container" class for the UserId aggregate class, as shown in Figure 5.14.

```
+---------------+
|     User      |
+---------------+       1..* +--------+
| STRING ident  |<>----------| UserId |
| ENUM category |            +--------+
+---------------+
```

Figure 5.14 - The User Class

The aggregate class contained in User is:

UserId
One or more. Identification of a user, as indicated by its type attribute (see Section 5.2.6.2.1).

This is represented in the XML DTD as follows:

```
<!ENTITY % attvals.usercat              "
    ( unknown | application | os-device )
  ">
<!ELEMENT User                          (
    UserId+
  )>
<!ATTLIST User
    ident               ID                      #IMPLIED
    category            %attvals.usercat;       'unknown'
  >
```

The User class has two attributes:

ident
Optional. A unique identifier for the user, see Section 4.4.9.

category
Optional. The type of user represented. The permitted values for this attribute are shown below. The default value is "unknown".

Rank	Keyword	Description
0	unknown	User type unknown
1	application	An application user
2	os-device	An operating system or device user

5.2.6.2.1 The UserId Class

The UserId class provides specific information about a user. More than one UserId can be used within the User class to indicate attempts to transition from one user to another, or to provide complete information about a user's (or process') privileges.

The UserId class is composed of two aggregate classes, as shown in Figure 5.15.

The aggregate classes that make up UserId are:

name
Zero or one. STRING. A user or group name.

number
Zero or one. INTEGER. A user or group number.


```
+--------------+
|    UserId    |
+--------------+       0..1 +--------+
| STRING ident |<>----------|  name  |
| ENUM type    |            +--------+
|              |       0..1 +--------+
|              |<>----------| number |
|              |            +--------+
+--------------+
```

Figure 5.15 - The UserId Class

This is represented in the XML DTD as follows:

```
<!ENTITY % attvals.idtype               "
    ( current-user | original-user | target-user | user-privs |
      current-group | group-privs )
  ">
<!ELEMENT UserId                        (
    name | number | (name, number)
  )>
<!ATTLIST UserID
    ident               ID                      #IMPLIED
    type                %attvals.idtype;        'original-user'
  >
```

The UserId class has two attributes:

ident
Optional. A unique identifier for the user id, see Section 4.4.9.

type
Optional. The type of user information represented. The permitted values for this attribute are shown below. The default value is "original-user".

Rank	Keyword	Description
0	current-user	The current user id being used by the user or process. On Unix systems, this would be the "real" user id, in general.
1	original-user	The actual identity of the user or process being reported on. On those systems that (a) do some type of auditing and (b) support extracting a user id from the "audit id" token, that value should be used. On those systems that do not support this, and where the user has logged into the system, the "login id" should be used.
2	target-user	The user id the user or process is attempting to become. This would apply,

		on Unix systems for example, when the user attempts to use "su," "rlogin," "telnet," etc.
3	user-privs	Another user id the user or process has the ability to use. On Unix systems, this would be the "effective" user id. Multiple UserId elements of this type may be used to specify a list of privileges.
4	current-group	The current group id (if applicable) being used by the user or process. On Unix systems, this would be the "real" group id, in general.
5	group-privs	Another group id the group or process has the ability to use. On Unix systems, this would be the "effective" group id. On BSD-derived Unix systems, multiple UserId elements of this type would be used to include all the group ids on the "group list."

5.2.6.3 The Process Class

The Process class is used to describe processes being executed on sources, targets, and analyers.

The Process class is composed of five aggregate classes, as shown in Figure 5.16.

```
+--------------+
|   Process    |
+--------------+            +------+
| STRING ident |<>----------| name |
|              |            +------+
|              |       0..1 +------+
|              |<>----------| pid  |
|              |            +------+
|              |       0..1 +------+
|              |<>----------| path |
|              |            +------+
|              |       0..* +------+
|              |<>----------| arg  |
|              |            +------+
|              |       0..* +------+
|              |<>----------| env  |
|              |            +------+
+--------------+
```

Figure 5.16 - The Process Class

The aggregate classes that make up Process are:

name
: Exactly one. STRING. The name of the program being executed. This is a short name; path and argument information are provided elsewhere.

pid
: Zero or one. INTEGER. The process identifier of the process.

path
: Zero or one. STRING. The full path of the program being executed.

arg
: Zero or more. STRING. A command-line argument to the program. Multiple arguments may be specified (they are assumed to have occurred in the same order they are provided) with multiple uses of arg.

env
: Zero or more. STRING. An environment string associated with the process; generally of the format "VARIABLE=value". Multiple environment strings may be specified with multiple uses of env.

This is represented in the XML DTD as follows:

```
<!ELEMENT Process                          (
   name, pid?, path?, arg*, env*
  )>
<!ATTLIST Process
   ident                ID                      #IMPLIED
  >
```

The Process class has one attribute:

ident
: Optional. A unique identifier for the process, see Section 4.4.9.

5.2.6.4 The Service Class

The Service class describes network services on sources and targets. It can identify services by name, port, and protocol. When Service occurs as an aggregate class of Source, it is understood that the service is one from which activity of interest is originating; and that the service is "attached" to the Node, Process, and User information also contained in Source. Likewise, when Service occurs as an aggregate class of Target, it is understood that the service is one to which activity of interest is being directed; and that the service is "attached" to the Node, Process, and User information also contained in Target.

The Service class is composed of four aggregate classes, as shown in

Figure 5.17.

```
              +--------------+
              |   Service    |
              +--------------+       0..1 +----------+
              | STRING ident |<>----------|   name   |
              |              |            +----------+
              |              |       0..1 +----------+
              |              |<>----------|   port   |
              |              |            +----------+
              |              |       0..1 +----------+
              |              |<>----------| portlist |
              |              |            +----------+
              |              |       0..1 +----------+
              |              |<>----------| protocol |
              |              |            +----------+
              +--------------+
                    /_\
                     |
                     +------------+
                                  |
                 +-------------+  |  +-------------+
                 | SNMPService |--+--| WebService  |
                 +-------------+     +-------------+
```

Figure 5.17 - The Service Class

The aggregate classes that make up Service are:

name
: Zero or one. STRING. The name of the service. Whenever possible, the name from the IANA list of well-known ports SHOULD be used.

port
: Zero or one. INTEGER. The port number being used.

portlist
: Zero or one. PORTLIST. A list of port numbers being used; see Section 4.4.8 for formatting rules.

protocol
: Zero or one. STRING. The protocol being used.

A Service MUST be specified as either (a) a name, (b) a port, (c) a name and a port, or (d) a portlist. The protocol is optional in all cases, but no other combinations are permitted.

Because DTDs do not support subclassing (see Section 4.3.4), the inheritance relationship between Service and the SNMPService and WebService subclasses shown in Figure 5.17 has been replaced with an aggregate relationship.

Service is represented in the XML DTD as follows:

```
<!ELEMENT Service                          (
   ((name | port | (name, port)) | portlist), protocol?,
   SNMPService?, WebService?
  )>
<!ATTLIST Service
   ident                ID                      #IMPLIED
  >
```

The Service class has one attribute:

ident
: Optional. A unique identifier for the service, see Section 4.4.9.

5.2.6.4.1 The WebService Class

The WebService class carries additional information related to web traffic.

The WebService class is composed of four aggregate classes, as shown in Figure 5.18.

```
                 +-------------+
                 |   Service   |
                 +-------------+
                       /_\
                        |
                 +-------------+
                 | WebService  |
                 +-------------+            +--------+
                 |             |<>----------|  url   |
                 |             |            +--------+
                 |             |       0..1 +--------+
                 |             |<>----------|  cgi   |
                 |             |            +--------+
                 |             |       0..1 +--------+
                 |             |<>----------| method |
                 |             |            +--------+
                 |             |       0..* +--------+
                 |             |<>----------|  arg   |
                 |             |            +--------+
                 +-------------+
```

Figure 5.18 - The WebService Class

The aggregate classes that make up WebService are:

url
: Exactly one. STRING. The URL in the request.

cgi
: Zero or one. STRING. The CGI script in the request, without arguments.

method
: Zero or one. STRING. The HTTP method (PUT, GET) used in the request.

arg
: Zero or more. STRING. The arguments to the CGI script.

This is represented in the XML DTD as follows:

```
<!ELEMENT WebService                       (
   url, cgi?, method?, arg*
  )>
```

5.2.6.4.2 The SNMPService Class

The SNMPService class carries additional information related to SNMP traffic.

The SNMPService class is composed of three aggregate classes, as shown in Figure 5.19.

```
              +-------------+
              |   Service   |
              +-------------+
                    /_\
                     |
              +-------------+
              | SNMPService |
              +-------------+       0..1 +-----------+
              |             |<>----------|    oid    |
              |             |            +-----------+
              |             |       0..1 +-----------+
              |             |<>----------| community |
              |             |            +-----------+
              |             |       0..1 +-----------+
              |             |<>----------|  command  |
              |             |            +-----------+
              +-------------+
```

Figure 5.19 - The SNMPService Class

The aggregate classes that make up SNMPService are:

oid
: Zero or one. STRING. The object identifier in the request.

community
Zero or one. STRING. The object's community string.

command
Zero or one. STRING. The command sent to the SNMP server (GET, SET. etc.).

This is represented in the XML DTD as follows:

```
<!ELEMENT SNMPService                    (
    oid?, community?, command?
  )>
```

6. Extending the IDMEF

As intrusion detection systems evolve, the IDMEF data model and DTD will have to evolve along with them. To allow new features to be added as they are developed, both the data model and the DTD can be extended as described in this section. As these extensions mature, they can then be incorporated into future versions of the specification.

6.1 Extending the Data Model

There are two mechanisms for extending the IDMEF data model, inheritance and aggregation:

+ Inheritance denotes a superclass/subclass type of relationship where the subclass inherits all the attributes, operations, and relationships of the superclass. This type of relationship is also called a "is-a" or "kind-of" relationship. Subclasses may have additional attributes or operations that apply only to the subclass, and not to the superclass.

+ Aggregation is a form of association in which the whole is related to its parts. This type of relationship is also referred to as a "part-of" relationship. In this case, the aggregate class contains all of its own attributes and as many of the attributes associated with its parts as required and specified by occurrence indicators.

Of the two mechanisms, inheritance is preferred, because it preserves the existing data model structure and also preserves the operations (methods) executed on the classes of the structure.

Note that the rules for extending the XML DTD (see below) set limits on the places where extensions to the data model may be made.

6.2 Extending the XML DTD

There are two ways to extend the IDMEF XML DTD:

1. The AdditionalData class (see Section 5.2.4.5) allows implementors to include arbitrary "atomic" data items (integers, strings, etc.) in an Alert or Heartbeat message. This approach SHOULD be used whenever possible. See Sections 8.4 and 8.6.

2. The AdditionalData class allows implementors to extend the XML DTD with additional DTD "modules" that describe arbitrarily complex data types and relationships. The remainder of this section describes this extension method.

To extend the IDMEF DTD with a new DTD "module," the following steps MUST be followed:

1. The IDMEF message MUST include a document type declaration (see Section 4.3.1.3).

2. The document type declaration MUST define a parameter entity (see Section 4.2.4) that contains the location of the extension DTD, and then reference that entity:

```
<!DOCTYPE IDMEF-Message SYSTEM "/path/to/idmef-message.dtd" [
  <!ENTITY % x-extension SYSTEM "/path/to/extension.dtd">
  %x-extension;
 ]>
```

   In this example, the "x-extension" parameter entity is defined and then referenced, causing the DTD for the extension to be read by the XML parser.

   The name of the parameter entity defined for this purpose MUST be a string beginning with "x-"; there are no other restrictions on the name (other than those imposed on all entity names by XML).

   Multiple extensions may be included by defining multiple entities and referencing them. For example:

```
<!DOCTYPE IDMEF-Message SYSTEM "/path/to/idmef-message.dtd" [
  <!ENTITY % x-extension SYSTEM "/path/to/extension.dtd">
  <!ENTITY % x-another SYSTEM "/path/to/another.dtd">
  %x-extension;
  %x-another;
 ]>
```

3. Extension DTDs MUST declare all of their elements and attributes in a separate XML namespace. Extension DTDs MUST NOT declare any elements or attributes in the "idmef" or default namespaces.

   For example, the "test" extension might be declared as follows:


```
<!ELEMENT test:test (
    test:a, test:b, test:c
  )>
<!ATTLIST test:test
    xmlns      CDATA   #IMPLIED
    xmlns:test CDATA   #IMPLIED
  >

<!ELEMENT test:a (#PCDATA)>
<!ATTLIST test:a
    test:attr   CDATA   #IMPLIED
  >

<!ELEMENT test:b (#PCDATA)>
<!ELEMENT test:c (#PCDATA)>
```

4. Extensions MUST only be included in IDMEF alert and heartbeat messages under an <AdditionalData> element whose "type" attribute contains the value "xml". For example:

```
<IDMEF-Message version="0.3">
  <Alert ident="...">
    ...
    <AdditionalData type="xml">
      <test:test xmlns:test="http://www.ietf.org/test.html"
                 xmlns="http://www.ietf.org/test.html">
        <test:a test:attr="...">...</test:a>
        <test:b>...</test:b>
        <test:c>...</test:c>
      </test:test>
    </AdditionalData>
  </Alert>
</IDMEF-Message>
```

See Section 8.7 for another example of extending the IDMEF DTD with XML.

7. Special Considerations

This section discusses some of the special considerations that must be taken into account by implementors of the IDMEF.

7.1 XML Validity and Well-Formedness

It is expected that IDMEF-compliant applications will not normally include the IDMEF DTD itself in their communications. Instead, the DTD will be referenced in the document type declaration in the IDMEF message (see Section 4.3.1.3). Such IDMEF documents will be well-formed and valid as defined in [5].

Other IDMEF documents will be specified that do not include the document prolog (e.g., entries in an IDMEF-format database). Such IDMEF documents will be well-formed but not valid.

Generally, well-formedness implies that a document has a single element that contains everything else (e.g., "<Book>"), and that all the other elements nest nicely within each other without any overlapping (e.g., a "chapter" does not start in the middle of another "chapter").

Validity further implies that not only is the document well-formed, but it also follows specific rules (contained in the Document Type Definition) about which elements are "legal" in the document, how those elements nest within other elements, and so on (e.g., a "chapter" does not begin in the middle of a "title"). A document cannot be valid unless it references a DTD.

XML processors are required to be able to parse any well-formed document, valid or not. The purpose of validation is to make the processing of that document (what's done with the data after it's parsed) easier. Without validation, a document may contain elements in nonsense order, elements "invented" by the author that the processing application doesn't understand, and so forth.

IDMEF documents MUST be well-formed. IDMEF documents SHOULD be valid whenever both possible and practical.

7.2 Unrecognized XML Tags

On occasion, an IDMEF-compliant application may receive a well-formed, or even well-formed and valid, IDMEF message containing tags that it does not understand. The tags may be either:

+ Recognized as "legitimate" (a valid document), but the application does not know the semantic meaning of the element's content; or

+ Not recognized at all.

IDMEF-compliant applications MUST continue to process IDMEF messages that contain unknown tags, provided that such messages meet the well-formedness requirement of Section 7.1. It is up to the individual application to decide how to process (or ignore) any content from the unknown elements(s).

7.3 Analyzer-Manager Time Synchronization

Synchronization of time-of-day clocks between analyzers and managers is outside the scope of this document. However, the following comments and suggestions are offered:

1. Whenever possible, all analyzers and managers should have their time-of-day clocks synchronized to an external source such as NTP or SNTP [13, 14], GPS/GOES/WWV clocks, or some other reliable time standard.

2. When external time synchronization is not possible, the IDMEF provides the <AnalyzerTime> element, which may be used to perform rudimentary time synchronization (see below).

3. IDMEF-compliant applications SHOULD permit the user to enable/disable the <AnalyzerTime> method of time synchronization as a configuration option.

A number of caveats apply to the use of <AnalyzerTime> for time synchronization:

1. <AnalyzerTime> works best in a "flat" environment where analyzers report up to a single level of managers. When a tree topology of high-level managers, intermediate relays, and analyzers is used, the problem becomes more complex.

2. When intermediate message relays (managers or otherwise) are involved, two scenarios are possible:

   a. The intermediaries may forward entire IDMEF messages, or may perform aggregation or correlation, but MUST NOT inject delay. In this case, time synchronization is end-to-end between the analyzer and the highest-level manager.

   b. The intermediaries may inject delay, due to storage or additional processing. In this case, time synchronization MUST be performed at each hop. This means each intermediary must decompose the IDMEF message, adjust all time values, and then reconstruct the message before sending it on.

3. When the environment is mixed, with some analyzers and managers using external time synchronization and some not, all managers and intermediaries must perform <AnalyzerTime> synchronization. This is because determining whether or not compensation is actually needed between two parties rapidly becomes very complex, and requires knowledge of other parts of the topology.

4. If an alert can take alternate paths, or be stored in multiple locations, the recorded times may be different depending on the path taken.

The above being said, <AnalyzerTime> synchronization is probably still better than nothing in many environments. To implement this type of synchronization, the following procedure is suggested:

1. When an analyzer or manager sends an IDMEF message, it should place the current value of its time-of-day clock in an

<AnalyzerTime> element. This should occur as late as possible in the message transmission process, ideally right before the message is "put on the wire."

2. When a manager receives an IDMEF message, it should compute the difference between its own time-of-day clock and the time in the <AnalyzerTime> element of the message. This difference should then be used to adjust the times in the <CreateTime> and <DetectTime> elements (NTP timestamps should also be adjusted).

3. If the manager is an intermediary and sends the IDMEF message on to a higher-level manager, and hop-by-hop synchronization is in effect, it should regenerate the <AnalyzerTime> value to contain the value of its own time-of-day clock.

7.4 NTP Timestamp Wrap-Around

From [14]:

> Note that, since some time in 1968 (second 2,147,483,648) the most significant bit (bit 0 of the integer part) has been set and that the 64-bit field will overflow some time in 2036 (second 4,294,967,296). Should NTP or SNTP be in use in 2036, some external means will be necessary to qualify time relative to 1900 and time relative to 2036 (and other multiples of 136 years). There will exist a 200-picosecond interval, henceforth ignored, every 136 years when the 64-bit field will be 0, which by convention is interpreted as an invalid or unavailable timestamp.

IDMEF-compliant applications MUST NOT send a zero-valued NTP timestamp unless they mean to indicate that it is invalid or unavailable. If an IDMEF-compliant application must send an IDMEF message at the time of rollover, the application should wait for 200 picoseconds until the timestamp will have a non-zero value.

Also from [14]:

> As the NTP timestamp format has been in use for the last 17 years, it remains a possibility that it will be in use 40 years from now when the seconds field overflows. As it is probably inappropriate to archive NTP timestamps before bit 0 was set in 1968, a convenient way to extend the useful life of NTP timestamps is the following convention:
>
> > If bit 0 is set, the UTC time is in the range 1968-2036 and UTC time is reckoned from 0h 0m 0s UTC on 1 January 1900.
> >
> > If bit 0 is not set, the time is in the range 2036-2104 and UTC time is reckoned from 6h 28m 16s UTC on 7 February 2036.
>
> Note that when calculating the correspondence, 2000 is not a leap

> year. Note also that leap seconds are not counted in the reckoning.

IDMEF-compliant applications in use after 2036-02-07T06:28:16Z MUST adhere to the above convention.

7.5 Digital Signatures

The joint IETF/W3C XML Signature Working Group is currently working to specify XML digital signature processing rules and syntax [15]. XML Signatures provide integrity, message authentication, and/or signer authentication services for data of any type, whether located within the XML that includes the signature or elsewhere.

The IDMEF requirements document assigns responsibility for message integrity and authentication to the communications protocol, not the message format. However, in situations where IDMEF messages are exchanged over other, less secure protocols, or in cases where the digital signatures must be archived for later use, the inclusion of digital signatures within an IDMEF message itself may be desirable.

Specifications for the use of digital signatures within IDMEF messages are outside the scope of this document. However, if such functionality is needed, use of the XML Signature standard is RECOMMENDED.

8. Examples

The examples shown in this section demonstrate how the IDMEF is used to encode alert data. These examples are for illustrative purposes only, and do not necessarily represent the only (or even the "best" way to encode these particular alerts). These examples should not be taken as guidelines on how alerts should be classified.

8.1 Denial of Service Attacks

The following examples show how some common denial of service attacks could be represented in the IDMEF.

8.1.1 The "teardrop" Attack

Network-based detection of the "teardrop" attack. This shows the basic format of an alert.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE IDMEF-Message PUBLIC "-//IETF//DTD RFCxxxx IDMEF v0.3//EN"
  "idmef-message.dtd">
```



```
<IDMEF-Message version="0.3">
  <Alert ident="abc123456789" impact="successful-dos">
    <Analyzer analyzerid="hq-dmz-analyzer01">
      <Node category="dns">
        <location>Headquarters DMZ Network</location>
        <name>analyzer01.bigcompany.com</name>
      </Node>
    </Analyzer>
    <CreateTime ntpstamp="0x12345678.0x98765432">
      2000-03-09T10:01:25.93464-05:00
    </CreateTime>
    <Source ident="a1b2c3d4">
      <Node ident="a1b2c3d4-001" category="dns">
        <name>badguy.hacker.net</name>
        <Address ident="a1b2c3d4-002" category="ipv4-net-mask">
          <address>123.234.231.121</address>
          <netmask>255.255.255.255</netmask>
        </Address>
      </Node>
    </Source>
    <Target ident="d1c2b3a4">
      <Node ident="d1c2b3a4-001" category="dns">
        <Address category="ipv4-addr-hex">
          <address>0xde796f70</address>
        </Address>
      </Node>
    </Target>
    <Classification origin="bugtraqid">
      <name>124</name>
      <url>http://www.securityfocus.com</url>
    </Classification>
  </Alert>
</IDMEF-Message>
```

8.1.2 The "ping of death" Attack

Network-based detection of the "ping of death" attack. Note the identification of multiple targets, and the identification of the source as a spoofed address.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE IDMEF-Message PUBLIC "-//IETF//DTD RFCxxxx IDMEF v0.3//EN"
  "idmef-message.dtd">

<IDMEF-Message version="0.3">
  <Alert ident="abc123456789" impact="attempted-dos">
    <Analyzer analyzerid="bc-sensor01">
      <Node category="dns">
        <name>sensor.bigcompany.com</name>
      </Node>
```

```
      </Analyzer>
      <CreateTime ntpstamp="0x12345678.0x98765432">
        2000-03-09T10:01:25.93464Z
      </CreateTime>
      <Source ident="a1a2" spoofed="yes">
        <Node ident="a1a2-1">
          <Address ident="a1a2-2" category="ipv4-addr">
            <address>222.121.111.112</address>
          </Address>
        </Node>
      </Source>
      <Target ident="b3b4">
        <Node>
          <Address ident="b3b4-1" category="ipv4-addr">
            <address>123.234.231.121</address>
          </Address>
        </Node>
      </Target>
      <Target ident="c5c6">
        <Node ident="c5c6-1" category="nisplus">
          <name>lollipop</name>
        </Node>
      </Target>
      <Target ident="d7d8">
        <Node ident="d7d8-1">
          <location>Cabinet B10</location>
          <name>Cisco.router.b10</name>
        </Node>
      </Target>
      <Classification origin="cve">
        <name>CVE-1999-128</name>
        <url>http://www.cve.mitre.org/</url>
      </Classification>
    </Alert>
  </IDMEF-Message>
```

8.2 Port Scanning Attacks

The following examples show how some common port scanning attacks could be represented in the IDMEF.

8.2.1 Connection To a Disallowed Service

Host-based detection of a policy violation (attempt to obtain information via "finger"). Note the identification of the target service, as well as the originating user (obtained, e.g., through RFC1413).

```
<?xml version="1.0" encoding="UTF-8"?>
```



```
  <!DOCTYPE IDMEF-Message PUBLIC "-//IETF//DTD RFCxxxx IDMEF v0.3//EN"
    "idmef-message.dtd">

  <IDMEF-Message version="0.3">
    <Alert ident="abc123456789" impact="attempted-recon">
      <Analyzer analyzerid="bc-sensor01">
        <Node category="dns">
          <name>sensor.bigcompany.com</name>
        </Node>
      </Analyzer>
      <CreateTime ntpstamp="0x12345678.0x98765432">
        2000-03-09T18:47:25+02:00
      </CreateTime>
      <Source ident="a123">
        <Node ident="a123-01">
          <Address ident="a123-02" category="ipv4-addr">
            <address>222.121.111.112</address>
          </Address>
        </Node>
        <User ident="q987-03" category="os-device">
          <UserId ident="q987-04" type="target-user">
            <name>badguy</name>
          </UserId>
        </User>
        <Service ident="a123-03">
          <port>31532</port>
        </Service>
      </Source>
      <Target ident="z456">
        <Node ident="z456-01" category="nis">
          <name>myhost</name>
          <Address ident="z456-02" category="ipv4-addr">
            <address>123.234.231.121</address>
          </Address>
        </Node>
        <Service ident="z456-03">
          <name>finger</name>
          <port>79</port>
        </Service>
      </Target>
      <Classification origin="vendor-specific">
        <name>finger</name>
        <url>http://www.vendor.com/finger</url>
      </Classification>
    </Alert>
  </IDMEF-Message>
```

8.2.2 Simple Port Scanning

Network-based detection of a port scan. This shows detection by a single analyzer; see Example 8.5 for the same attack as detected by a

correlation engine. Note the use of <portlist> to show the ports that were scanned.

```
  <?xml version="1.0" encoding="UTF-8"?>

  <!DOCTYPE IDMEF-Message PUBLIC "-//IETF//DTD RFCxxxx IDMEF v0.3//EN"
    "idmef-message.dtd">

  <IDMEF-Message version="0.3">
    <Alert ident="abc123456789" impact="successful-recon-limited">
      <Analyzer analyzerid="hq-dmz-analyzer62">
        <Node category="dns">
          <location>Headquarters Web Server</location>
          <name>analyzer62.bigcompany.com</name>
        </Node>
      </Analyzer>
      <CreateTime ntpstamp="0x12345678.0x98765432">
        2000-03-09T15:31:00-08:00
      </CreateTime>
      <Source ident="abc01">
        <Node ident="abc01-01">
          <Address ident="abc01-02" category="ipv4-addr">
            <address>222.121.111.112</address>
          </Address>
        </Node>
      </Source>
      <Target ident="def01">
        <Node ident="def01-01" category="dns">
          <name>www.bigcompany.com</name>
          <Address ident="def01-02" category="ipv4-addr">
            <address>123.234.231.121</address>
          </Address>
        </Node>
        <Service ident="def01-03">
          <portlist>5-25,37,42,43,53,69-119,123-514</portlist>
        </Service>
      </Target>
      <Classification origin="vendor-specific">
        <name>portscan</name>
        <url>http://www.vendor.com/portscan</url>
      </Classification>
    </Alert>
  </IDMEF-Message>
```

8.3 Local Attacks

The following examples show how some common local host attacks could be represented in the IDMEF.

8.3.1 The "loadmodule" Attack

Host-based detection of the "loadmodule" exploit. This attack involves tricking the "loadmodule" program into running another program; since "loadmodule" is set-user-id "root," the executed program runs with super-user privileges. Note the use of <User> and <Process> to identify the user attempting the exploit and how he's doing it.

```
  <?xml version="1.0" encoding="UTF-8"?>

  <!DOCTYPE IDMEF-Message PUBLIC "-//IETF//DTD RFCxxxx IDMEF v0.3//EN"
    "idmef-message.dtd">

  <IDMEF-Message version="0.3">
    <Alert ident="abc123456789" impact="attempted-admin">
      <Analyzer analyzerid="bc-fs-sensor13">
        <Node category="dns">
          <name>fileserver.bigcompany.com</name>
        </Node>
        <Process>
          <name>monitor</name>
          <pid>8956</pid>
          <arg>monitor</arg><arg>-d</arg>
          <arg>-m</arg><arg>idmanager.bigcompany.com</arg>
          <arg>-l</arg><arg>/var/logs/idlog</arg>
        </Process>
      </Analyzer>
      <CreateTime ntpstamp="0x12345678.0x98765432">
        2000-03-09T08:12:32.3-05:00
      </CreateTime>
      <Source ident="a1a2">
        <User ident="a1a2-01" category="os-device">
          <UserId ident="a1a2-02" type="original-user">
            <name>joe</name>
            <number>13243</number>
          </UserId>
        </User>
        <Process ident="a1a2-03">
          <name>loadmodule</name>
          <path>/usr/openwin/bin</path>
        </Process>
      </Source>
      <Target ident="z3z4">
        <Node ident="z3z4-01" category="dns">
          <name>fileserver.bigcompany.com</name>
        </Node>
      </Target>
      <Classification origin="bugtraqid">
        <name>33</name>
        <url>http://www.securityfocus.com</url>
      </Classification>
```

```
  </Alert>
</IDMEF-Message>
```

The IDS could also indicate that the target user is the "root" user, and show the attempted command; the alert might then look like:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE IDMEF-Message PUBLIC "-//IETF//DTD RFCxxxx IDMEF v0.3//EN"
  "idmef-message.dtd">

<IDMEF-Message version="0.3">
  <Alert ident="abc123456789" impact="attempted-admin">
    <Analyzer analyzerid="bc-fs-sensor13">
      <Node category="dns">
        <name>fileserver.bigcompany.com</name>
      </Node>
      <Process>
        <name>monitor</name>
        <pid>8956</pid>
        <arg>monitor</arg><arg>-d</arg>
        <arg>-m</arg><arg>idmanager.bigcompany.com</arg>
        <arg>-l</arg><arg>/var/logs/idlog</arg>
      </Process>
    </Analyzer>
    <CreateTime ntpstamp="0x12345678.0x98765432">
      2000-03-09T08:12:32.3-05:00
    </CreateTime>
    <Source ident="a1a2">
      <User ident="a1a2-01" category="os-device">
        <UserId ident="a1a2-02" type="original-user">
          <name>joe</name>
          <number>13243</number>
        </UserId>
      </User>
      <Process ident="a1a2-03">
        <name>loadmodule</name>
        <path>/usr/openwin/bin</path>
      </Process>
    </Source>
    <Target ident="z3z4">
      <Node ident="z3z4-01" category="dns">
        <name>fileserver.bigcompany.com</name>
      </Node>
      <User ident="z3z4-02" category="os-device">
        <UserId ident="z3z4-03" type="target-user">
          <name>root</name>
          <number>0</number>
        </UserId>
      </User>
      <Process ident="z3z4-04">
        <name>sh</name>
```



```
        <pid>25134</pid>
        <path>/bin/sh</path>
      </Process>
    </Target>
    <Classification origin="bugtraqid">
      <name>33</name>
      <url>http://www.securityfocus.com</url>
    </Classification>
  </Alert>
</IDMEF-Message>
```

8.3.2 The "phf" Attack

Network-based detection of the "phf" attack. Note the use of the <WebService> element to provide more details about this particular attack.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE IDMEF-Message PUBLIC "-//IETF//DTD RFCxxxx IDMEF v0.3//EN"
  "idmef-message.dtd">

<IDMEF-Message version="0.3">
  <Alert ident="abc123456789" impact="attempted-recon">
    <Analyzer analyzerid="bc-sensor01">
      <Node category="dns">
        <name>sensor.bigcompany.com</name>
      </Node>
    </Analyzer>
    <CreateTime ntpstamp="0x12345678.0x98765432">
      2000-03-09T08:12:32-01:00
    </CreateTime>
    <Source ident="abc123">
      <Node ident="abc123-001">
        <Address ident="abc123-002" category="ipv4-addr">
          <address>222.121.111.112</address>
        </Address>
      </Node>
      <Service ident="abc123-003">
        <port>21534</port>
      </Service>
    </Source>
    <Target ident="xyz789">
      <Node ident="xyz789-001" category="dns">
        <name>www.bigcompany.com</name>
        <Address ident="xyz789-002" category="ipv4-addr">
          <address>123.45.67.89</address>
        </Address>
      </Node>
      <Service>
        <port>8080</port>
```



```
        <WebService>
          <url>
            http://www.bigcompany.com/cgi-bin/phf?/etc/group
          </url>
          <cgi>/cgi-bin/phf</cgi>
          <method>GET</method>
        </WebService>
      </Service>
    </Target>
    <Classification origin="bugtraqid">
      <name>629</name>
      <url>http://www.securityfocus.com</url>
    </Classification>
  </Alert>
</IDMEF-Message>
```

8.4 System Policy Violation

In this example, logins are restricted to daytime hours. The alert reports a violation of this policy that occurs when a user logs in a little after 10:00pm. Note the use of <AdditionalData> to provide information about the policy being violated.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE IDMEF-Message PUBLIC "-//IETF//DTD RFCxxxx IDMEF v0.3//EN"
  "idmef-message.dtd">

<IDMEF-Message version="0.3">
  <Alert ident="abc123456789" impact="attempted-user">
    <Analyzer analyzerid="bc-ds-01">
      <Node category="dns">
        <name>dialserver.bigcompany.com</name>
      </Node>
    </Analyzer>
    <CreateTime ntpstamp="0x12345678.0x98765432">
      2000-03-09T22:18:07-05:00
    </CreateTime>
    <Source ident="s01">
      <Node ident="s01-1">
        <Address category="ipv4-addr">
          <address>127.0.0.1</address>
        </Address>
      </Node>
      <Service ident="s01-2">
        <port>4325</port>
      </Service>
    </Source>
    <Target ident="t01">
      <Node ident="t01-1" category="dns">
        <name>mainframe.bigcompany.com</name>
```



```
      </Node>
      <User ident="t01-2" category="os-device">
        <UserId ident="t01-3" type="current-user">
          <name>louis</name>
          <number>501</number>
        </UserId>
      </User>
      <Service ident="t01-4">
        <name>login</name>
        <port>23</port>
      </Service>
    </Target>
    <Classification origin="vendor-specific">
      <name>out-of-hours activity</name>
      <url>http://my.company.com/policies</url>
    </Classification>
    <AdditionalData type="date-time" meaning="start-time">
      2000-03-09T07:00:00-05:00
    </AdditionalData>
    <AdditionalData type="date-time" meaning="stop-time">
      2000-03-09T19:30:00-05:00
    </AdditionalData>
  </Alert>
</IDMEF-Message>
```

8.5 Correlated Alerts

The following example shows how the port scan alert from Section 8.2.2 could be represented if it had been detected and sent from a correlation engine, instead of a single analyzer.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE IDMEF-Message PUBLIC "-//IETF//DTD RFCxxxx IDMEF v0.3//EN"
  "idmef-message.dtd">

<IDMEF-Message version="0.3">
  <Alert ident="abc123456789" impact="successful-recon-largescale">
    <Analyzer analyzerid="bc-corr-01">
      <Node category="dns">
        <name>correlator01.bigcompany.com</name>
      </Node>
    </Analyzer>
    <CreateTime ntpstamp="0x12345678.0x98765432">
      2000-03-09T15:31:07Z
    </CreateTime>
    <Source ident="a1">
      <Node ident="a1-1">
        <Address ident="a1-2" category="ipv4-addr">
          <address>222.121.111.112</address>
        </Address>
```

```
        </Node>
      </Source>
      <Target ident="a2">
        <Node ident="a2-1" category="dns">
          <name>www.bigcompany.com</name>
          <Address ident="a2-2" category="ipv4-addr">
            <address>123.234.231.121</address>
          </Address>
        </Node>
        <Service ident="a2-3">
          <portlist>5-25,37,42,43,53,69-119,123-514</portlist>
        </Service>
      </Target>
      <Classification origin="vendor-specific">
        <name>portscan</name>
        <url>http://www.vendor.com/portscan</url>
      </Classification>
      <CorrelationAlert>
        <name>multiple ports in short time</name>
        <alertident>123456781</alertident>
        <alertident>123456782</alertident>
        <alertident>123456783</alertident>
        <alertident>123456784</alertident>
        <alertident>123456785</alertident>
        <alertident>123456786</alertident>
        <alertident analyzerid="a1b2c3d4">987654321</alertident>
        <alertident analyzerid="a1b2c3d4">987654322</alertident>
      </CorrelationAlert>
    </Alert>
  </IDMEF-Message>
```

8.6 Heartbeat

This example shows a heartbeat message that provides "I'm alive and working" information to the manager. Note the use of <AdditionalData> elements, with "meaning" attributes, to provide some additional information.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE IDMEF-Message PUBLIC "-//IETF//DTD RFCxxxx IDMEF v0.3//EN"
  "idmef-message.dtd">

<IDMEF-Message version="0.3">
  <Heartbeat ident="abc123456789">
    <Analyzer analyzerid="hq-dmz-analyzer01">
      <Node category="dns">
        <location>Headquarters DMZ Network</location>
        <name>analyzer01.bigcompany.com</name>
      </Node>
    </Analyzer>
```



```
    <CreateTime ntpstamp="0x12345678.0x98765432">
      2000-03-09T14:07:58Z
    </CreateTime>
    <AdditionalData type="real" meaning="%memused">
      62.5
    </AdditionalData>
    <AdditionalData type="real" meaning="%diskused">
      87.1
    </AdditionalData>
  </Heartbeat>
</IDMEF-Message>
```

8.7 XML Extension

The following example shows how to extend the IDMEF DTD with XML. In the example, the VendorCo company has decided it wants to add geographic information to the Node class. To do this, VendorCo creates a Document Type Definition that defines how their class will be formatted:

```
<!ELEMENT VendorCo:NodeGeography (
    VendorCo:latitude, VendorCo:longitude, VendorCo:elevation?
  )>
<!ATTLIST VendorCo:NodeGeography
    xmlns                 CDATA                   #IMPLIED
    xmlns:VendorCo        CDATA                   #IMPLIED
    VendorCo:node-ident CDATA                     #REQUIRED
  >

<!ELEMENT VendorCo:latitude     (#PCDATA) >
<!ELEMENT VendorCo:longitude    (#PCDATA) >
<!ELEMENT VendorCo:elevation    (#PCDATA) >
```

The VendorCo:NodeGeography class will contain the geographic data in three aggregate classes, VendorCo:latitude, VendorCo:longitude, and VendorCo:elevation. To associate the information in this class with a particular node, the VendorCo:node-ident attribute is provided; it must contain the same value as the ident attribute on the relevant Node element.

To make use of this DTD now, VendorCo follows the rules in Section 6.2 and defines a parameter entity called "x-vendorco" within the Document Type Declaration, and then references this entity. In the alert, the DTD's elements are included under the AdditionalData element, with a type attribute of "xml", as shown below.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE IDMEF-Message PUBLIC "-//IETF//DTD RFCxxxx IDMEF v0.3//EN"
  "idmef-message.dtd" [
    <!ENTITY % x-vendorco SYSTEM "vendorco.dtd">
```



```
    %x-vendorco;
  ]>

<IDMEF-Message version="0.3">
  <Alert ident="abc123456789" impact="successful-dos">
    <Analyzer analyzerid="hq-dmz-analyzer01">
      <Node category="dns">
        <location>Headquarters DMZ Network</location>
        <name>analyzer01.bigcompany.com</name>
      </Node>
    </Analyzer>
    <CreateTime ntpstamp="0x12345678.0x98765432">
      2000-03-09T10:01:25.93464-05:00
    </CreateTime>
    <Source ident="a1b2c3d4">
      <Node ident="a1b2c3d4-001" category="dns">
        <name>badguy.hacker.net</name>
        <Address ident="a1b2c3d4-002" category="ipv4-net-mask">
          <address>123.234.231.121</address>
          <netmask>255.255.255.255</netmask>
        </Address>
      </Node>
    </Source>
    <Target ident="d1c2b3a4">
      <Node ident="d1c2b3a4-001" category="dns">
        <Address category="ipv4-addr-hex">
          <address>0xde796f70</address>
        </Address>
      </Node>
    </Target>
    <Classification origin="bugtraqid">
      <name>124</name>
      <url>http://www.securityfocus.com</url>
    </Classification>
    <AdditionalData type="xml">
      <VendorCo:NodeGeography VendorCo:node-ident="a1b2c3d4-001">
        <VendorCo:latitude>38.89</VendorCo:latitude>
        <VendorCo:longitude>-77.02</VendorCo:longitude>
      </VendorCo:NodeGeography>
    </AdditionalData>
  </Alert>
</IDMEF-Message>
```

9. The IDMEF Document Type Definition

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- ***************************************************************
 *****************************************************************
 *** Intrusion Detection Message Exchange Format (IDMEF) XML DTD ***
 ***                Version 0.3, 14 February 2001                ***
```



```
 ***                                                             ***
 *** The use and extension of the IDMEF XML DTD are described in ***
 *** RFC XXXX, "Intrusion Detection Message Exchange Format Data ***
 *** Model and Extensible Markup Language (XML) Document Type    ***
 *** Definition," D. Curry and H. Debar.                         ***
 *******************************************************************
 *************************************************************** -->

<!-- =============================================================
 ===================================================================
 === SECTION 1. Attribute list declarations.
 ===================================================================
 ============================================================== -->

<!--
 | Attributes of the IDMEF element.  In general, the fixed values of
 | these attributes will change each time a new version of the DTD
 | is released.
 -->
<!ENTITY % attlist.idmef            "
    version              CDATA                    #FIXED    '0.3'
  ">

<!--
 | Attributes of all elements.  These are the "XML" attributes that
 | every element should have.  Space handling, language, and name
 | space.
 -->
<!ENTITY % attlist.global           "
    xmlns:idmef          CDATA                    #FIXED
        'urn:iana:xml:ns:idmef'
    xmlns                CDATA                    #FIXED
        'urn:iana:xml:ns:idmef'
    xml:space            (default | preserve)     'default'
    xml:lang             NMTOKEN                  #IMPLIED
  ">

<!-- =============================================================
 ===================================================================
 === SECTION 2. Attribute value declarations.  Enumerated values for
 ===            many of the element-specific attribute lists.
 ===================================================================
 ============================================================== -->

<!--
 | Values for the Address.category attribute.
 -->
<!ENTITY % attvals.addrcat          "
    ( unknown | atm | e-mail | lotus-notes | mac | sna | vm |
      ipv4-addr | ipv4-addr-hex | ipv4-net | ipv4-net-mask |
      ipv6-addr | ipv6-addr-hex | ipv6-net | ipv6-net-mask )
  ">
```

```
<!--
 | Values for the AdditionalData.type attribute.
 -->
<!ENTITY % attvals.adtype            "
    ( boolean | byte | character | date-time | integer | ntpstamp |
      portlist | real | string | xml )
  ">

<!--
 | Values for the Alert.impact attribute.
 -->
<!ENTITY % attvals.impact            "
    ( unknown | bad-unknown | not-suspicious | attempted-admin |
      successful-admin | attempted-dos | successful-dos |
      attempted-recon | successful-recon-limited |
      successful-recon-largescale | attempted-user |
      successful-user )
  ">

<!--
 | Values for the Node.category attribute.
 -->
<!ENTITY % attvals.nodecat           "
    ( unknown | ads | afs | coda | dfs | dns | kerberos | nds |
      nis | nisplus | nt | wfw )
  ">

<!--
 | Values for the Classification.origin attribute.
 -->
<!ENTITY % attvals.origin            "
    ( unknown | bugtraqid | cve | vendor-specific )
  ">

<!--
 | Values for the Id.type attribute.
 -->
<!ENTITY % attvals.idtype            "
    ( current-user | original-user | target-user | user-privs |
      current-group | group-privs )
  ">

<!--
 | Values for the User.category attribute.
 -->
<!ENTITY % attvals.usercat           "
    ( unknown | application | os-device )
  ">

<!--
 | Values for yes/no attributes such as Source.spoofed and
 | Target.decoy.
```



```
 -->
<!ENTITY % attvals.yesno                "
    ( unknown | yes | no )
  ">

<!-- ===============================================================
 ===============================================================
 === SECTION 3. Top-level element declarations.  The IDMEF-Message
 ===            element and the types of messages it can include.
 ===============================================================
 =============================================================== -->

<!ELEMENT IDMEF-Message                 (
    (Alert | Heartbeat)*
  )>
<!ATTLIST IDMEF-Message
    %attlist.global;
    %attlist.idmef;
  >

<!ELEMENT Alert                         (
    Analyzer, CreateTime, DetectTime?, AnalyzerTime?, Source*,
    Target*, Classification+, ToolAlert?, OverflowAlert?,
    CorrelationAlert?, AdditionalData*
  )>
<!ATTLIST Alert
    ident               CDATA                   '0'
    impact              %attvals.impact;        'unknown'
    %attlist.global;
  >

<!ELEMENT Heartbeat                     (
    Analyzer, CreateTime, AnalyzerTime?, AdditionalData*
  )>
<!ATTLIST Heartbeat
    ident               CDATA                   '0'
    %attlist.global;
  >

<!-- ===============================================================
 ===============================================================
 === SECTION 4. Subclasses of the Alert element that provide more
 ===            data for specific types of alerts.
 ===============================================================
 =============================================================== -->

<!ELEMENT CorrelationAlert              (
    name, alertident+
  )>
<!ATTLIST CorrelationAlert
    %attlist.global;
  >
```



```
<!ELEMENT OverflowAlert                 (
    program, size?, buffer?
  )>
<!ATTLIST OverflowAlert
    %attlist.global;
  >

<!ELEMENT ToolAlert                     (
    name, command?, alertident+
  )>
<!ATTLIST ToolAlert
    %attlist.global;
  >

<!-- ===============================================================
 ===============================================================
 === SECTION 5.  The AdditionalData element.  This element allows an
 ===             alert to include additional information that cannot
 ===             be encoded elsewhere in the data model.
 ===============================================================
 =============================================================== -->

<!ELEMENT AdditionalData            ANY >
<!ATTLIST AdditionalData
    type                %attvals.adtype;        'string'
    meaning             CDATA                   #IMPLIED
    %attlist.global;
  >

<!-- ===============================================================
 ===============================================================
 === SECTION 6. Elements related to identifying entities - analyzers
 ===            (the senders of these messages), sources (of
 ===            attacks), and targets (of attacks).
 ===============================================================
 =============================================================== -->

<!ELEMENT Analyzer                      (
    Node?, Process?
  )>
<!ATTLIST Analyzer
    analyzerid          CDATA                   '0'
    %attlist.global;
  >

<!ELEMENT Source                        (
    Node?, User?, Process?, Service?
  )>
<!ATTLIST Source
    ident               CDATA                   '0'
    spoofed             %attvals.yesno;         'unknown'
    interface           CDATA                   #IMPLIED
```



```
    %attlist.global;
  >

<!ELEMENT Target                        (
    Node?, User?, Process?, Service?
  )>
<!ATTLIST Target
    ident               CDATA                   '0'
    decoy               %attvals.yesno;         'unknown'
    interface           CDATA                   #IMPLIED
    %attlist.global;
  >

<!-- ===============================================================
 ===============================================================
 === SECTION 7. Support elements used for providing detailed info
 ===            about entities - addresses, names, etc.
 ===============================================================
 =============================================================== -->

<!ELEMENT Address                       (
    address, netmask?
  )>
<!ATTLIST Address
    ident               CDATA                   '0'
    category            %attvals.addrcat;       'unknown'
    vlan-name           CDATA                   #IMPLIED
    vlan-num            CDATA                   #IMPLIED
    %attlist.global;
  >

<!ELEMENT Classification                (
    name, url
  )>
<!ATTLIST Classification
    origin              %attvals.origin;        'unknown'
    %attlist.global;
  >

<!ELEMENT Node                          (
    location?, (name | Address), Address*
  )>
<!ATTLIST Node
    ident               CDATA                   '0'
    category            %attvals.nodecat;       'unknown'
    %attlist.global;
  >

<!ELEMENT Process                       (
    name, pid?, path?, arg*, env*
  )>
<!ATTLIST Process
```

```
      ident               CDATA                   '0'
      %attlist.global;
    >

  <!ELEMENT Service                           (
      ((name | port | (name, port)) | portlist), protocol?,
      SNMPService?, WebService?
    )>
  <!ATTLIST Service
      ident               CDATA                   '0'
      %attlist.global;
    >

  <!ELEMENT SNMPService                       (
      oid?, community?, command?
    )>
  <!ATTLIST SNMPService
      %attlist.global;
    >

  <!ELEMENT User                              (
      UserId+
    )>
  <!ATTLIST User
      ident               CDATA                   '0'
      category            %attvals.usercat;       'unknown'
      %attlist.global;
    >

  <!ELEMENT UserId                            (
      name | number | (name, number)
    )>
  <!ATTLIST UserId
      ident               CDATA                   '0'
      type                %attvals.idtype;        'original-user'
      %attlist.global;
    >

  <!ELEMENT WebService                        (
      url, cgi?, method?, arg*
    )>
  <!ATTLIST WebService
      %attlist.global;
    >

  <!-- ===============================================================
   ===============================================================
   === SECTION 8. Simple elements with sub-elements or attributes of a
   ===            special nature.
   ===============================================================
   =============================================================== -->
```



```
  <!ELEMENT AnalyzerTime        (#PCDATA) >
  <!ATTLIST AnalyzerTime
      ntpstamp            CDATA                   #REQUIRED
      %attlist.global;
    >

  <!ELEMENT CreateTime          (#PCDATA) >
  <!ATTLIST CreateTime
      ntpstamp            CDATA                   #REQUIRED
      %attlist.global;
    >

  <!ELEMENT DetectTime          (#PCDATA) >
  <!ATTLIST DetectTime
      ntpstamp            CDATA                   #REQUIRED
      %attlist.global;

    >

  <!ELEMENT alertident          (#PCDATA) >
  <!ATTLIST alertident
      analyzerid          CDATA                   #IMPLIED
      %attlist.global;
    >

  <!-- ===============================================================
   ===============================================================
   === SECTION 9. Simple elements with no sub-elements and no special
   ===            attributes.
   ===============================================================
   =============================================================== -->

  <!ELEMENT address             (#PCDATA) >
  <!ATTLIST address
      %attlist.global;
    >

  <!ELEMENT arg                 (#PCDATA) >
  <!ATTLIST arg
      %attlist.global;
    >

  <!ELEMENT buffer              (#PCDATA) >
  <!ATTLIST buffer
      %attlist.global;
    >

  <!ELEMENT cgi                 (#PCDATA) >
  <!ATTLIST cgi
      %attlist.global;
    >
```



```
  <!ELEMENT command             (#PCDATA) >
  <!ATTLIST command
      %attlist.global;
    >

  <!ELEMENT community           (#PCDATA) >
  <!ATTLIST community
      %attlist.global;
    >

  <!ELEMENT env                 (#PCDATA) >
  <!ATTLIST env
      %attlist.global;
    >

  <!ELEMENT location            (#PCDATA) >
  <!ATTLIST location
      %attlist.global;
    >

  <!ELEMENT method              (#PCDATA) >
  <!ATTLIST method
      %attlist.global;
    >

  <!ELEMENT name                (#PCDATA) >
  <!ATTLIST name
      %attlist.global;
    >

  <!ELEMENT netmask             (#PCDATA) >
  <!ATTLIST netmask
      %attlist.global;
    >

  <!ELEMENT number              (#PCDATA) >
  <!ATTLIST number
      %attlist.global;
    >

  <!ELEMENT oid                 (#PCDATA) >
  <!ATTLIST oid
      %attlist.global;
    >

  <!ELEMENT path                (#PCDATA) >
  <!ATTLIST path
      %attlist.global;
    >

  <!ELEMENT pid                 (#PCDATA) >
  <!ATTLIST pid
```



```
      %attlist.global;
    >

  <!ELEMENT port                (#PCDATA) >
  <!ATTLIST port
      %attlist.global;
    >

  <!ELEMENT portlist            (#PCDATA) >
  <!ATTLIST portlist
      %attlist.global;
    >

  <!ELEMENT program             (#PCDATA) >
  <!ATTLIST program
      %attlist.global;
    >

  <!ELEMENT protocol            (#PCDATA) >
  <!ATTLIST protocol
      %attlist.global;
    >

  <!ELEMENT size                (#PCDATA) >
  <!ATTLIST size
      %attlist.global;
    >

  <!ELEMENT url                 (#PCDATA) >
  <!ATTLIST url
      %attlist.global;
    >
```

10. Security Considerations

This Internet-Draft describes a data format 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.

11. 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-04.txt, December 28, 2000, work in progress.

[4] Mansfield, G. and D. Curry, "Intrusion Detection Message Exchange Format: Comparison of SMI and XML Implementations," draft-ietf-idwg-xmlsmi-01.txt, August 22, 2000, work in progress.

[5] World Wide Web Consortium (W3C), "Extensible Markup Language (XML) 1.0 (Second Edition)," W3C Recommendation, October 6, 2000. http://www.w3.org/TR/2000/REC-xml-20001006.

[6] World Wide Web Consortium (W3C), "Namespaces in XML," W3C Recommendation, January 14, 1999. http://www.w3.org/TR/1999/REC-xml-names-19990114.

[7] Berners-Lee, T., Fielding, R.T., and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," RFC 2396, August 1998.

[8] Mealling, M., "The IANA XML Registry," draft-mealling-iana-xmlns-registry-00.txt, November 17, 2000, work in progress.

[9] Rumbaugh, J., Jacobson, I., and G. Booch, "The Unified Modeling Language Reference Model," ISBN 020130998X, Addison-Wesley, 1998.

[10] Freed, N., "IANA Charset Registration Procedures," BCP 19, RFC 2278, January 1998.

[11] Alvestrand, H., "Tags for the Identification of Languages," RFC 3066, BCP 47, January 2001.

[12] International Organization for Standardization (ISO), "International Standard: Data elements and interchange formats - Information interchange - Representation of dates and times," ISO 8601, Second Edition, December 15, 2000.

[13] Mills, D., "Network Time Protocol (Version 3) Specification, Implementation, and Analysis," RFC 1305, March 1992.

[14] Mills, D., "Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI," RFC 2030, October 1996.

[15] Eastlake, D., Reagle, J., and D. Solo, "XML-Signature Syntax and Processing," draft-ietf-xmldsig-core-11.txt, November 1, 2000, work in progress.

12. Acknowledgements

The following individuals contributed substantially to this document and should be recognized for their efforts. This document would not exist without their help:

Dominique Alessandri, IBM Corporation
James L. Burden, California Independent Systems Operator
Marc Dacier, IBM Corporation
David J. Donahoo, AFIWC
Michael Erlinger, Harvey Mudd College
Ming-Yuh Huang, The Boeing Company
Joe McAlerney, Silicon Defense
Glenn Mansfield, Cyber Solutions, Inc.
Paul Osterwald, Intrusion.com
James Riordan, IBM Corporation
Stephane Schitter, IBM Corporation
Michael J. Slifcak, Internet Security Systems, Inc.
Paul Sangree, Cisco Systems
Michael Steiner, University of Saarland
Steven R. Snapp, CyberSafe Corporation
Stuart Staniford-Chen, Silicon Defense
Maureen Stillman, Nokia IP Telephony
Vimal Vaidya, AXENT
Andreas Wespi, IBM Corporation
John C. C. White, MITRE
Eric D. Williams, Information Brokers, Inc.
S. Felix Wu, North Carolina State University

13. Author's Addresses

David A. Curry
Internet Security Systems, Inc.
345 State Route 17 South
Upper Saddle River, NJ 07458 USA
Phone: +1 201 934-4207
Email: davy@iss.net

Herve Debar
France Telecom R & D
42 Rue des Coutures
14000 Caen FRANCE
Phone: +33 2 31 75 92 61
Email: herve.debar@francetelecom.fr

Intrusion Detection Working Group
Mailing List: idwg-public@zurich.ibm.com
To Subscribe: idwg-public-request@zurich.ibm.com
List Archive: http://www.semper.org/idwg-public/
Web Site: http://www.silicondefense.com/idwg/

Full Copyright Statement

Appendix A - Changes From the Last Draft

The following is the list of major changes that have been made to the IDMEF Data Model/XML DTD Internet-Draft since the last version.

A.1 Internet-Draft Document Changes

The Internet-Draft document has been completely rearranged, and some sections have been rewritten, to make the entire document read in a more logical way. This document represents a more complete "merge" of the previously separate data model and XML DTD documents, and removes the redundancies and contradictions between the two.

The data model and XML markup are presented together for each class now. This means, in general, that there is a UML diagram, an excerpt from the DTD, and explanatory text for each class.

The language on character encodings has been rewritten to correct numerous errors involving the distinctions between ISO/IEC 10646 and Unicode. In the process, the requirement that IDMEF messages "MUST" be encoded in UTF-8 or UTF-16 was changed to "SHOULD," to allow implementors to use other encodings if they need to (at the risk of losing portability).

The section on extending the IDMEF DTD has been completely rewritten (see A.4, below).

A.2 New Model for the User Class

The User class has been replaced with the new model defined by Glenn Mansfield and Dave Curry, with modifications suggested by Herve Debar. This change was proposed and agreed to on the idwg-public mailing list in October/November 2000.

The new class results in the following general format for user information:

```
<User category="unknown|application|os-device">
  <UserId type="original-user|current-user|target-user|user-privs|
               current-group|group-privs">
    <name>user name</name>
    <number>user id</number>
  </UserId>
  ...
</User>
```

See Section 5.2.6.2 for details.

A.3 New Date-Time Representation

The representation of date and time information has been replaced with the new model proposed defined by Paul Sangree. This change was proposed and agreed to at the December, 2000 IETF/IDWG meeting in San Diego.

The changes are:

1. Eliminate the <Time> element, and introduce a new <CreateTime> element to go with the <DetectTime> and <AnalyzerTime> elements.

2. Eliminate the <time> and <date> elements.

3. Adopt the ISO 8601:2000 standard for date and time formats.

4. Change <ntpstamp> from an element to an attribute.

This results in the following general format:

```
<Alert>
  <AnalyzerTime ntpstamp="0xBDFA4701.0x32C6">
    2000-12-25T01:00:01.15+00:00
  </AnalyzerTime>
  <DetectTime ntpstamp="0xBDFA4701.0x32C6">
    2000-12-25T01:00:01.15+00:00
  </DetectTime>
  <CreateTime ntpstamp="0xBDFA4701.0x32C6">
    2000-12-25T01:00:01.15+00:00
  </CreateTime>
</Alert>
```

Language has been added to the Internet-Draft to cover formatting the date/time strings according to ISO 8601:2000, and to cover some subleties of using the NTP timestamp.

See Sections 4.4.6, 5.2.5, and 7.4 for details.

A.4 New XML DTD Extension Mechanism

The content model for <AdditionalData> has been changed from "#PCDATA" to "ANY", and "xml" has been added as a possible value for the "type" attribute.

The purpose of this is to fix the extensibility problems discussed at the December, 2000 IETF/IDWG meeting in San Diego. By making this change, we can allow people to include additional DTDs in their IDMEF markup (e.g., one for packet headers), and to put all the new markup underneath <AdditionalData type="xml">. This change will also allow us to make better (i.e., correct) use of XML Namespaces.

The text in previous Internet-Drafts told people how to add or change elements in their IDMEF Messages. All of that language has been removed, and replaced with new requirements that basically say that the only way you can do extensions is by including new DTDs, and anything you include gets put under <AdditionalData type="xml">. Futhermore, any extensions you do add must use a different XML Namespace (i.e., they can't use "idmef" or the default namespace) to avoid conflicts with existing IDMEF elements and attributes.

This change also addresses the issues raised on the idwg-public mailing list by Tara Whalen (including data that uses a radically different data model, such as anomaly data) and Joe McAlerney (include packet header data). To do this, just write a DTD for the data you want to include, and put the data (with all your new tags) under <AdditionalData>. You get the data in the alert, and any managers that don't know what to do with it can just throw it away; the rest of the IDMEF format does not change.

NOTE: Dave Curry was tasked with investigating this problem in San Diego; the above is his solution. We are limited in how we can solve the problem, mostly because of the limitations imposed by DTDs. XML Schemas will, we think, allow us a more "elegant" solution, but they are currently only in Candidate Recommendation status within the W3C, which in effect means we'll be unable to use them until "Version 2" of IDMEF, given our current timetable.

See Section 6 for details.

A.5 Changes to the Service Class

The Service class is now an aggregate class of both Source and Target. To accomodate this, the <sport> and <dport> elements of <Service> have been replaced by a single <port> element, and the <portlist> element has been restored (it was going to be deleted).

This now means that "source" information goes under <Service> in <Source>, and "destination" information goes under <Service> in <Target>, which is more intuitive than putting it all under <Target>, as you used to have to do.

A.6 Support for Isolated Networks, Multi-Interface Sensors

Support for isolated networks and sensors with multiple interfaces has been added, as proposed by Paul Sangree and agreed to at the December, 2000 IETF/IDWG meeting in San Diego:

1. Added an optional <interface> attribute to <Source> and <Target> which can be used to identify the interface on which a network sensor saw the traffic.

2. Added optional attributes "vlan-num" and "vlan-name" to the <Address> element.

A.7 Unique Identifier Names

The unique identifier attributes used by many of the classes have been "unified" into two types:

1. The "analyzerid" attribute, used to identify analyzers, must be unique across all analyzers in the intrusion detection environment.

2. The "ident" attribute, used on many other classes, must be unique across all messages sent by the individual analyzer. The name of this attribute is now "ident" on all classes; previsouly it had several different names.

The Internet-Draft does not provide any guidance on how unique values should be created.

A.8 Removal of <Environment> and <Argument> Elements

The <Environment> and <Argument> elements have been removed, moving <arg> and <env> up one level in the XML. This was proposed by Paul Sangree and agreed to at the December, 2000 IETF/IDWG meeting in San Diego.

A.9 Removal of "unknown" AdditionalData Type

The value "unknown" has been removed from the list of possible values for the "type" attribute on <AdditionalData>, as it makes no sense. This was proposed by Paul Sangree and agreed to at the December, 2000 IETF/IDWG meeting in San Diego.

A.10 Documentation of Time Synchronization Caveats

The caveats on <AnalyzerTime> and its use for time synchronization, as presented by Paul Sangree at the December, 2000 IETF/IDWG meeting in San Diego, have been included in the Internet-Draft.

Some general recommendations on how implementors should handle the time synchronization problem have also been included.

See Section 7.3 for details.

Appendix B - Problem Issues and Proposed Changes Yet to Be Decided

This section describes problem issues that have arisen, as well as a number of changes that have been proposed, on the idwg-public mailing list or at the December, 2000 IETF/IDWG meeting in San Diego. They have (for the most part) not been discussed by the membership, and no approval/rejection decision has been made.

Discussion of these issues and proposals from interested parties, on the idwg-public mailing list or at the IDWG sessions during IETF quarterly meetings, is both welcome and encouraged.

B.1 Problem Issues

1. There are four reasons to provide the "analyzerid" and "ident" attributes:

   a. Allows XML elements under AdditionalData to refer to specific elements they are extending (see Section 8.7). This use is explicitly defined.

   b. Allows the ToolAlert and CorrelationAlert classes to provide references to the alerts that were used to generate them. This use is explicitly defined.

   c. Provides a "hint" to the manager so that it may make more efficient use of its database by internally referencing existing information, rather than duplicating it. This use is implicitly defined, but never stated explicitly.

   d. Allows an analyzer to conserve bandwidth by sending a reference to an already-transmitted data item, rather than repeating the data in every message. This use is hinted at, but never stated explicitly.

   The problem arises with (d) -- if this use is allowed, there is an assumption (or unstated requirement) that the manager will store all data received from the analyzer in persistent storage (a cache or database). If the manager does not do this, it may receive a reference to data it does not have, because there is no way for the manager to request that the analyzer retransmit the data.

   One possible workaround to this problem is to require the analyzer to retransmit the full data periodically, so that managers needing the data can capture it. This is plagued with problems however, in specifying how frequently the analyzer must retransmit the full data, and what managers do with references they receive before they have received the full data.

   A second idea is to create "parallel" elements for the reference notation (AnalyzerRef, SourceRef, NodeRef, etc.). These elements' "ident" attributes would refer to a previously-sent "normal" element (Analyzer, Source, Node, etc.). The specification would be modified to say that the "Ref" elements should only be used if some external mechanism has been provided by the implementor to ensure that the referenced data is either already available to the manager, or can be obtained by the manager.

Another, similar workaround would be to add another attribute, say "isref", to all the elements with "ident" attributes. If "isref" is "no", then the element is a "normal" element. If "isref" is yes, then the element refers to previously-sent data. The specification would again be modified to say that the "Ref" elements should only be used if some external mechanism has been provided by the implementor to ensure that the referenced data is either already available to the manager, or can be obtained by the manager.

A fourth possibility is to introduce some mechanism for managers to request data from analyzers. This idea was proposed for similar reasons a long time ago, and was the topic of much discussion before and during the Adelaide IDWG/IETF meeting in April, 2000. The ultimate decision reached in Adelaide was that the idea introduced too many other problems, and it was dropped at that time.

2. The uniqueness rules on "analyzerid" and "ident" may not be enough if the "reference" behavior described above is permitted. Because an "ident" is only unique within a single analyzer, both "ident" and "analyzerid" must be specified to get a value unique within the whole intrusion detection environment.

   This has been addressed in CorrelationAlert and ToolAlert by adding the "analyzerid" attribute to the alertident element. But if the reference behavior above is permitted, then a similar attribute would have to be added to all the classes that provide "ident" attributes.

   Another possibility would be to change the requirements for uniqueness in "ident" to make the values globally unique (or at least unique within the intrusion detection environment). To date however, no proposals for how to do this have been offered (which is one of the reasons the requirements are the way they are at present).

B.2 Proposals From Paul Sangree

1. Add <AdditionalData> to <Classification> to allow the provision of other information besides a URL and a name, such as identifying categories of attacks or vulnerabilities to which the alert belongs.

2. Add a <Context> element for representing alert context, such as the data that preceded or followed it in a TCP stream, or the packet contents that triggered the alert.

   NOTE: It was agreed in San Diego that this proposal needs more work. (The new extension mechanism may address this issue as well.)

3. Add summarized lists (count attributes). Some alerts may have a large number of sources and or targets; in such cases the scale of the attack may be useful information, but the actual source and target addresses are not.

   NOTE: It was agreed in San Diego that this is generally desirable, but that the issue of which particular elements should have count attributes added to them needs further work.

4. Add information about automated actions taken by an analyzer. Add a new Action aggregate class to Alert that describes what action was taken, and where.

5. Remove <sport> and <dport> from <Service> and add <port> to <Source> and <Target>, to allow specifying multiple source ports.

   NOTE: This proposal has been obsoleted by the changes to the Service class described in Section A.5.

6. Add "class", "manufacturer", "model", and "version" attributes to the <Analyzer> element to allow analyzers to be better identified, and to give hints on what stuff in <AdditionalData> might mean.

7. Fix the definition of the "impact" attribute by breaking it into three smaller attributes, "severity", "completion", and "type".

   NOTE: It was agreed in San Diego that the idea of breaking this up is desirable, but that the specifics still need to be worked out. Most people liked a "severity" with three (hi/med/low) to seven (use the ones from syslog - emerg/alert/crit/err/warning/notive/info/debug) values.

B.3 Proposals From Andy Walther

1. Add a <File> element (plus several sub-elements) to support host-based systems that want to provide information on files.

2. Add a <Connection> element (plus sub-elements) to consolidate information that's "buried in the hierarchy." Much of the contents in <Source>, <Target>, and <Service> would also appear here.