ABSTRACT

MCBRIDE, SEAN PATRICK. Data Organization and Abstraction for Distributed Intrusion Detection. (Under the direction of Dr. Robert St. Amant)

Due to the rapid pace of technological development, we find that old systems are “thrown away” in favor of newer technology. However, we find that data created by these earlier systems is persistent. A Digital Rosetta Stone [16] must be created to allow newer systems to correctly process data created by earlier technology. This document provides a case study of techniques that can be used to create a Digital Rosetta Stone between data formats and within a single evolving format.

The intrusion detection domain provides a solid basis for this study. In a distributed intrusion detection system, many sensors and analyzers must communicate with each other. The Intrusion Detection Message Exchange Format (IDMEF) is a standardized XML format for such communication. To its detriment, the IDMEF specification has been evolving since its inception. Also, the XML parsing during queries can be cumbersome and hinder intrusion detection. Therefore, two Digital Rosetta Stones were created. One migrates information between different versions of the IDMEF standard. The other translates IDMEF XML information into a relational database management system to improve query performance.
Data Organization and Abstraction for Distributed Intrusion Detection.

By

Sean Patrick McBride

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Department of
Computer Science

Raleigh
2005

APPROVED BY:

________________________________________  _________________________________________
Dr. Christopher Healey                  Dr. Laurie Williams

________________________________________
Dr. Robert St. Amant
Chair of Advisory Committee
Biography

Sean Patrick McBride was born in Syracuse, New York to parents Everett and Suzanne McBride on March 17, 1978. He attended Bucknell University in Lewisburg, Pennsylvania and received a Bachelor of Science degree in Computer Science and Engineering. In 2000 he took a position with Ciber, Inc. as consultant and worked for two years before attending North Carolina State University. After receiving his Master of Science degree from North Carolina State University he plans to continue his studies in Mixed-Initiative Interfaces and potentially receive a Ph.D.
Acknowledgements

I would like to thank the many different people helped me create this body of work. They provided the guidance and support I needed to reach my goal. First, I would like to thank my committee members, Dr. Robert St. Amant, Dr. Christopher Healey, and Dr. Laurie Williams. Their insight and direction proved to be outstanding. I would like to give a special thanks to my committee chair, Dr. St. Amant, who provided much aid and support beyond the topics of this thesis.

I cannot describe all the support provided by all of my peers. Alex, Brent, Lloyd, Sarat, Laura, Thomas and everyone in the lab for providing pointers and assistance while I researched this topic. Joe, Claudia, and Dan were largely responsible for making my time at NC State far more enjoyable.

I am deeply indebted to my family, who has always been there for me throughout my lifetime. Thank you for what you have done all these years.

Finally, I would like to thank Holly, whose love, support, and endless patience helped me through the final hours of my work. Thank you.
# Table of Contents

List of Tables ........................................................................................................ vii
List of Figures ......................................................................................................... viii
1. Introduction .........................................................................................................1
2. Background .........................................................................................................5
3. Upgrade Tool ........................................................................................................9
    3.1. System Design .......................................................................................... 9
    3.2. Data Elimination .......................................................................................12
    3.3. External Retention ....................................................................................12
    3.4. Specification Transgression .......................................................................12
    3.5. IDMEF Version Differences ......................................................................13
        3.5.1. IDMEFv00 – IDMEFv01 ................................................................. 14
        3.5.1.1. Version .................................................................................... 14
        3.5.1.2. Query and Response ................................................................. 14
        3.5.1.3. NTP Timestamp ....................................................................... 17
        3.5.1.4. Signature .................................................................................. 18
        3.5.1.5. Name and Classification ............................................................ 19
        3.5.1.6. Identification ............................................................................ 21
        3.5.1.7. ToolAlert .................................................................................. 23
        3.5.1.8. OverflowAlert ......................................................................... 23
        3.5.1.9. Netmask .................................................................................... 24
        3.5.1.10. Sources and Targets ................................................................. 25
        3.5.1.11. Dates ....................................................................................... 26
        3.5.1.12. Process and WebService .......................................................... 26
        3.5.1.13. User ........................................................................................ 27
        3.5.1.14. Impact ...................................................................................... 27
        3.5.1.15. Method ..................................................................................... 29
        3.5.1.16. Reaction and Confidence ......................................................... 30
        3.5.2. IDMEFv01 – IDMEFv02 ................................................................. 30
            3.5.2.1. AdditionalData ....................................................................... 30
        3.5.3. IDMEFv02 – IDMEFv03 ................................................................. 31
            3.5.3.1. Version .................................................................................. 31
            3.5.3.2. Identification ........................................................................ 31
            3.5.3.3. Classification ........................................................................ 34
3.5.3.4. Time ................................................................................................................34
3.5.3.5. Additional Data..................................................................................................36
3.5.3.6. Spoof and Decoy ..............................................................................................37
3.5.3.7. Source and Target ..........................................................................................38
3.5.3.8. Impact .............................................................................................................39
3.5.3.9. Correlation Name ............................................................................................40
3.5.3.10. Address ..........................................................................................................40
3.5.3.11. User ...............................................................................................................41
3.5.3.12. WebService ...................................................................................................44
3.5.3.13. Process ..........................................................................................................45
3.5.3.14. Service Ports .................................................................................................46

3.5.4. IDMEFv03 – IDMEFv04 ...................................................................................... 47
3.5.4.1. Version............................................................................................................47
3.5.4.2. Analyzer..........................................................................................................47
3.5.4.3. Alert ................................................................................................................47

3.5.5. IDMEFv04 – IDMEFv05 ...................................................................................... 49
3.5.5.1. Assessment......................................................................................................49
3.5.5.2. File System......................................................................................................53
3.5.5.3. WebService.....................................................................................................53
3.5.5.4. Analyzer..........................................................................................................54

3.5.6. IDMEFv05 – IDMEFv06 ...................................................................................... 54
3.5.6.1. Version............................................................................................................54

3.5.7. IDMEFv06 – IDMEFv07 ...................................................................................... 55
3.5.7.1. Identical Structure...........................................................................................55

3.5.8. IDMEFv07 – IDMEFv08 ...................................................................................... 55
3.5.8.1. Identical Structure...........................................................................................55

3.5.9. IDMEFv08 – IDMEFv09 ...................................................................................... 56
3.5.9.1. Identical Structure...........................................................................................56

3.5.10. IDMEFv09 – IDMEFv10 ..................................................................................... 56
3.5.10.1. File System....................................................................................................56
3.5.10.2. Simple Network Management Protocol version 3 ........................................59

3.5.11. IDMEFv10 – IDMEFv11 .................................................................................... 60
3.5.11.1. Identical Structure.........................................................................................60

3.5.12. IDMEFv11 – IDMEFv12 .................................................................................... 61
3.5.12.1. Impact ...........................................................................................................61
3.5.12.2. Analyzer........................................................................................................61
3.5.12.3. Classification .................................................................................................62
3.5.12.4. Identification .................................................................................................65
3.5.12.5. File System....................................................................................................66
3.5.12.6. Internet Protocol............................................................................................68

3.5.13. IDMEFv12 – IDMEFv13 .................................................................................... 69
3.5.13.1. Additional Data.............................................................................................69
List of Tables

Table 3.1 – IDMEFv00 to IDMEFv01 Identifier Attributes................................................... 22
Table 3.2 – IDMEFv02 to IDMEFv03 Identifier Attributes................................................... 33
Table 3.3 - IDMEF Time Formats .......................................................................................... 35
Table 3.4 – Impact Mapping Function.................................................................................... 52
Table 3.5 - Legal IDMEFv10 fstype Values........................................................................... 57
Table 3.6 - Legal IDMEFv10 permission Values................................................................... 57
List of Figures

Figure 1.1 – Digital Rosetta Stone Model [16]......................................................................... 2
Figure 3.1 – Upgrade Tool Design .......................................................................................... 11
Figure 3.2 – IDMEFv00 Query/Response Process .................................................................. 16
Figure 3.3 – IDMEFv00 Name to IDMEFv01 Classification Mapping ....................................... 21
Figure 3.4 - IDMEFv02 to IDMEFv03 User Mapping .............................................................. 43
Figure 3.5 - IDMEFv02 to IDMEFv03 WebService Argument Mapping ................................... 45
Figure 3.6 - IDMEFv04 Alert action Attribute ....................................................................... 49
Figure 3.7 – IDMEFv05 Assessment, Action, Impact, & Confidence Creation ......................... 52
Figure 3.8 - IDMEFv09 to IDMEFv10 File Analysis ............................................................... 58
Figure 3.9 - IDMEFv09 to IDMEFv10 FileAccess Analysis ..................................................... 59
Figure 3.10 - IDMEFv11 Classification to IDMEFv12 Reference Mapping ............................. 64
Figure 3.11 - IDMEFv11 to IDMEFv12 Identifiers .................................................................. 66
Figure 3.12 - IDMEFv12 to IDMEFv13 Target File Mapping .................................................. 70
Figure 4.1 – Database Import Tool Design ............................................................................. 74
Figure 4.2 – Database Connection Setup Screen ..................................................................... 75
1. Introduction

As technology continues to evolve at a rapid pace, there will be a greater need for new systems to integrate and process data output by antiquated systems. A common example of this exists in the Microsoft Office Word© word processing application. Microsoft Word offers the ability to read and edit documents created by earlier versions of Microsoft Office Word©. However, nearly every user of this feature finds that the process of reading and writing these earlier formats is not always reliable. Frequently information within the document is lost or mistranslated, causing differences to appear between the original document and the document presented by the application. Also, only a select number of previous versions are allowed to be read or written. A possible reason for these shortcomings and the incomplete set of allowed versions is the lack of a comprehensive Digital Rosetta Stone.

In 1799, Napoleon’s army discovered a large piece of basalt while digging the foundation of an addition to a fort outside el-Rashid (Rosetta), Egypt. This stone, known as the Rosetta Stone, contained an inscription of a decree passed by the council of priests in 196 BC praising King Ptolemy V. The decree was inscribed three times in three different languages: hieroglyphics, demotic (native Egyptian script), and Greek. At the time of the discovery, all knowledge of hieroglyphic texts had been lost, albeit, there was great knowledge of ancient Greek dialects. Thanks to the multiple inscriptions on the Rosetta Stone, we can translate many hieroglyphic and demotic texts that were once undecipherable.
The Digital Rosetta Stone [16] is a model for maintaining long-term access to digital documents. From this point forward, the word “document” is used to mean any digital entity created according to a specified and complete format. When successfully implemented, the Digital Rosetta Stone model can preserve access to antiquated documents regardless of software or hardware based obsolescence. The model consists of three distinct processes: knowledge preservation, data recovery, and document reconstruction. Knowledge preservation entails gathering and maintaining a dictionary of the type and representation of information which can be contained within a document and the media it is stored on. Data recovery involves extracting a document from a medium and then migrating the data to a non-antiquated medium. Document reconstruction is the process by which the information stored within the original document is translated to a format used by the modern system.

![Figure 1.1 – Digital Rosetta Stone Model [16]](image-url)
This document outlines a case study of a Digital Rosetta Stone in the intrusion detection domain. Comprehensive intrusion detection frequently involves multiple sensors and analyzers to communicate with each other before an attack on a computer system is identified. An evolving standard for communication between elements within an intrusion detection environment is the Intrusion Detection Message Exchange Format (IDMEF). This format has undergone several revisions and requires a Digital Rosetta Stone to allow an element within an intrusion detection environment to receive messages from another element outputting an obsolete IDMEF message.

This case study focuses primarily upon techniques used during the knowledge preservation and documentation reconstruction processes of the Digital Rosetta Stone. As the specification of storage media is beyond the scope of the IDMEF, the data recovery process is not applicable to this scenario. Data recovery is handled by Apache’s Xerces XML parser which handles all the I/O necessary to read the XML from disk. For this study, two tools were created to perform knowledge preservation and document reconstruction: an upgrade tool and a database import tool. Both the upgrade tool and database import tool employ knowledge preservation and document reconstruction processes. However, the upgrade tool addresses translation between different versions of a single evolved format and the database import tool describes translation between two very different formats.

Design and construction of these tools examine techniques that can be utilized to compile a metaknowledge archive [16]. This archive is the result of knowledge preservation process and is used to translate documents from one form to another during the document
reconstruction process. Also examined are multiple techniques that can be utilized to efficiently translate obsolete documents during the document reconstruction process. These techniques include data elimination, external retention, and specification transgression. Data elimination involves removing obsolete information from a document during translation. External retention requires temporary removal of information from a document until a later time when it can be reinserted into the translated document. Specification transgression requires temporarily ignoring some portion of a format specification, thereby creating an illegal document on a temporary basis. When applied to the sequential and direct migration process of the upgrade and database import tools, respectively, the data elimination, external retention, and specification transgression techniques provide insight as to how data formats should effectively evolve and how to exchange data between formats.
2. Background

The number of intrusion detection systems (IDSs) and intrusion detection techniques is growing at an extremely fast rate. These systems can be simple and monitor single events, such as network traffic or a host’s resources. They can also be complex and analyze multiple events in search of multi-staged attacks. Additionally, large-scale intrusion detection environments typically contain multiple sensors and analyzers to adequately protect multiple hosts. These sensors and analyzers are frequently produced by different vendors and, thus, have different proprietary communication protocols. The IDMEF structure is a solution to alleviate these differences without losing specific information that can be transmitted in the original proprietary schemas.

Some attacks are quite simple and only require a single event to achieve the attacker’s goals. An example of such an attack is the Ping of Death. The Ping of Death attack involves creating a single, malformed ICMP ECHO ("ping") datagram. If a vulnerable system receives an ICMP packet with a size of at least 65536 bytes, a buffer overflow will occur, causing variable results, including reboots, kernel panics, etc. A Ping of Death attack can easily be detected by a "network sniffer" identifying the malformed packet during a single event—its transmission along the network branch monitored by the "network sniffer".

Other attacks are multi-staged and require numerous events and actions on multiple hosts to achieve the attacker’s end goal. For example, Distributed Denial of Service (DDoS) attacks typically require an attacker to gain control of one or more "Master" hosts. A "Master" then uses various vulnerabilities to exploit and install services on multiple "Daemon" hosts.
Services on the “Daemon” host may lie dormant until commands are received from a “Master” host. At this point, “Daemon” hosts flood a target host, causing the target, and possibly the surrounding network, to be unusable. An example, of a DDoS incident is the August 17, 1999, “trinoo” attack on the University of Minnesota [13]. In this attack 227 “Daemon” hosts flooded a single system, rendering the target and target network unusable for two days.

DDoS and other multi-staged or multi-host attacks are typically more difficult to detect prior to their completion. Successful detection requires a much larger IDS environment than is necessary for detecting a single-event or single-host attack. It is quite likely that a single analyzer or sensor will not be able to adequately monitor a large-scale IDS environment. Therefore, multiple analyzers and sensors are needed to work in conjunction with each other to form a distributed intrusion detection system (DIDS). Together, these sensors and analyzers can identify potential wide-area attacks. By coordinating multiple analyzers and sensors (possibly from various vendors) a common communication medium must exist to allow successful communication of detected events.

The Intrusion Detection Working Group of the Internet Engineering Task Force is developing an Intrusion Detection Message Exchange Format (IDMEF) to provide a non-proprietary basis for multiple sensors and analyzers to exchange information. This information is stored in the form of an XML document, which can be sent by any means appropriate to the intrusion detection environment. Since its inception in late 1999 to early
2000, the IDMEF data model and corresponding XML structure has focused on and addressed five major problems inherent to event and alert communication.

1. Alert information is heterogeneous.
2. Sensor types vary.
3. Analyzer capabilities vary.
4. Operating environments vary.
5. Commercial vendors have different objectives.

Due to the enormous benefits of a standardized communication structure, many IDSs utilize IDMEF as a means of communicating and storing alert information. However, there is one major flaw in the adaptation of IDMEF into these systems—the standardization of IDMEF is not yet complete.

IDMEF continues to evolve to this date. There are fifteen different internet drafts specifying the IDMEF structure and there is no reason to believe that a final version will appear any time soon. (All past versions are archived at http://www.watersprings.org.) As new technologies develop they should be incorporated into the IDMEF specification. This includes information about technologies that can be reported as well as information presented by the technologies that are doing the reporting (i.e. technologies that may come under attack and intrusion detection technologies). Considering this, it is possible for sensors, analyzers, and managers to utilize multiple IDMEF versions within a single distributed intrusion detection environment. To allow accurate communication between these components, differences between IDMEF versions need to be addressed.
In addition to the problem of multiple IDMEF versions within a single intrusion detection environment, the use of XML as a communication structure provides a hindrance to intrusion detection. A typical IDS can generate hundreds of thousands of alerts per day. When coupled with other IDSs in a DIDS the number of generated alerts can grow exponentially. Therefore, it becomes increasingly difficult to analyze the large volume of data to identify and react to an attack before a significant amount of damage is done. Querying XML files has been shown to be significantly slower than querying a relational database management system (RDBMS). [22] Therefore, analyzers that utilize a relational database should detect an attack much more quickly than if IDMEF XML files were used alone.

To address these issues with IDMEF, we have created two tools that will allow improved communication and detection rates within a DIDS: an upgrade tool and a database import tool. The upgrade tool processes any IDMEF XML file and upgrades the contents of the file to the latest specification, which is then output as an IDMEF XML file. At the time of this writing, the latest version is IDMEF version 14. The database import tool loads the latest version IDMEF XML file (IDMEFv14) and stores the data into a more efficient RDBMS. By conjoining these tools, it is possible to load any version of IDMEF XML file and store its contents in a database as the most recent format, thereby allowing data from multiple sensors and analyzers to be efficiently examined regardless of the IDMEF version they output.
3. Upgrade Tool

The IDMEF specification is continually evolving and it is possible that elements within an intrusion detection environment may be utilizing any of the previously specified formats. Therefore, to allow multiple elements within an intrusion detection environment to effectively communicate with each other a tool is needed to act as a broker for incoming messages of varying formats and produce a single output format. The upgrade tool performs this action by accepting any version of IDMEF XML file and creating an equivalent IDMEFv14 XML file.

3.1. System Design

The IDMEF upgrade tool operates by taking a single IDMEF XML file as input and outputting an IDMEF version 14 XML file (shown in Figure 3.1). To accomplish this, the input file is parsed and its data is inserted into an IDMEF structure corresponding to the version specified by the user. The upgrade tool sequentially migrates data contained within one IDMEF structure to the next IDMEF structure. The process is repeated until the IDMEF version 14 structure is populated. At this time, the upgrade tool outputs an IDMEF version 14 XML file based upon the IDMEF version 14 structure. Differences between IDMEF versions and the processes used to address these differences are detailed in section 3.5.

This design was selected because of its simplified maintenance as new IDMEF specifications are released. When a new IDMEF internet draft is published, mappings from the previous structure to the new structure need to be examined and implemented. Also, a new IDMEF to
XML process needs to be written for the new IDMEF specification. Only if an attribute or element exists within the new specification and is not present in the immediately precedencing specification, should all previous specifications be examined to determine if the new attribute or element’s value needs to be imported from a previous version. Otherwise, no other previous IDMEF versions need to be examined. If a non-sequential design was chosen, mappings would need to be created between the new version and every preceding version.

The multiple IDMEF structures, corresponding to each if the IMDEF specification versions form the basis of the metaknowledge archive. Each structure outlines all of the information, which can be contained within a corresponding IDMEF XML file. The mappings between the elements within the structures outline the differences and translations between IDMEF versions and thereby complete the metaknowledge archive. These differences are their associated translations are detailed in section 3.5.
When the upgrade tool is performing the document recreation process, three techniques are utilized according to the metaknowledge archive: data elimination, external retention, and specification transgression. External retention and specification transgression apply only to a sequential translation of an evolved format. During a direct translation between two formats there is only one mapping involved between two formats. Therefore, information is either translated or eliminated across the single mapping.
3.2. Data Elimination

The process of data elimination is exactly as it sounds—information is permanently removed as it is translated from one version to another. If information contained within an IDMEF structure cannot be mapped to the next IDMEF structure, it is eliminated in a manner that does not breach any requirement of the subsequent IDMEF specification. Although this technique is lossy, it does not create any miscommunication. If an incorrect mapping were created to force all information to be translated to the next IDMEF version, the resulting alert or heartbeat would convey information about an event that did not occur. This could cause an IDS to not identify an attack because of a corrupted information base.

3.3. External Retention

External retention is an improvement over data elimination by creating a temporary loss of information. If data within one IDMEF structure cannot be mapped to the subsequent IDMEF structure, data retention would call for the data to be eliminated. However, should the data be useful during mappings to later IDMEF versions, external retention calls for the data to be stored outside the IDMEF structure until it is mapped to a later IDMEF version. This creates a temporary loss of information while the data is stored externally.

3.4. Specification Transgression

Specification transgression improves over data elimination and external retention, but at the cost of violating IDMEF specifications. Like external retention, specification transgression can be used when information within an IDMEF structure cannot be mapped to the next
IDMEF version. However, as external retention calls for the information to be removed and stored in a repository, specification transgression allows the information to remain within the IDMEF structure even though it is in violation of the immediate IDMEF specification. As the sequential translation process continues, the invalid is later mapped into a valid IDMEF structure according to the IDMEF specifications. This procedure creates no loss of information in any IDMEF structure.

3.5. IDMEF Version Differences

For the most part, IDMEF specifications contain incremental differences from the immediately preceding version. However, since the specifications are published via internet draft, the number of differences between versions can vary greatly. Internet drafts are only valid for six months at a time. Therefore, IDMEF specifications are typically released every six months regardless of the evolutionary state of the format. This results in some versions containing no differences from the previous versions. However, it also results in some versions differing greatly from the previous version. The only way to determine the extent of these differences is to thoroughly examine the latest version and compare it to the previous version, documenting all differences. Once documented, each difference must be addressed and a mapping created between the previous version’s IDMEF structure and the structure of the new version. If an element or attribute is added or a type changed in the new specification, all previous specifications should be analyzed to determine if external retention or specification transgression should be utilized to create a mapping from any previous version to the new version. Only after all of these steps have been completed is the metaknowledge archive able to encompass the new IDMEF specification.
3.5.1. IDMEFv00 – IDMEFv01

3.5.1.1. Version

During the upgrade process from IDMEF version 00 to IDMEF version 01, it is important to note that the version number for all IDMEF-Messages has changed from “1.1” to “0.1”. To account for this difference, if the version attribute of the IDMEF-Message element exists in the IMDEF version 00 structure, the upgrade tool changes the value to “0.1” in the IDMEF version 01 structure.

The version attribute in the Alert element has also changed between IDMEF version 00 and IDMEF version 01. In IDMEF version 00, the version attribute of the Alert class is deemed to be optional. However, in IDMEF version 01, this attribute must be present. To alleviate the situation where the version attribute is not present in the IDMEF version 00 structure, the upgrade tool always inserts a version attribute into the IDMEF version 01 Alert element with the mandatory value of “1”.

3.5.1.2. Query and Response

The Query and Response elements outlined within the IDMEF specification were created to allow analyzers to communicate with each other, while reducing network bandwidth. In the example given in the IDMEF version 00 specification [10], analyzer A has 100 messages to send to analyzer B. In each of these messages, the Source element’s information is exactly the same. Rather than sending all 100 messages with duplicate Source elements, analyzer A sends the first message with the complete Source information to analyzer B. For the purpose of this example, we will assign the value “12345” as the id of the source. Then, the
subsequent 99 messages contain an abbreviated Source element formed by the tag <Source id="12345" />. When analyzer B receives the first message it is processed and cached. Upon receiving the remaining 99 messages, analyzer B examines the <Source id="12345" /> tag and retrieves the cached Source information from the first message by looking up source information where the Source element’s id attribute is equal to “12345”. According to specification, identification information is unique for all Source elements reported by a single Analyzer within an IDS environment (see section 3.5.1.6). This process allows the Source element’s information to be added locally at analyzer B and reduce network traffic between analyzers A and B.

The Query and Response elements are only utilized if analyzer B no longer has the Source element’s information from the first message in the analyzer’s cache. This can happen for a variety of reasons including garbage collection, non-sequential receipt of messages, inability of the analyzer to cache information, etc. When this occurs, analyzer B cannot determine the Source element’s information from the <Source id="12345" /> tag. To determine this information, analyzer B will send a query to analyzer A containing the element tag <Source id="12345"/> and a unique id attribute. Upon receiving this message, analyzer A will retrieve complete information for the specified source and send this information in a response back to analyzer B, where the id of the response is the same as the id of the query. When analyzer B receives the response from analyzer A, analyzer B can populate the original alert message with the source information contained within the response message.
Only in the event of a cache hit—the receiving analyzer recognizes the abbreviated message—can this technique reduce network bandwidth for a small cost of processing by the receiving analyzer. However, in the event of a cache miss, the cost is fairly high as network bandwidth is consumed by the query/response mechanism as well as additional processing by both the sending and receiving analyzers. In the extreme event that the receiving analyzer cannot retrieve any cached information, the above example will result in 99 Query/Response combinations. This will consume much more network bandwidth and analyzer processing than if the Source element’s information was included in each of the original 100 messages.

In addition, the Query/Response technique assumes that the analyzer sending the original alert messages will be able to retrieve the needed information if a Query message is received. This cannot be guaranteed due to memory capacity limitations inherent in all computer systems. If an analyzer receives a query message relating to event records that have been cleared from memory, the analyzer cannot respond accordingly.
Since correct responses to queries cannot be guaranteed, and the Query/Response specification is deterministic of a protocol rather than a message structure, the query and response elements were dropped from the IDMEF specification. As the query and response elements were dropped from the IDMEF specification permanently, they can simply be dropped when upgrading from IDMEF version 00 to IDMEF version 01.

### 3.5.1.3. NTP Timestamp

In IDMEF version 01, the concept of a network time protocol (NTP) [21] was adopted to the specification. To incorporate NTP into the IDMEF structure, several changes had to be made. The first was to create a new element named ntpstamp. In practice, an NTP timestamp is a single unsigned 64-bit fixed point number representing a number of seconds relative to 0h 1 January 1900 GMT. The first 32 bits of the timestamp is the integer portion of the fixed-point number and the second 32 bits is the fraction portion. This schema allows a precision to the level of 200 picoseconds without causing undue complexity when more rough time estimates are required by an application.

In the IDMEF specification, NTP timestamps are represented as a single string containing two 32-bit hexadecimal values separated by a period (‘.’). The first of these two values represents the integer portion of the NTP timestamp and the second represents the fraction portion. For example, “0x3a2d8b3a.0x0” represents 16:41:30h 15 December 2000.

Another change to the IDMEF version 01 specification was to include the new ntpstamp element as an aggregate class of the Time, DetectTime, and AnalyzerTime elements. In each
of these situations, the ntpstamp is required to be present and will override the date and time elements if there is a discrepancy. During the upgrade process, ntpstamp elements are created using existing time and date information within the Time, DetectTime, and AnalyzerTime classes.

The final change to the IDMEF specification is to include the NTP timestamp in the additional data element. Since the Additional Data element is only a container for any possible data, the only change that needs to be performed is to include “ntpstamp” as a possible value of the type enumerated type. During the upgrade process, this will not require additional changes to the format as the incoming version will not contain a specified ntpstamp element and there is no adequately feasible process to determine if an unknown element is indeed an NTP timestamp. All unknown elements should remain unknown.

3.5.1.4. Signature

In IDMEF version 00, the Alert element can contain a signature element depicting the intrusion signature that caused the alert. In IDMEF version 01, this signature element was removed from the specification. The reason behind this could be that the presence of a signature element would require that all signatures be unique and known throughout the entire detection environment. Additionally, a signature applies only to misuse detection where the attack method is known [20]. A signature does not address anomaly detection.

Additionally, a single signature element in the Alert element requires that the intrusion detection environment contains a single set of misuse detection signatures. Establishing such
a set is very difficult if not impossible within an environment containing hardware and software analyzers from different vendors. A founding principle of the IDMEF data model is to provide coordination between multiple intrusion detection products. [26, 27, 28, 29, 30, 31, 32, 33, 34, 35] Creating a single set of misuse detection signatures across all products within an intrusion detection environment is extremely difficult, if not impossible. Therefore, the Signature element was dropped in IDMEF version 01. To account for this change, the upgrade tool omits all encountered Signature elements when upgrading from IDMEF version 00 to IDMEF version 01.

3.5.1.5. Name and Classification

In IDMEF version 00, the Alert element contains at least one Name element. This element is used to identify the vulnerability causing the alert. In IDMEF version 01, this Name element was expanded and renamed Classification. Classification is a more appropriate name for this identifying element. The name Name connotes that the attack has a name. However, this element focuses on the vulnerability which the attack attempted to exploit. By renaming the element Classification, the element more closely conveys the meaning that the Alert pertains to a specific vulnerability.

Both the Name and Classification elements make use of public vulnerability dictionaries [1, 5] to provide a common basis for vulnerability naming and information exchanges. These dictionaries provide common resources and naming for vulnerabilities regardless of platform and application constraints. This allows different types of analyzers from different vendors to use the same set of vulnerabilities and naming conventions regardless of their detection
techniques. These vulnerabilities are identified by library and an identifier. In the Name element, the library is identified by the origin attribute and the identifier is the string held within element. Similarly, the Classification element contains library information in an origin attribute and the identifier in a name sub-element. However, the Classification element expands upon the Name element by including a URL for a manager (human or computer) can obtain more information about the vulnerability.

When upgrading from IDMEF version 00 to IDMEF version 01, the upgrade tool moves name and origin information from the IDMEF version 00 Name element to the IDMEF version 01 element with only one modification. In IDMEF version 00, the allowed values for the origin attribute are “unknown”, “bugtraqid”, “cve”, and “policy”. Similarly, the allowed values in IDMEF version 01 are “unknown”, “bugtraqid”, “cve”, and “vendor-specific”. The slight difference in values requires that the upgrade tool examine the contents of the origin attribute of the Name element in IDMEF version 00. If the value found is “policy”, the origin attribute in the IDMEF version 01 Classification element is set to “unknown”. If the value found in IDMEF version 00 is not “policy”, then the existing value is migrated into the IDMEF version 01 structure without change. Although, the IDMEF version 01 attribute allows “vendor-specific” as a legal value for the origin attribute, “policy” from IDMEF version 00 cannot be mapped to “vendor-specific” in IDMEF version 01. The “policy” value in IDMEF version 00 represents an intrusion detection environment specific policy and the “vendor-specific” value in IDMEF version 01 represents an analyzer’s classification scheme. The incongruence of these meanings does not allow a mapping from “policy” to “vendor-specific”.
In IDMEF version 01, URL information is required to be present within the Classification element. To account for the newly introduced and required URL, the upgrade tool populates the url element of the Classification element with the value “unknown”. Although not a valid URL, the “unknown” value is sufficient to signify that the URL cannot be determined.

![Figure 3.3 – IDMEFv00 Name to IDMEFv01 Classification Mapping](image)

3.5.1.6. Identification

Between IDMEF version 00 and IDMEF version 01, many identification attributes have changed. The semantics behind the identification attributes, however, remains the same. Identifiers for Address, Analyzer, Node, Process, Service, Source, SNMPService, Target, User, and WebService elements are unique for a given set of attributes and sub-elements. This permits multiple data model elements, each with a different identifier, to a single physical element. The reasoning behind this is that the identifier must be unique for a single logical element within the data model. This logical element may or may not correspond to a single element in the physical structure of the environment.
The Alert and Heartbeat elements are slightly different. Each Alert and Heartbeat element generated by a single analyzer must have a unique identifier. Therefore, it is possible to have multiple Alerts with an equivalent identifier, given that the Alerts were generated by different analyzers.

When upgrading from IDMEF version 00 to IDMEF version 01, changing an element’s identifier attribute is a simple matter of changing the name of the attribute, only. The identifier’s value does not change. There is only one exception to this policy. For the Alert element, if the identification attribute contains the value “(none)”, the IDMEF version 00 default, it is changed to “0”, the IDMEF version 01 default. In IDMEF version 01, the Alert, Analyzer, Heartbeat, and Service identification attributes are required. However, the identification attributes are also required in IDMEF version 00. Therefore, no special processing is required to create a new identifier, if there is no identifier in the previous version. The changes between version 00 and version 01 are outlined in the table below.

<table>
<thead>
<tr>
<th>Element</th>
<th>Version 00 Identifier attribute</th>
<th>Version 01 Identifier attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>id</td>
<td>ident</td>
</tr>
<tr>
<td>Alert</td>
<td>id</td>
<td>alertid</td>
</tr>
<tr>
<td>Analyzer</td>
<td>id</td>
<td>ident</td>
</tr>
<tr>
<td>Heartbeat</td>
<td>id</td>
<td>heartbeatid</td>
</tr>
<tr>
<td>Node</td>
<td>id</td>
<td>ident</td>
</tr>
<tr>
<td>Process</td>
<td>id</td>
<td>ident</td>
</tr>
<tr>
<td>Service</td>
<td>id</td>
<td>ident</td>
</tr>
<tr>
<td>Source</td>
<td>id</td>
<td>sourceid</td>
</tr>
<tr>
<td>SNMPService</td>
<td>id</td>
<td>ident</td>
</tr>
<tr>
<td>Target</td>
<td>id</td>
<td>targetid</td>
</tr>
<tr>
<td>User</td>
<td>id</td>
<td>ident</td>
</tr>
<tr>
<td>WebService</td>
<td>id</td>
<td>ident</td>
</tr>
</tbody>
</table>
3.5.1.7. ToolAlert

In addition to new identification attributes, there is also a subtle change in the ToolAlert element’s alertid elements. The ToolAlert element is used to describe an identified attack tool such as “trinoo.” [13] In IDMEF version 00, there must be at least one alertid element contained within the ToolAlert element. However, it is possible that the Alert containing the ToolAlert is the only identified Alert related to the attack tool described by the ToolAlert element. In this situation the single alertid element contains redundant identification information. In IDMEF version 01, this problem was remedied by allowing the ToolAlert element to contain zero or more alertid elements. This change plays little role in the upgrade process, as alertids from IDMEF version 00 will be moved without change into the IDMEF version 01 structure. However, the change is significant and should be noted.

In addition to the alertid requirement changes, a new command element was added to the ToolAlert element. This addition contains an identified command or operation of the attack tool described by the ToolAlert element. In IDMEF version 01, there can be zero or more command as multiple or possibly no commands could be identified by the analyzer. Due to this specification change the upgrade tool omits command elements from the IDMEF version 01 structure when upgrading from IDMEF version 00 to version 01.

3.5.1.8. OverflowAlert

In IDMEF version 00, the OverflowAlert is required to maintain the data used to overflow a buffer during an attack. Although very beneficial, an analyzer may not be able to identify or store the data used the attack. Therefore, in IDMEF version 01, the buffer element of the
OverflowAlert element is allowed to be absent. Although this is a significant change in the data format scheme, it plays little role in the upgrade process as all buffer data present in the buffer element of the IDMEF version 00 structure will be moved to a buffer element in the IDMEF version 01 structure with no modifications.

3.5.1.9. Netmask

Netmask information was expanded in IDMEF version 01. In the IDMEF version 00 specification, there is no explicit placeholder for netmask information. Since a netmask can provide a great deal of information about a network address and the network that address is associated with, a netmask element was added to the Address element in the IDMEF version 01 specification. This element will contain IP version 4 or IP version 6 netmask information as it relates to the address specified. Additionally, a new address element was added to the Address element to contain the actual address specified. Note that this new element starts with a lowercase ‘a’ to differentiate it from the older Address element, which begins with a capital ‘A’. In addition to these changes, the category attribute of the Address element was expanded to allow “ipv4-net-mask” and “ipv6-net-mask” as legal values to declare that the Address element contains an Internet Protocol version 4 or Internet Protocol version 6 address and netmask, respectively. When upgrading from IDMEF version 00 to IDMEF version 01, there is no additional processing to account for netmasks. As IDMEF version 00 contains no netmask information, it is impossible to infer this information into the upgraded IDMEF version 01 structure. To complete the upgrade, the Address’s category attribute’s value will remain unchanged, the optional netmask element will be omitted, and a new
address element will be created to contain the address information originally contained within the Address element.

3.5.1.10. Sources and Targets

In IDMEF version 00, an analyzer is not required to specify whether or not a source is spoofed or a target is a decoy. This decision was most likely made because an analyzer may not be able to determine this information. For this reason the spoofed attribute of the Source element and the decoy attribute of the Target element could be omitted. However, not knowing if a Source is spoofed or a Target is a decoy can be an important piece of information for IDSs [6, 18]. Since the spoofed and decoy attributes both contain “unknown” as a legal value, these attributes are required in IDMEF version 01. In this version, if an analyzer cannot determine whether or not a Source is spoofed or a Target is a decoy, the spoofed or decoy attributes must contain “unknown”, respectively. To account for this change the upgrade tool analyzes all IDMEF version 00 Sources and Targets. If an IDMEF version 00 Source element does not contain a spoofed attribute, the corresponding IDMEF version 01 Source will contain a spoofed attribute equal to the value “unknown”. Similarly, if an IDMEF version 00 Target element does not contain a decoy attribute, the resulting IDMEF version 01 Target element will contain a decoy element with the value “unknown”. By this method the, now required spoofed and decoy attributes are accounted for.
### 3.5.1.11. Dates

Date formats differ between IDMEF version 00 and IDMEF version 01. According to the version 00 specification, dates should be represented as ccyy/mm/dd (e.g. 2004/02/02). In version 01, the specification states that “dashes” should be used instead of “slashes” (i.e. ccyy-mm-dd). To upgrade from version 00 to version 01, all date elements are examined, replacing slashes (‘/’) with dashes (‘-’).

### 3.5.1.12. Process and WebService

In IDMEF version 01, there must be exactly one name element within the Process element. However, in IDMEF version 00, it is possible for there to be no name elements in a Process element. To account for this change, the upgrade tool examines all Process elements from IDMEF version 00, looking for a name element. If a name element is found, the contents of the element are moved into the corresponding name element of the IDMEF version 01 structure. However, if a name element is not found, a new name element must be created for the corresponding Process element in IDMEF version 01. This new element contains a single value—“unknown”.

Similarly, the url element within the IDMEF version 00 of the WebService element is allowed to be absent. However, in IDMEF version 01, a WebService element must contain url element. Just as with the Process element, all WebService elements are examined by the upgrade tool in search of a url element. If found, the url element is migrated to IDMEF version 01 with no change. However, if a url element is not detected within the WebService element, a new url element is created within the IDMEF version 01 WebService element.
This new url element contains “unknown” as its single value. For both the Process and WebService elements the “unknown” value is sufficient to signify that the name or url element cannot be determined while maintaining the integrity of the IDMEF structure.

3.5.1.13. User

Similar to the changes for the name and url elements of the Process and WebService elements, respectively, the category attribute of the User element is required in IDMEF version 01, but optional in IDMEF version 00. However, unlike the Process and WebService elements, a new value does not need to be created for the User element’s category attribute. In both IDMEF version 00 and IDMEF version 01, the allowed values for the category attribute are “unknown”, “application”, and “os-device”. Therefore, the upgrade tool examines all User elements in IDMEF version 00. If a category attribute is not found, the category attribute of the IDMEF version 01 User element is set to “unknown”. This value states that no user information is provided. Additionally, “unknown” is the default value for both IDMEF version 00 and version 01. Therefore, a nonexistent category element is equivalent to a category element with the value “unknown”.

3.5.1.14. Impact

Between IDMEF version 00 and IDMEF version 01, the impact attribute of the Alert element contains two differences. The first of these differences is that the impact attribute is optional in IDMEF version 00 and required in IDMEF version 01. To account for this change, the upgrade tool examines all Alerts in the IDMEF version 00 structure. If an impact attribute is not found, one is created in the corresponding IDMEF version 01 Alert element. This
constructed attribute is then assigned the value “unknown”, which is default value for the element in both versions.

In addition, the list of possible values for the Alert element’s impact attribute has changed between IDMEF version 00 and IDMEF version 01. Specifically, the “successful-recon” value was removed and replaced with “successful-recon-limited” and “successful-recon-largescale”. In IDMEF version 00, the “successful-recon” value signifies that a reconnaissance probe completed successfully. In IDMEF version 01, this concept is expanded upon by using “successful-recon-limited” to indicate that a successful reconnaissance probe retrieved limited information. Similarly, the “successful-recon-largescale” value in IDMEF version 01 represents that a successful reconnaissance probe retrieved much information. Although this is an important difference, the upgrade tool does not attempt to convert the “successful-recon” value to “successful-recon-limited” or “successful-recon-largescale”. The reason for this is that the impact attribute is converted to an element in IDMEF version 05. This conversion uses a mapping function that uses “successful-recon”, “successful-recon-limited”, and “successful-recon-largescale” as possible inputs. Therefore, if the impact attribute is “successful-recon” in IDMEF version 00, the impact attribute of the corresponding Alert element in IDMEF version 01 is set to “successful-recon” even though this is not a legal value according to the IDMEF version 01 specification. This mapping is an example of specification transgression.
3.5.1.15. Method

Every Alert element in IDMEF version 00 may contain a method attribute to describe the technique used to detect an event. According to the IDMEF version 00 specification, this attribute can only contain the following values: “unknown”, “behavior”, “correlation”, “knowledge”, and “policy”. Having a limited list of possible values introduces a problem of maintenance. Maintaining a list of possible intrusion detection techniques is extremely difficult. The number of intrusion detection processes is growing at an extremely fast rate and judging from the amount of intrusion detection research currently being conducted this number will continue to grow. Also, intrusion detection is a highly competitive and secretive market and vendors frequently do not wish to divulge their detection techniques. As a result, analyzers may omit the method attribute from Alerts they generate based upon the privacy policies of the vendor. Thirdly, a small security flaw is introduced by describing how an event was detected. If an attacker knows how an analyzer detects an intrusion, then he will also know what to avoid when exploiting a vulnerability. Thereby, a collection of IDMEF messages become a library of known detection techniques for a set of analyzers.

For these reasons, most likely, the method attribute was dropped from the Alert element in the IDMEF version 01. To account for this change, the upgrade tool omits the method attribute when upgrading from IDMEF version 00 to IDMEF version 01. As this attribute or its equivalent does not exist in any other IDMEF version, it is unlikely that this information will be reintroduced to the format. Therefore, the attribute is dropped and not saved for reintroduction.
3.5.1.16. Reaction and Confidence

During the upgrade process from IDEMF version 00 to IDMEF version 01, the Reaction element and confidence attribute of the Alert element are removed. However, both pieces of information will reappear later in the upgrade process. Reaction information will emerge in the action attribute of the Alert element in IDMEF version 04 and the confidence information will reemerge in the Confidence element of the Assessment, which is a sub-element of Alert in IDMEF version 05. Because of this, the upgrade tool saves the reaction and confidence information associated with the Alert in IDMEF version 00 for reinsertion in IDMEF versions 04 and 05. To correctly associate this information with the upgraded alert, the reaction and confidence information is stored along with the unique identifier for the alert, provided by the upgrade tool.

3.5.2. IDMEFv01 – IDMEFv02

3.5.2.1. AdditionalData

Upgrading from IDMEF version 01 to IDMEF version 02 requires one change. In the IDMEF version 02 specification, “unknown” is no longer a valid value for the type attribute of the AdditionalData element. During an upgrade from version 01 to version 02, the type attribute is examined. If the value of the attribute is equivalent to “unknown” the attribute is omitted from the new IDMEF version 02 structure. This is a valid procedure as the type attribute is optional in IDMEF version 02 and the omission of the attribute conveys the same meaning as “unknown”, which was the default value for previous versions.
3.5.3. IDMEFv02 – IDMEFv03

3.5.3.1. Version

During the upgrade process from IDMEF version 02 to IDMEF version 03, it is important to note that the version number for all IDMEF-Messages has changed from “0.1” to “0.3”. During the upgrade from IDMEF version 02 to IDMEF version 03, if the version attribute of the IDMEF-Message element exists, it should be changed to “0.3”.

In addition, the version attribute of the Alert element has been dropped from the specification in IDMEF version 03. In previous IDMEF versions, the Alert element contained a version attribute designate the class hierarchy used within an IDMEF version. This was important because the data model used as a basis for IDMEF was maintained as a separate series of internet drafts. However, in the IDMEF version 03 specification, the data model was included within the format specification. Therefore, the version attribute of the Alert element was no longer needed. All the necessary version information is contained within the IDMEF-Message element’s version attribute. The upgrade tool conforms to this difference by omitting the version attribute from the Alert element when constructing the IDMEF version 03 structure.

3.5.3.2. Identification

Between IDMEF version 02 and IDMEF version 03 the names of many identification attributes have changes. These changes occurred to create a more uniform identification structure for all elements. The basis for this modification is that all identifiable elements can be uniquely identified by an (analyzer, identifier) pair, where the analyzer is the origin of the
IDMEF message. Within an intrusion detection environment, every analyzer must have a unique identifier. Element identifiers are not required to be unique within an intrusion detection environment, but must be unique within the context of an originating analyzer. These requirements will allow an element to be identified by an (analyzerid, identifier) pair, where the analyzerid uniquely identifies an originating analyzer within an intrusion detection environment and the ident value is unique for an element with respect to the originating analyzer.

To conform to the (analyzerid, identifier) definition for identification, all IDMEF version 03 Analyzer elements are identified by an optional analyzerid attribute. Similarly, all other identifiable IDMEF version 03 elements are identified by an ident attribute. The correlation of the ident attribute to the analyzerid attribute is built within the existing IDMEF structure.

This naming scheme requires the upgrade tool to rename multiple element identifiers found in the IDMEF version 02 structure to match the element identifiers for the IDMEF version 03 specification. Specifically, identifier attributes are renamed for the Alert, Analyzer, Heartbeat, Source, and Target elements. Only the attribute names are modified. The values of the identifier attributes are not changed between versions. Table 3.2 describes the IDMEF version 02 identifier attribute for each element and the renamed IDMEF version 03 identifier attribute.
Table 3.2 – IDMEFv02 to IDMEFv03 Identifier Attributes

<table>
<thead>
<tr>
<th>Element</th>
<th>IDMEF version 02 identifier</th>
<th>IDMEF version 03 identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alert</td>
<td>alertid</td>
<td>ident</td>
</tr>
<tr>
<td>Analyzer</td>
<td>ident</td>
<td>analyzerid</td>
</tr>
<tr>
<td>Heartbeat</td>
<td>heartbeatid</td>
<td>ident</td>
</tr>
<tr>
<td>Source</td>
<td>sourceid</td>
<td>ident</td>
</tr>
<tr>
<td>Target</td>
<td>targetid</td>
<td>ident</td>
</tr>
</tbody>
</table>

In addition to identifier attribute name changes, references to identifier attributes within the CorrelationAlert and ToolAlert elements have changed between IDMEF version 02 and IDMEF version 03. In IDMEF version 02, CorrelationAlert elements reference Alert elements via a list of alertid elements. Each of these elements contains a single string representing an Alert element identifier. In IDMEF version 03, the alertid elements have been replaced by an array of alertident elements. This name change conforms with the Alert element’s identifier attribute change from alertid to ident. Similarly, the alertid elements within the ToolAlert element are renamed alertident in the IDMEF version 03 structure. To account for this change, CorrelationAlert and ToolAlert elements are examined by the upgrade tool. Every alertid element found in the IDMEF version 02 CorrelationAlert and ToolAlert elements is renamed alertident in the IDMEF version 03 structure. The content of the renamed elements is not modified.

In the event that an analyzer does not provide an analyzerid, elements created by that analyzer cannot be uniquely identified within an intrusion detection environment—elements cannot satisfy the (analyzerid, ident) pair requirement. Therefore, all identifier attributes are optional according to the IDMEF version 03 specification. This stipulation forced the
Service element’s ident attribute to be optional in IDMEF version 03 while it is required in IDMEF version 02. This difference requires little processing by the upgrade tool. Since the ident attribute is required by all IDMEF specifications prior to IDMEF version 03, the value for the ident attribute is transferred into the IDMEF version 03 structure without modification.

### 3.5.3.3. Classification

Prior to IDMEF version 03, the Classification element contained an optional origin attribute. This attribute is used to designate a vulnerability library utilized in identification of the event causing the Alert. In IDMEF version 03, the origin attribute is no longer considered optional and is required to be present for every Classification element. To account for this change the upgrade tool examines all IDMEF version 02 Classification elements in search of an origin attribute. If the attribute exists, its value is inserted into the corresponding IDMEF version 03 origin attribute. If the attribute does not exist, a new origin attribute is created in the IDMEF version 03 structure. For all IDMEF versions containing a Classification element, the default value for an origin attribute is “unknown”. Therefore, when the upgrade tool creates a new origin attribute in the IDMEF version 03 structure, it is set equal to “unknown”.

### 3.5.3.4. Time

IDMEF version 03 is the first version to support SNTP version 4. Although the time structures in IDMEF version 01 and IDMEF version 02 are adequate to accommodate SNTP version 4, the time structures in IDMEF version 03 differ greatly to help make time
information more succinct. Firstly, Alert and Heartbeat elements each contained a single 
Time element in IDMEF version 02. This Time element contained date, time, and (UTC) 
offset elements, and an ntpstamp attribute describing when the Alert or Heartbeat was 
created. In addition, the Time element could contain DetectTime, and AnalyzerTime 
elements, each of which contained date, time, and (UTC) offset elements, and an ntpstamp 
attribute. These elements convey the time that the event causing an alert was detected and 
the current time of the analyzer, respectively.

In the IDMEF version 03 specification, the single Time element is replaced by three separate 
elements: AnalyzerTime, CreateTime, and DetectTime. Each of these elements contains a 
single datetime value and a (NTP) timestamp attribute. According to all IDMEF 
specifications from version 03 to version 14, a datetime value combines date, time, and UTC 
offset information into a single unit. This value can take one of nine possible formats 
included in the ISO 8601:2000 format. [17] See Table 3.3 below.

<table>
<thead>
<tr>
<th>IDMEF allowed ISO 8601:2000 Time Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>YYYY-MM-DDThh:mm:ssZ</td>
</tr>
<tr>
<td>YYYY-MM-DDThh:mm:ss.ssZ</td>
</tr>
<tr>
<td>YYYY-MM-DDThh:mm:ss,ssZ</td>
</tr>
<tr>
<td>YYYY-MM-DDThh:mm:ss+hh:mm</td>
</tr>
<tr>
<td>YYYY-MM-DDThh:mm:ss-hh:mm</td>
</tr>
<tr>
<td>YYYY-MM-DDThh:mm:ss.ss+hh:mm</td>
</tr>
<tr>
<td>YYYY-MM-DDThh:mm:ss.ss-hh:mm</td>
</tr>
<tr>
<td>YYYY-MM-DDThh:mm:ss+hh:mm</td>
</tr>
<tr>
<td>YYYY-MM-DDThh:mm:ss-hh:mm</td>
</tr>
</tbody>
</table>

Table 3.3 - IDMEF Time Formats
The IDMEF version 03 Alert element must contain a CreateTime element and may also contain an AnalyzerTime element or Detect Time element. Since Heartbeats are not created by detecting events, the IDMEF version 03 Heartbeat element cannot contain a DetectTime element, but it is required to contain a CreateTime element and may also contain an AnalyzerTime element.

When transitioning from IDMEF version 02 to IDMEF version 03, the upgrade tool extracts information from the AnalyzerTime and DetectTime elements in the IDMEF version 02 Time element to create the basis for the IDMEF version 03 AnalyzerTime and DetectTime elements. The date, time, offset, and timestamp information within the IDMEF version 02 Time element is used as the basis for the IDMEF version 03 CreateTime element. The ntpstamp attributes for the IDMEF version 03 AnalyzerTime, CreateTime, and DetectTime elements come directly from the IDMEF version 02 AnalyzerTime, Time, and DetectTime elements, respectively, without change. Creating the datetime value for the IDMEF version 03 AnalyzerTime, CreateTime, and DetectTime elements is derived by merging the date, time, and offset elements of the IDMEF version 02 AnalyzerTime, Time, and CreateTime elements. The formats of the date, time, and offset elements already match their respective subsections in the ISO 8601:2000 format, so parsing is kept at a minimum.

3.5.3.5. Additional Data

The transition from separate date and time elements to a single datetime value affects more than the time elements in Alerts and Heartbeats. The AdditionalData element must support datetime information in IDMEF version 03. In IDMEF version 02 the AdditionalData
element could contain a single date or time value, indicated by the type attribute equal to “date” or “time”. However, in IDMEF version 03, all date and time information must be in a datetime format and identified by type attribute being set to “date-time”. The upgrade tool handles this difference by examining all AdditionalData elements within an Alert or Heartbeat. If there is exactly one AdditionalData element with a type attribute equal to “date” and exactly one AdditionalData element with a type attribute equal to “time”, then the date and time information is concatenated to form a single date-time string as describe in section 3.5.3.4. For all other situations, all date and time AdditionalData elements are dropped from the structure; there is no method by which to associate multiple date and time elements or represent a date or time singly.

In addition to the “date-time” value, the IDMEF version 03 AdditionalData element’s type attribute is allowed to contain two values not present in the IDMEF version 02 specification: “portlist” and “xml”. These two values allow the AdditionalData element to contain a list of multiple ports as well as a hierarchical structure of information in XML format. Although a major change to structure and abilities, the upgrade tool does not perform much processing with regards to these modifications. Since portlist and XML information is not supported in IDMEF version 02 and earlier, this information will not be encountered during the upgrade process.

3.5.3.6. Spoof and Decoy

Between IDMEF version 00 and IDMEF version 01, the spoofed attribute of the Source element and decoy attribute of the Target element were changed from optional to required.
This was done to explicitly require an analyzer to signal if a Source or Target was known to be spoofed or decoyed, even when the analyzer is unable to tell. This requirement may have been established because knowing if a Source or Target is spoofed or decoyed can be very informative during intrusion detection [6, 18]. However, “unknown” is the default value for both the spoofed and decoy attributes. Therefore, if either of these attributes is omitted from a Source or Target element, the values of these attributes is implied as “unknown”. This can help reduce message size, while conveying the same information.

As the spoofed and decoy attributes of the Source and Target elements are required in IDMEF version 02, the upgrade tool does not need to perform any special action regarding these attributes. All information within the Source and Target elements is transferred from the IDMEF version 02 structure to the IDMEF version 03 structure without change. This includes the spoofed attribute of the Source element and decoy attribute of the Target element.

3.5.3.7. Source and Target

Besides their spoofed and decoy attributes, the Source and Target elements contain other differences between IDMEF version 02 and IDMEF version 03. The first of these differences is that both the Source and Target elements contain a new interface attribute in IDMEF version 03. The interface attribute is designed to be a free-form string that will indicate the interface that the Source or Target was seen on. As this information is not present in IDMEF version 02 and is not required in IDMEF version 03, the upgrade tool
omits the interface attribute from the Source and Target elements when upgrading from IDMEF version 02 to IDMEF version 03.

The second difference is that the Source element in IDMEF version 03 contains a Service element that is not present in the IDMEF version 02 structure. As it is possible for a known service to be the origin of an attack, a Service element must exist in the Source element to convey this information. For example, the Sapphire worm (also named Slammer) takes advantage of a Microsoft SQL Server buffer overflow vulnerability to replicate and transmit itself. In this attack SQL Server is not only the Target of an attack, but it is also a Source of the buffer overflow. Therefore, depending on how the event is analyzed, the service can be the Target, the Source, or both.

Although, this difference is important to note, it plays little role in the upgrade process. A Source element’s service is impossible to derive using the IDMEF version 02 structure and the Service element is not required to be present within the IDMEF version 03 structure. Therefore, the upgrade tool does not create a new Service element within the Source element when upgrading from IDMEF version 02 to IDMEF version 03.

3.5.3.8. Impact

Between IDMEF version 02 and IDMEF version 03, the impact attribute of the Alert element contains one subtle differences--the attribute is required in IDMEF version 02 and optional in IDMEF version 03. Although an important change to the IDMEF structure, this difference requires little processing on behalf of the upgrade tool. As the attribute is required in IDMEF
version 02, there should always be an impact attribute present within the Alert element when upgrading from IDMEF version 02 to IDMEF version 03. The upgrade tool moves the impact attribute along with its value to the corresponding IDMEF version 03 Alert element with no change.

### 3.5.3.9. Correlation Name

In IDMEF version 03, the CorrelationAlert element contains a name element that is not present in IDMEF version 02. This element is meant to contain the name of the correlation method used to group the Alerts designated by the CorrelationAlert element. According to the IDMEF version 03 specification, there must be exactly one name element within every CorrelationAlert element. Therefore, the upgrade tool created a new name element for every CorrelationAlert element in the IDMEF version 03 structure. Each new name element is populated with the value “unknown”, as this element cannot be derived from the IDMEF version 02 structure.

### 3.5.3.10. Address

The Address element within the IDMEF version 03 structure, contains a few minor differences from the Address element within the IDMEF version 02 structure. The first of these differences is that the category attribute is no longer required to be present for every Address element. Also, the category attribute was expanded to include “ipv6-addr-hex” as a legal value. This value designates that the Address element is a version 6 Internet Protocol address represented in hexadecimal form. These two changes to the category attribute require little additional processing by the upgrade tool. According to the IDMEF version 02
specification, the category element is required to be present for every Address attribute. Therefore the upgrade tool will always encounter a category attribute when upgrading from IDMEF version 02 to IDMEF version 03. The attribute and its value will be transferred to the IDMEF version 03 structure without change, unless the category element’s value is “ipv6-addr” and Address element’s address element contains a hexadecimal value. In this situation, the “ipv6-addr” value of the category element is changed to “ipv6-add-hex” in the IDMEF version 03 structure.

A second group of differences between the IDMEF version 02 and IDMEF version 03 Address elements involves the incorporation of virtual LAN information. The IDMEF version 03 Address element is allowed contain two attributes that do not exist in the IDMEF version 02 structure: vlan-name and vlan-num. These optional attributes contain the name or number, respectively, of the virtual LAN that the address belongs to. As IDMEF version 02 does not support virtual LAN information, the upgrade tool omits the vlan-name and vlan-num attributes from the IDMEF version 03 Address element.

3.5.3.11. User

The IDMEF version 02 User element contains name, user id, group, group id, etc. Although this information is important during intrusion detection process, this information is misrepresentative of an actual user. In many operating systems, a human user can have access to multiple user names, groups, etc. Therefore, in the IDMEF version 03 structure, a new element contains all of this information—UserID. Every UserID element is allowed to contain an ident attribute to uniquely identify the UserID as described in section 3.5.3.2. The
UserID element may also contain a type attribute that will describe what the UserID represents (i.e. current, original, or target user, user privileges, current group or group privileges). As many allow both a user name and user number, the UserID element contains two elements that contain this information: name and number. At least one of these elements must be populated to specify a valid UserID.

The User element remains in IDMEF version 03, but the contents of the element are quite different than previous versions. The IDMEF version 03 User element contains two optional attributes: ident and category. Both of these elements exist in IDMEF version 02 and are unchanged in IDMEF version 03 with an exception that the category attribute is not required in IDMEF version 03 as it is in IDMEF version 02. All elements within the IDMEF version 02 User element do not exist in IDMEF version 03. Only an array of one or more UserID elements exists within the IDMEF version 03 User element.

The upgrade tool handles these differences by examining the User element within the IDMEF version 02 Source and Target elements. The IDMEF version 02 ident and category attributes and values are moved into the IDMEF version 03 structure without change. If the IDMEF version 02 User element contains a name or uid element, a new UserID element is created within the corresponding IDMEF version 03 User element. The new UserID element contains a type attribute set with the value “current-user” and name or number elements containing the values found in the IDMEF version 02 User element’s name or uid elements, respectively. Similarly, if the IDMEF version 02 User element contains a group or gid element, a UserID element is created within the corresponding IDMEF version 03 User...
element. This UserID element contains a type attribute set with the value “current-group” and name and number elements containing the values found in the IDMEF version 02 User element’s group and gid elements, respectively. The serial and address elements within the IDMEF version 02 User element are not transferred to the IDMEF version 03 structure as there is no adequate structure to house this information and associate it with a specific User element.

![Diagram](image-url)

**Figure 3.4 - IDMEFv02 to IDMEFv03 User Mapping**
3.5.3.12. WebService

Between IDMEF version 02 and IDMEF version 03, the WebService element changed in two ways. The first of these is that the ident attribute does not exist in IDMEF version 03. The WebService element is designed to be a subclass of the Service element. As XML does not support subclassing, the WebService element is an aggregate class within the Service element. Since the Service element contains an ident attribute, there is no need for an ident attribute within the WebService element. According to the definition of the Service element’s identifier, the ident element is unique for all sets of values within the Service element. Thus, if a Service element does not contain a WebService element or has a WebService element detailing different information, the Service element would have a different ident value. The upgrade tool accounts for this difference by omitting all ident attributes from the IDMEF version 03 structure when migrating data from the IDMEF version 02 WebService element.

The second difference between IDMEF version 02 and IDMEF version 03 WebService elements revolves around arguments. In IDMEF version 02, the WebService element contains a single Arguments element, which contains no information besides an array of arg elements. Each of these arg elements contains a single argument value. In IDMEF version 03, the Arguments element does not exist. As a replacement, the WebService element contains an array of arg elements each containing a single argument, exactly like the IDMEF version 02 arg elements. This structure eliminates an unnecessary tier and decreases the IDMEF message size. The upgrade tool handles this change by moving all the arg elements...
within the IDMEF version 02 Arguments element directly into the IDMEF version 03 WebService element, without modification.

3.5.3.13. Process

Similar to the removal of the Arguments element from the WebService element, both the Arguments and Environment elements are removed from the Process element when transitioning from IDMEF version 02 to IDMEF version 03. In IDMEF version 02, both the Arguments and Environment elements are simply containers for arrays of arg and env elements, respectively. No additional information is provided. In IDMEF version 03, these unnecessary container elements do not exist. The Process element contains an array of arg elements and an array of env elements, directly. The upgrade tool mirrors this change by
examining each Arguments and Environment element within an IDMEF version 02 Process element. The two arrays of arg and env elements are extracted from the Arguments and Environment elements and reinserted into the corresponding IDMEF version 03 Process element without modification.

### 3.5.3.14. Service Ports

As the IDMEF version 03 structure allows for Service information to be contained within both Source and Target elements, there is no need for source port and destination port information to be stored within a Service element. In IDMEF version 02, Service information was only present within a Target element. Therefore, it is necessary to include source and destination port information encapsulated within sport and dport elements, respectively. However, in IDMEF version 03, Service elements exist within both Source and Target elements. If sport and dport information remained with the Service element, this information would be redundant as source and destination port information would be repeated in both the Source and Target element’s Service element. To prevent redundant information, the Service element contains only one port element in the IDMEF version 03 structure. The sport and dport elements do not exist. If the Service element exists within a Source element, it is implied that the port is a source port. Similarly, if the Service element exists within a Target element, it is implied that the port is a destination port.

The upgrade tool handles this difference by moving data within an IDMEF version 02 Service element’s dport element into the corresponding IDMEF version 03 Service element’s port element. This technique is used because IDMEF version 02 only supports Service
information for targets of an event. Therefore, it is impossible to correctly derive source
service information and insert a Service element into an IDMEF version 03 Source element.
Only if a Source’s Service element can be derived, could sport data be inserted into the new
port element.

3.5.4. IDMEFv03 – IDMEFv04

3.5.4.1. Version

During the upgrade process from IDMEF version 03 to IDMEF version 04, it is important to
note that the version number for all IDMEF-Messages has changed from “0.3” to “0.5”.
During the upgrade from version 03 to version 04, if the version attribute of the IDMEF-
Message element exists, it should be changed to “0.5”.

3.5.4.2. Analyzer

In the IDMEF version 04 specification, four additional attributes were added to the Analyzer
element: class, manufacturer, model, and version. These attributes were included in the
specification to provide additional information about the hardware or software analyzer
sending the message. All of these attributes are free form strings and optional according to
the IDMEF version 04 specification. Therefore, the upgrade tool omits these attributes when
upgrading from IDMEF version 03 to IDMEF version 04.

3.5.4.3. Alert

There are two minor differences in the Alert element between IDMEF version 03 and
IDMEF version 04. The fist of these differences is that the impact attribute of the Alert
element was changed from an enumerated type to a free form string. In IDMEF version 03, the impact attribute was only allowed to contain the following values: “unknown”, “bad-unknown”, “not-suspicious”, “attempted-admin”, “successful-admin”, “attempted-dos”, “successful-dos”, “attempted-recon”, “successful-recon-limited”, “successful-recon-largescale”, “attempted-user”, and “successful-user”. In IDMEF version 04, the impact attribute is not limited to any specific set of values. Therefore, the upgrade tool simply uses one of the previously designated values as the new value for the impact attribute of the Alert element in IDMEF version 04.

The second difference between the Alert element in IDMEF version 03 and IDMEF version 04 is that an action attribute was added. This attribute is congruent to the Reaction element found in IDMEF version 00. Although, this element was eliminated in IDMEF version 01, the upgrade tool saved the string value contained within the element, if one existed. Upon creating the IDMEF version 04 structure, the upgrade tool can use this previous value as the value of the new action attribute. If there was no Reaction element or the IDMEF structure is not being upgraded from version 00, the optional impact attribute is omitted from the new structure. Since it is possible for a single IDMEF-Message to contain multiple alerts and heartbeats, the upgrade tool associates a stored reaction element with the unique identifier that the tool has provided for the alert. This allows the tool to create the action attribute in the correct Alert element in IDMEF version 03.
3.5.5. IDMEFv04 – IDMEFv05

3.5.5.1. Assessment

In IDMEF version 05, a new logical grouping of elements was created to assess a possible attack. This new structure involves creating four new IDMEF elements: Assessment, Action, Impact, and Confidence. Incorporating these elements involves inserting the Assessment element into the Alert element, then placing the Action, Impact, and Confidence elements within the Assessment element. To complete the upgrade from IDMEF version 04 to IDMEF version 05, the upgrade tool must create a new Assessment element if one of three situations occurs. The first is that the IDMEF version 04 Alert contains an action attribute. The second is if the IDMEF version 04 Alert contains an impact attribute. And the third is if the Alert contains a saved confidence value from the IDMEF version 00 form of the alert. The latter situation can only occur if the upgrade process began with IDMEF version 00. If any of these situations occur, a new Assessment element must be added to the Alert element. If none of the three situations occurs, then the Assessment element is omitted from the Alert element.
The Assessment element is only a container for three different elements: Action, Confidence, and Impact. The Action element contains data relevant to a response to an event. The Action element contains two possible values used to convey this information. The first is the category attribute, which can contain “block-installed”, “notification-sent”, “taken-offline”, or “other” as legal values. The second value is a description string existing between the <Action> and </Action> XML tags. The upgrade tool creates an Action element using the action attribute of the Alert element from the IDMEF version 04 specification. If this attribute exists, the value is compared to the list of legal category values. If there is a match, an Action element is created with a category attribute equal to the IDMEF version 04 Alert element’s action attribute. If there is no match, an Action element is created with the category attribute equal to “other” and the Action element’s contents (description) is set to the incoming attribute. Finally, if there is no action attribute in the incoming IDMEF version 04 Alert element, no Action element is created—the IDMEF version 05 specification allows for zero or more Action elements per Assessment element.

A second sub-element of the Assessment element is the Confidence element. This element provides an estimation of the validity of the analyzer’s assessment of the event. In IDMEF version 00, there is a confidence attribute for the Alert element. This attribute is allowed to contain integer values ranging from 0 to 100, with 0 meaning that the confidence value is unknown. However, this attribute is removed when upgraded to IDMEF version 01. The upgrade tool, recognizing this attribute, stores the confidence attribute’s value in association with the Alert element for reinsertion in the IDMEF version 05 Confidence element. If the upgrade process began with IDMEF version 00 and there was a confidence attribute
associated with an alert and the value of this attribute is not equal to 0 a new Confidence element is created within the Assessment element of the associated Alert. This new Confidence element contains a rating attribute and a numeric rating, with values from 0.0 to 1.0, if the rating attribute is set to “numeric”. The upgrade tool populates the new Confidence element with a rating attribute set to “numeric” and a numeric rating equal to 1/100 of the IDMEF version 00 confidence attribute value.

The third element that may be created within the Assessment element is the Impact element. This element describes the analyzer’s assessment of the impact caused by the event on the target. This information is contained within three optional attributes: severity, completion, and type. Each of these three attributes is allowed a specific set of possible values. The severity element can have the values “low”, “medium”, or “high”. The Completion attribute can be “failed” or “succeeded”. And the type attribute can be “admin”, “dos”, “file”, “recon”, “user” or “other”. In IDMEF version 04, the Alert element contained an impact attribute that was allowed to contain free-form text with no restrictions. This structure provides an impasse in the upgrade process and creating a new Impact element for an Alert. However, in the versions prior to IDMEF version 04, the impact attribute of an Alert was only permitted to be a specific set of values: “unknown”, “bad-unknown”, “not-suspicious”, “attempted-admin”, “successful-admin”, “attempted-dos”, “successful-dos”, “attempted-recon”, “successful-recon”, “successful-recon-limited”, “successful-recon-largescale”, “attempted-user”, and “successful-user”. Knowing these values the upgrade process performs a mapping function to translate these values into a new Impact element. This mapping function is described in the table below.
Figure 3.7 – IDMEFv05 Assessment, Action, Impact, & Confidence Creation

Table 3.4 – Impact Mapping Function

<table>
<thead>
<tr>
<th>Alert.impact</th>
<th>Impact.severity</th>
<th>Impact.completion</th>
<th>Impact.type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unknown</td>
<td>&lt;blank&gt;</td>
<td>&lt;blank&gt;</td>
<td>&lt;blank&gt;</td>
</tr>
<tr>
<td>Bad-unknown</td>
<td>High</td>
<td>&lt;blank&gt;</td>
<td>&lt;blank&gt;</td>
</tr>
<tr>
<td>Not-suspicious</td>
<td>Low</td>
<td>&lt;blank&gt;</td>
<td>&lt;blank&gt;</td>
</tr>
<tr>
<td>Attempted-admin</td>
<td>&lt;blank&gt;</td>
<td>&lt;blank&gt;</td>
<td>Admin</td>
</tr>
<tr>
<td>Successful-admin</td>
<td>&lt;blank&gt;</td>
<td>Succeeded</td>
<td>Admin</td>
</tr>
<tr>
<td>Attempted-dos</td>
<td>&lt;blank&gt;</td>
<td>Succeeded</td>
<td>Dos</td>
</tr>
<tr>
<td>Successful-dos</td>
<td>&lt;blank&gt;</td>
<td>Succeeded</td>
<td>Dos</td>
</tr>
<tr>
<td>Attempted-recon</td>
<td>&lt;blank&gt;</td>
<td>&lt;blank&gt;</td>
<td>Recon</td>
</tr>
<tr>
<td>Successful-recon</td>
<td>&lt;blank&gt;</td>
<td>Succeeded</td>
<td>Recon</td>
</tr>
<tr>
<td>Successful-recon-limited</td>
<td>&lt;blank&gt;</td>
<td>Succeeded</td>
<td>Recon</td>
</tr>
<tr>
<td>Successful-recon-largescale</td>
<td>&lt;blank&gt;</td>
<td>Succeeded</td>
<td>Recon</td>
</tr>
<tr>
<td>Attempted-user</td>
<td>&lt;blank&gt;</td>
<td>&lt;blank&gt;</td>
<td>User</td>
</tr>
<tr>
<td>Successful-user</td>
<td>&lt;blank&gt;</td>
<td>Succeeded</td>
<td>User</td>
</tr>
<tr>
<td>&lt;other&gt;</td>
<td>&lt;blank&gt;</td>
<td>&lt;blank&gt;</td>
<td>&lt;blank&gt;</td>
</tr>
</tbody>
</table>
3.5.5.2. File System

The IDMEF version 05 specification marks the emergence of file system information with the message exchange format. Since individual files can be the target of an attack, a new FileList element was added to the Target element in IDMEF version 05. This new element is simply a container for one or more File elements, each containing information pertaining to access, inode, linkages, data size, name, path, etc. Despite introduction of this new information, the upgrade tool does not process the file system information when upgrading from IDMEF version 04 to IDMEF version 05 for two reasons. First, there is no file information included in IDMEF version 04. Second, the FileList element, which contains all this new file information, is not required within the Target element.

Similarly, the Node and UserID elements were expanded to incorporate file information. The category attribute of the Node element is allowed to contain “hosts” as a valid value in IDMEF version 05. Within the UserID element, the type attribute includes “other-privs” as a legal value. Although, these changes are directly related to existing IDMEF version 04 and 05 elements, the allowed values of these attributes within IDMEF version 04 still exist within IDMEF version 05. Therefore, the upgrade tool moves all the data within these attributes from the IDMEF version 04 structure to the IMDEF version 05 structure with no modifications.

3.5.5.3. WebService

During the upgrade from IDMEF version 04 to IDMEF version 05, the WebService element undergoes one slight change. In IDMEF version 04, the method element within the
WebService element contains the HTTP method used in the service request. However, in the IDMEF version 05 specification, a new http-method element contains this information and the previous method element is eliminated. Since the structure of these two elements are identical—free form strings—the upgrade tool simply moves the string information from the IDMEF version 04 method element into the IDMEF version 05 http-method element.

3.5.5.4. Analyzer

In the IDMEF version 05 specification, two additional attributes were added to the Analyzer element: ostype and osversion. These attributes were included in the specification to provide operating system information about the hardware or software analyzer sending the message. Both of these attributes are free form strings and optional according to the IDMEF version 05 specification. Therefore, the upgrade tool omits these attributes when upgrading from IDMEF version 04 to IDMEF version 05.

3.5.6. IDMEFv05 – IDMEFv06

3.5.6.1. Version

During the upgrade process from IDMEF version 05 to IDMEF version 06, it is important to note that the version number for all IDMEF-Messages has changed from “0.5” to “1.0”. During the upgrade from version 05 to version 06, if the version attribute of the IDMEF-Message element exists, it should be changed to “1.0”.

54
3.5.7. IDMEFv06 – IDMEFv07

3.5.7.1. Identical Structure

Between IDMEF version 06 and IDMEF version 07, there are no structural or logical differences in the format specifications. The only reason for a new version is that the previous version had expired. Therefore, a new identical version had to be created to maintain a current working version. There are only a few formatting and wording differences and corrections between the specifications themselves—not the information described by the specifications. As a result, the upgrade tool simply moves all data from the IDMEF version 06 structure to the IDMEF version 07 structure. There are no modifications to the data during this stage.

3.5.8. IDMEFv07 – IDMEFv08

3.5.8.1. Identical Structure

Between IDMEF version 07 and IDMEF version 08, there are no structural or logical differences in the format specifications. The only reason for a new version is that the previous version had expired. Therefore, a new identical version had to be created to maintain a current working version. There are only a couple wording differences between the specifications themselves—not the information described by the specifications. As a result, the upgrade tool simply moves all data from the IDMEF version 07 structure to the IDMEF version 08 structure. There are no modifications to the data during this stage.
3.5.9. IDMEFv08 – IDMEFv09

3.5.9.1. Identical Structure

Between IDMEF version 08 and IDMEF version 09, there are no structural or logical differences in the format specifications. The only reason for a new version is that the previous version contained a few minor errors. These errors were corrected in IDMEF version 09. However, the information conveyed in both versions is identical. As a result, the upgrade tool simply moves all data from the IDMEF version 08 structure to the IDMEF version 09 structure. There are no modifications to the data during this stage.

3.5.10. IDMEFv09 – IDMEFv10

3.5.10.1. File System

Between IDMEF version 09 and IDMEF version 10, two significant differences appear within the file system information. The first of these changes is that the fstype attribute of the File element has changed from a free-form string in IDMEF version 09 to a specified set of values in IDMEF version 10. Similarly, the permission elements within the FileAccess element were, also, allowed to be free-form text elements in IDMEF version 09 and must be one of a specific set of values in IDMEF version 10. To accommodate this change the upgrade tool examines the contents of the IDMEF version 09 fstype attribute and permission elements and compares them to the legal set of values in IDMEF version 10. The legal values for the File element’s fstype attribute in IDMEF version 10 are shown in Table 3.5. The specified values for the permission elements of the FileAccess element are shown in Table 3.6. If the IDMEF version 09 value of the fstype attribute and permission element match the legal values for IDMEF version 10, the information is moved into the IDMEF
structure without change. However, if the IDMEF version 09 information does not match the legal lists for IDMEF version 10, the upgrade tool must make some modifications to the data structure.

<table>
<thead>
<tr>
<th>ftype</th>
<th>permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>ufs</td>
<td>noAccess</td>
</tr>
<tr>
<td>efs</td>
<td>read</td>
</tr>
<tr>
<td>nfs</td>
<td>write</td>
</tr>
<tr>
<td>afs</td>
<td>execute</td>
</tr>
<tr>
<td>ntfs</td>
<td>search</td>
</tr>
<tr>
<td>fat16</td>
<td>delete</td>
</tr>
<tr>
<td>fat32</td>
<td>executeAs</td>
</tr>
<tr>
<td>pcfs</td>
<td>changePermissions</td>
</tr>
<tr>
<td>joilet</td>
<td>takeOwnership</td>
</tr>
<tr>
<td>iso9660</td>
<td></td>
</tr>
</tbody>
</table>

If there is a mismatch for the ftype attribute, the upgrade tool omits the ftype attribute from the File element in the IDMEF version 10 structure. Although the IDMEF version 10 specification states that the ftype attribute is required for each File element, this element is allowed to be absent from the File element starting with IDMEF version 12. Therefore, even though omitting the ftype attribute from the File element is a violation of IDMEF version 10, it will be legal in the final IDMEF version 14 structure. This process is further described in Figure 3.8.
Similarly, if there is a mismatch for a permission element within the FileAccess element, the upgrade tool omits the information. However, at least one permission element is required to exist within every FileAccess element for all subsequent IDMEF versions. Therefore, all permission elements cannot simply be omitted from a FileAccess element. Rather, the entire FileAccess element is removed from the parent File element, if there are no valid permission elements. Although there is a loss of information by this process, this is the only legal change allowed under the current set of IDMEF specifications.
3.5.10.2. Simple Network Management Protocol version 3

In IDMEF version 10, support for Simple Network Management Protocol version 3 (SNMPv3) [15] was incorporated into the IDMEF structure. Whereas earlier IDMEF versions only supported SNMPv1 [9] and SNMPv2 [15]. SNMPv1 and SNMPv2 services utilize three major pieces of information: a community, an object identifier, and a command. The community is a specific SNMP administrative domain under which an SNMP command is issued to a device within the Management Information Base (MIB). The command is the specific request issued to a device (GET, SET, etc.). Devices are uniquely identified within an MIB by an object identifier (OID). SNMPv3 introduces security and access control that was not present in SNMPv1 and SNMPv2. To integrate security the concept of a community was replaced with a context. A context represents a set of physical or logical devices in the
SNMP environment. To issue a command to a device, an SNMP client must have security access to the context containing the device. Access is determined by a context engine, which bases access control off of the SNMP client’s security name.

Due to the security changes between SNMPv2 and SNMPv3, three elements were added to the SNMP element: contextEngineID, contextName, securityName. Each of these elements is to be used only if the SNMP service is utilizing SNMPv3. Otherwise, the community element is used. Since IDMEF version 09 does not support SNMPv3, the upgrade tool does not populate the SNMPv3 elements within the SNMPService element when upgrading from IDMEF version 09 to IDMEF version 10. All other elements within the IDMEF version 09 SNMP element are moved to the IDMEF version 10 structure without change.

3.5.11. IDMEFv10 – IDMEFv11

3.5.11.1. Identical Structure

Between IDMEF version 10 and IDMEF version 11, there are no structural or logical differences in the format specifications. The only reason for a new version is that the previous version had expired. Therefore, a new identical version had to be created to maintain a current working version. Therefore, a new identical version had to be created to maintain a current working version. There are a few wording differences and many formatting changes between the specifications, however the information described by the specifications had not changed. As a result, the upgrade tool simply moves all data from the IDMEF version 10 structure to the IDMEF version 11 structure. There are no modifications to the data during this stage.
3.5.12. IDMEFv11 – IDMEFv12

3.5.12.1. Impact

In IDMEF version 12, the severity attribute of the Impact element adopts a new possible value—“info”. In IDMEF versions prior to version 12, there was no distinction between Alerts that are part of an attack and Alerts that are generated for informational purposes only. The latter of these two Alert types can be quite common in many systems. For many services (i.e. cron, rsync, etc.) events are detected and recorded as part of a logging process. When encoded into IDMEF, these events must be encapsulated within an Alert element. To show that the Alert is not part of an attack, but contains data from an informational event, the severity attribute in the Impact element is set to “info”. Although an important aspect of IDMEF version 12, the upgrade tool does not attempt to determine if an Alert is an informational event or not. Rather, the IDMEF version 11 severity attribute of the Impact element is transferred to the IDEMF version 12 structure with no change.

3.5.12.2. Analyzer

During the upgrade process from IDMEF version 11 to IDMEF version 12, two important changes to the Analyzer element should be noted. First, in IDMEF version 12 and later, the Analyzer element is allowed to contain an additional Analyzer element. By inserting an Analyzer element into an Analyzer element, a form of recursion is created allowing analyzers to indicate that they are passing the IDMEF-message along from another analyzer. This also allows the IDMEF-message structure to more accurately represent where a message originated within a hierarchical intrusion detection environment.
Second, the IDMEF version 12 specification allows for an Analyzer element to contain a name attribute to help identify the analyzer. Although, the name attribute does not overrule the analyzerid attribute in identifying an analyzer, it may allow for a human to more easily identify the analyzer. The analyzerid attribute for an analyzer must be unique within the intrusion detection environment. Therefore, if automatically generated, it may appear cryptic to a human managing the analyzer. However, with a name attribute, the analyzer may be easier to identify. It is important to note that the name attribute cannot be used to positively identify an analyzer with the environment. For true identification, the analyzerid attribute must be used. Since the name attribute in the Analyzer element is optional, the upgrade tool does not create a name attribute in the IDMEF version 12 structure from the information in the IDMEF version 11 structure—the results could be as cryptic as an auto-generated analyzerid and go against the premise for the name attribute's incorporation. Therefore, the name attribute is omitted from the IDMEF version 12 Analyzer structure during the upgrade process.

3.5.12.3. Classification

In IDMEF version 11 an Alert element contains one or more Classification elements. Each of these Classification elements is used to “name” the Alert and provide information about the vulnerability that the attack attempted to exploit. This information includes a name for the vulnerability, the originating library (i.e. CVE [1], Bugtraq [5], or vendor specific), and a URL where a manager can obtain more information about the vulnerability. As vulnerability libraries continue to grow, there will be greater amounts of overlap between libraries. For example, the teardrop denial of service attack is contained within both the bugtraq and CVE
libraries (bugtraq id: 124 and CVE name: CAN-1999-0015). When an analyzer identifies a teardrop denial of service attack, an Alert element can be created containing two Classification elements—one for the bugtraq library and one for the CVE library.

By allowing a single Alert element to contain multiple Classification elements referring to a single vulnerability creates a semantic contradiction. To resolve this problem, IDMEF version 12 requires that an Alert element is to contain a single Classification element. This Classification element contains a short string identifying the vulnerability, a unique identifier, and zero or more Reference elements. Each of these Reference elements contains the name, origin, and url information originally contained within the IDMEF version 11 Classification element with one small modification. The origin attribute of the IDMEF version 11 Classification element is allowed the following values: “unknown”, “bugtraqid”, “cve”, and “vendor-specific”. However, in IDMEF version 12, the origin attribute of the Reference element can have two additional values: “osvdb” and “user-specific”. These values state that the Reference refers to the Open Source Vulnerability Database [2] or is user specified, respectively. In addition, the Reference element can contain a meaning attribute that is used to further describe vendor and user specific vulnerability references.

The upgrade tool handles all of these differences in the following manner. Every Alert element in the IDMEF version 11 structure is examined for Classification elements. For every Classification element encountered, a Reference element is created within the Classification element of the corresponding IDMEF version 12 Alert element. The origin attribute and name and url elements from the IDMEF version 11 Classification element are
copied into the corresponding IDMEF version 12 Reference element with no change. Although, the Reference element does allow additional values for the origin attribute, it is impossible to derive them from the IDMEF version 11 structure. The meaning attribute of the Reference element is omitted from the IDMEF version 12 construct. This is done because the meaning of vendor specific vulnerabilities cannot be derived without that vendor’s library and the meaning element is not required to be present in the Reference element. The IDMEF version 12 Classification element for the corresponding Alert must have a text attribute. This text attribute is created by concatenating the names of every IDMEF version 11 Classification element’s name element, separated by a comma (i.e. Classification[0].name + “,” + Classification[1].name + “,” + … + Classification[n].name). A graphical description of this process can be found in Figure 3.10.

![Figure 3.10 - IDMEFv11 Classification to IDMEFv12 Reference Mapping](image-url)
3.5.12.4. Identification

Between IDMEF version 11 and IDMEF version 12, identification attributes within the Alert and Heartbeat elements changed, slightly. In IDMEF version 11, Alerts and Heartbeats are identified each identified by an ident attribute. This attribute uniquely identifies an Alert or Heartbeat send by a particular analyzer. Analyzer elements are uniquely identified by an analyzerid attribute and must utilize this attribute if the analyzer is to use the ident attribute to identify Alert and Heartbeat elements. This allows every Alert and Heartbeat element in the intrusion detection environment to be identified by an (analyzerid, ident) pair.

The (analyzerid, ident) pairing also uniquely identifies Address, File, Node, Process, Service, Source, Target, User, and UserID elements. However, the ident attribute for these elements has an additional restriction. Not only must the ident attribute unique from a particular analyzer, but the ident attribute for one of these elements must be unique for each combination of attributes and elements contained within that specific element. For example, Source element A contains a Node element with a category element equal to “unknown”. Source element B contains all the exact information as Source element A with one exception. The category element of the Node element within Source element B is omitted. Because of this slight difference, the ident attribute of Source elements A and B must be different.

To avoid confusion of requirements between the ident attribute of the Alert and Heartbeat elements and the more restrictive ident attribute of the Address, File, Node, Process, Service, Source, Target, User, and UserID elements, IDMEF version 12 specifies that a new messageid attribute is to contain identification information of the Alert and Heartbeat...
elements. The upgrade tool accounts for this difference between IDMEF version 11 and IDMEF version 12. If an Alert or Heartbeat element contains an ident attribute in the IDMEF version 11 structure, an equivalent messageid attribute is created in the IDMEF version 12 structure for the corresponding Alert or Heartbeat. If an Alert or Heartbeat in the IDMEF version 11 structure does not contain an ident attribute, then a messageid attribute is not created for the corresponding Alert or Heartbeat in the IDMEF version 12 structure.

![Figure 3.11 - IDMEFv11 to IDMEFv12 Identifiers](image)

3.5.12.5. File System

Between IDMEF version 11 and IDMEF version 12, file system information changed in two ways. The first is that the fstype attribute is no longer required to be present within the File element. This modification occurred because some analyzers have difficulty determining the file system being used by a system. [11] This difference requires little action to be taken by the upgrade tool. If a File element contains an fstype attribute, that attribute is created in the corresponding IDMEF structure. If no fstype is present, the attribute is not created.

Although IDMEF version 11 requires an fstype attribute to exist within the File element, the element may not exist. During the upgrade from IDMEF version 09 to IDMEF version 10,
the fstype attribute changed from a free-form string to an attribute with a specific set of possible values. In the event that the IDMEF version 09 string did not match any value in the IDMEF version 10 set, the attribute was omitted, even though this violated the IDMEF version 10 structure. This is allowed because, during the upgrade process from IDMEF version 11 to IDMEF version 12, the omitted attribute becomes legal once again.

The second change to file system information between IDMEF versions 11 and IDMEF version 12 is the incorporation of checksum information. Checksum information is regularly used to verify file integrity. If a file does not pass a checksum test then the file has been modified from its original form and may be involved in some form of attack. To include this information in the IDMEF structure, IDMEF version 12 contains a new Checksum element containing the checksum value, the algorithm used to determine the checksum value, and the key used by the algorithm (if appropriate). This information is held within a value element, algorithm attribute, and key element respectively. To attach checksum information to an individual file, each File element in the IDMEF structure contains zero or more Checksum elements. Each checksum value is dependent upon the algorithm and key used. Therefore, a Checksum element should be created for every algorithm/key combination used. Since there is no checksum information in the IDMEF version 11 structure and IDMEF version 12 File elements do not require checksum information to be present, the upgrade tool does not create any Checksum elements when upgrading from IDMEF version 11 to IDMEF version 12.
3.5.12.6. Internet Protocol

The IDMEF version 12 specification takes advantage of increasing internet protocol standardization. The Internet Assigned Numbers Authority (IANA) is largely responsible for this standardization. This organization has published and maintains a list of internet protocol names and numbers. [3, 4] By using these standardized lists, analyzers can accurately communicate a service’s protocol to another analyzer without discrepancy. The IDMEF version 12 specification incorporates the IANA’s protocol standardizations by adding two optional attributes to the Service element: iana_protocol_name and iana_protocol_number. Either or both of these attributes can be used when an analyzer identifies a Service’s protocol within the IANA lists. If an analyzer cannot specify an IANA protocol name or number then the protocol is identified by a port or list of ports. To upgrade from IDMEF version 11 to IDMEF version 12, the upgrade tool omits the iana_protocol_name and iana_protocol_number from the IDMEF version 12 structure. Although it is possible for the upgrade tool to look up a service’s protocol within the IANA lists by using the Service element’s name element, the upgrade tool does not perform this action. If this feature were integrated into the tool, then the upgrade tool would require additional maintenance as the IANA’s lists are updated.

In addition to the IANA protocol names and numbers, the IDMEF version 12 Service element is expanded to specify an internet protocol version being used by the service. A new optional attribute named ip_version is included in the IDMEF version 12 specification. This attribute is to contain a single integer specifying the internet protocol being used (i.e. 4 or 6). As this attribute is not required according to the IDMEF version 12 specification and there is
no method to derive which protocol version is being used, the upgrade tool omits the attribute when upgrading from IDMEF version 11 to IDMEF version 12.

3.5.13. IDMEFv12 – IDMEFv13

3.5.13.1. Additional Data

In IDMEF version 13, the AdditionalData element was expanded from its previous specification in IDMEF version 12. Specifically, the type attribute was expanded to include “byte-string” as a legal value. With the incorporation of this additional value, the AdditionalData element could indicate that it contains binary string (i.e. byte[] ). Prior to this expansion, all binary data longer than one byte would be marked as a “string” by the type attribute of the AdditionalData element. Although an important change in the IDMEF specification, this difference plays little role in the upgrade process. The upgrade tool copies the type attribute from IDMEF version 12 to IDMEF version 13 with no modification.

3.5.13.2. File System

Prior to IDMEF version 13, all of a Target’s file information was contained within a FileList element. Since its introduction, the FileList element is simply a container for an array of File elements, nothing more. The IDMEF version 13 specification removed the FileList element from the Target element and replaced it with the array of Files originally contained within the FileList element. Since the FileList contained no additional information, the exchange of a FileList element for an array of File elements provides one less layer of data with no information loss. To incorporate this change from IDMEF version 12 to IDMEF version 13, the upgrade tool extracts the array of File elements from an IDMEF version 12 Target
element’s FileList element and inserts that array directly into the corresponding IDMEF version 13 Target element.

In addition to the removal of the FileList element, the IDMEF version 13 specification also permits Multipurpose Internet Mail Extensions (MIME) type information, as described by [14], to be associated with each File element. This information is stored as a free-form string in a new file-type attribute of the File element. According to the IDMEF version 13 specification, the file-type attribute is not required to be present in a File element. Therefore, the upgrade tool creates an IDMEF version 13 element using the data stored in the IDMEF version 12 File element, omitting file-type information from the IDMEF version 13 File element.

Figure 3.12 - IDMEFv12 to IDMEFv13 Target File Mapping
3.5.13.3. Heartbeat

Throughout all IDMEF versions, the Heartbeat element is used as an indicator of the health of an analyzer. The intended use of a Heartbeat element is that an analyzer will send a message containing a Heartbeat element to a manager at regular intervals (every 15 minutes, every 2 hours, etc.). If a manager does not receive a message containing a heartbeat element, it can assume that the analyzer is offline. Prior to IDMEF version 13, the manager was required to determine the time period that must elapse before assuming that an analyzer is offline. There is no built-in means for an analyzer to convey how often it is sending heartbeats to the analyzer. Starting in IDMEF version 13, a heartbeatInterval element was added to the Heartbeat element. This new heartbeatInterval element is to contain the number of seconds between consecutive heartbeats. Although, not required to exist, the heartbeatInterval element finally allows for an analyzer to inform a manager as to when the next heartbeat will be sent. The upgrade tool recognizes the significance of the new heartbeatInterval and allows it to exist in the IDMEF version 13 structure. However, since the element may be absent from the structure and the IDMEF version 12 structure does not contain an equivalent data structure, the heartbeatInterval element is omitted from the IDMEF version 13 structure when upgrading from IDMEF version 12.

3.5.13.4. Terminal

Unlike all previous versions, the IDMEF version 13 specification allows for the incorporation of terminal information within the IDMEF structure. This information is stored in the new tty attribute of the UserID element. This attribute should contain a string representing the terminal a user (human or process) was operating through when the Alert
was generated. Since the UserID element is utilized within the FileAccess to represent the user or group to which specific file attribute apply, the tty attribute must be optional; terminal information is unrelated to file access permissions. To account for the influx of terminal data from IDMEF version 12 to IDMEF version 13, the upgrade tool must omit the tty attribute from all UserID elements in the IDMEF version 13 structure. This is a legal option because the tty attribute does not exist in IDMEF version 12 and the value of tty attribute cannot be determined by the IDMEF version 12 structure.

3.5.14. IDMEFv13 – IDMEFv14

3.5.14.1. Identical Structure

Between IDMEF version 13 and IDMEF version 14, there are no structural or logical differences in the format specifications. The only reason for a new version is that the previous version contained a few minor errors. These errors were corrected in IDMEF version 14 along with a few formatting changes. However, the information conveyed in both versions is identical. As a result, the upgrade tool simply moves all data from the IDMEF version 13 structure to the IDMEF version 14 structure. There are no modifications to the data during this stage.
4. Database Import Tool

XML documents are an efficient means of storing information and are becoming the de facto standard for data exchange over the internet. However, XML documents have been shown to provide poor performance with regards to performing queries upon the data they contain. [22] Much of this performance hit is due to necessary parsing of the document for every query. Relational Database Management Systems (RDBMSs) do not have the same parsing needs as XML files do and, therefore, have increased performance when performing queries upon stored data.

In distributed intrusion detection systems information is typically gathered from several sources then sent and analyzed at centralized locations. This information quickly stockpiles and efficient queries must be made to analyze the data. Therefore, we have created a database import tool, which accepts IDMEF version 14 XML files and stores the information in a designated RDBMS.

4.1. System Design

The database import tool was designed to be portable, extensible, and provide an efficient mapping from the treelike IDMEF XML structure to the flat relational database structure. To accomplish this, the tool is built around four major components, as shown in Figure 4.1. The Database Connection component manages all transactions between the application and the database via an external JDBC Driver. By using Java and a JDBC driver, the application can
interact on multiple platforms and with any database with a JDBC driver. The JDBC Driver must be supplied by the user and must be on a valid CLASSPATH.

The database import tool loads IDMEF version 14 XML data into a RDBMS database in three distinct stages. First, the application utilizes information provided by the user to connect to a RDMBS via a specified JDBC driver. Second, the database import tool examines the database, verifying that all necessary tables have been created. Third, the tool loads the information contained within the IDMEF version 14 XML file into the specifically mapped table structures.

![Database Import Tool Design](image)

**Figure 4.1** – Database Import Tool Design
4.2. Database Connection Configuration

The Database Connection through the JDBC driver is configured via a Database Connection Setup Screen shown in Figure 4.2. This screen allows users to designate a JDBC Driver, Database URL, Database Name, Username, and Password. The JDBC Driver is the driver the user wishes to use to connect to the database. The Database URL line allows specification of the server managing the database connection. By default the Database URL is populated with localhost designating that the server is located on the machine running the database import tool. As database servers typically contain multiple databases, the Database Name field allows the user to specify which database on the server they wish to connect to. The database import tool assumes that each database is an individual intrusion detection environment. Therefore, data from different environments should not be imported into the same database. The Username and Password fields provide authentication information to the designated database. The username provided must be able to create tables in the database and perform insert, select, and modify SQL statements on the created tables.

![Database Connection Setup Screen](image)

**Figure 4.2** – Database Connection Setup Screen
4.3. XML to RDBMS Mapping

The database import tool is designed to construct database tables based around the IDMEF version 14 specification and populate those tables using an IDMEF version 14 XML file.

Many different strategies have been developed to map XML documents to a RDBMS schema. [7, 8, 12, 19, 24, 25] However as described by [8] it is unlikely that any automated XML to RDBMS will be ideal for all applications. Therefore, the database tool contains a custom IDMEF XML to RDBMS mapping. This mapping and resulting schema was designed to be extensible as well as efficient.

To create an extensible and efficient schema, the following rules were adopted with regards to creating relations.

1. All elements uniquely identified within the intrusion detection environment must map to an individual relation.
2. Uniquely identifying attributes exist within all relations for uniquely identifiable elements.
3. All attributes of an element are attributes of that element’s relation.
4. If an element is encapsulated by only one parent element and is not uniquely identifiable, then the attributes of the child element are attributes of the parent element’s relation and the child element’s relation is eliminated.
5. All element relations contain an internal identifier attribute (Internal_id) to be used as a primary key. This attribute is not related to the information contained within the relation.
6. Exact reconstruction of an IDMEF XML document is not required as long as information is not lost.

7. Subclass elements exist within a separate relation from their parent element.

8. Many-to-many relationships from element E to element F are contained within a separate relation.

9. One-to-many relationships from element E to element F are established by inserting the primary key for element E’s relation into the relation for element F.

10. If there is a one-to-many relationship from element E to element G and a one-to-many relationship from element F to element G and element G is not uniquely identifiable, separate relations are created for each relationship (i.e. between E and G and between F and G).

Rules 1 and 2 were created to speed the identification process when an analyzer uniquely identifies an existing element. By separating identifiable elements into individual relations, information insertion and extraction regarding those elements only need to operate on a single table. Additionally, by required the analyzerid to be present along with the identifier attribute in the relation, there is no need to join the identified element’s relation with the analyzer’s relation to uniquely identify the element. By reducing identification overhead, performance should be increased.

Rule 3 constructs the basis for all element relations in the RDBMS schema. By importing an element’s possible attributes into the element’s relation all information regarding the element can be represented within the relation.
Rule 4 allows relations within the evolving database schema to be condensed. By incorporating a child element into its parent element’s relation, there is no longer a need to join the parent and child elements’ relations to extract IDMEF information. To avoid violating rule 1, the child element must not be uniquely identifiable. Also, this process can only be done if there is at most one of the child elements within the parent element. For example, the IDMEF version 14 Alert element contains at most one AnalyzerTime element. Since the AnalyzerTime element is not uniquely identifiable, the AnalyzerTime relation attributes are incorporated into the Alert relation and the AnalyzerTime relation is eliminated. This allows AnalyzerTime information to be available without performing a join between two relations. Thereby performance is increased.

Rule 5 creates a common primary key for all relations based upon an IDMEF element. Every relation established by the database import tool contains one attribute named Internal_id. The value of the Internal_id attribute is automatically established by incrementing the Internal_id attribute for every element record inserted into the relation. Some RDBMSs support autoincrementation, which would allow the RDBMS to automatically create the Internal_id value without intervention from the database import tool. However, not all RDBMSs support autoincrementation. Therefore, in the interest of portability, the Internal_id value is incremented by the database import tool. It is important to note that the database import tool does not perform any database locking or synchronization and, therefore, multiple instances of the database import tool cannot interact with the same database simultaneously.
Rule 6 states that after an IDMEF XML document is imported into the database, the database does not need to exactly reflect the contents of the XML document. In many XML to RDBMS mapping algorithms [7, 8, 12, 19, 24, 25] XML documents used as input can be perfectly recreated according to the information stored in the database. Those algorithms that cannot exactly recreate an input XML document are often considered lossy [7]. The database import tool does not need the ability to recreate an input XML document for two reasons. Firstly, in the IDMEF structure, the order of elements does not hold any relevance to the information contained within the structure. Therefore, ordering of elements is ignored by the database import tool. Secondly, some elements within the IDMEF structure can be uniquely identified and referenced. In these situations an analyzer may not provide all the information about an element, but rather, it may simply reference an element. When this occurs the referenced element is queried using the IDMEF unique identifier and reference using the internal identifier (Internal_id) is used. Thus, the referenced element contains at least as much information as provided by the IDMEF XML document.

Rule 7 describes a fairly standard strategy used when converting entity relation diagrams (ERD) to relations. One relation is created using all the attributes for the superclass. Additional relations are created for each subclass, containing attribute applicable to that subclass and using the same primary key as the superclass. Another two common practices involve combining all subclass and superclass attributes into a single large relation and creating relations for every subclass-superclass combination. Considering that Overflow, Tool, and Correlation Alerts are only a small subset of all Alerts described by IDMEF, the single relation technique was rejected. As many attributes in this single large relation would
be blank, there will be much wasted space in the database, depending on the RDMBS storage and compression policies. If relations were created for all subclass-superclass combinations, there would be an equal number of joins between tables to extract all information from an IDMEF alert. There is no performance or storage savings over creating separate subclass and superclass relations. However, by creating separate subclass and superclass relations, the relational database schema more closely represents the treelike IDMEF XML structure and, thus, will be more extensible.

Rule 8 designates how a many-to-many relationship should be represented within the relational schema. In the event that a single IDMEF element contains multiple elements of the same type and a single element can be contained by more than one element, there exists a many-to-many relationship between the elements. Therefore, there should be a separate relation for each element type, containing the information about each element. To create the many-to-many relationship between elements, a third relation is created containing the primary keys of each element. For example, a single Alert element can contain multiple Source elements and a single Source element can be contained by multiple Alert elements. Therefore, a many-to-many relationship exists between Alerts and Sources. To represent this structure in the RDBMS schema, two relations are created—one for the Alert elements and another for the Source elements. To represent the many-to-many relationship between Alerts and Sources, an Alert_Source relation is created containing two attributes—the Internal_id for an Alert element and the Internal_id for a Source element. This is a fairly common technique when developing RDBMS schemas.
Rules 9 and 10 describe how one-to-many relationships are established in the relational database schema. According to rule 9, if an IDMEF element contains multiple elements of the same type, there will be two relations created—one for each element. In this structure, the child element is identified as being contained within a parent element by including the parent element’s primary key (Internal_id) as an attribute in the child element’s relation. For example, an IDMEF-Message element can contain multiple Alert elements and an Alert element can only be contained within one IDMEF-Message element. This one-to-many relationship from IDMEF-Message to Alert is established by creating separate relations for IDMEF-Message and Alert. Within the Alert relation, there is an attribute named Message_id. This attribute identifies the parent IDMEF-Message for every child Alert element entered in the Alert relation.

Rule 10 handles a specific situation where two different elements have a one-to-many relationship to a third type of element. In this situation, a relation is created for each of the two parent elements and two relations are created for the single child element. Each of the child element relations contains the primary key to one of the two parent elements. For example, in the IDMEF specification, both an Alert element and a Heartbeat element can contain multiple AdditionalData elements. This situation creates two one-to-many relationships—from Alert to AdditionalData and from Heartbeat to AdditionalData. According to rule 9, relations are created for all three elements and the AdditionalData relation will contain keys referencing the Alert and AdditionalData relations. However, since an AdditionalData element cannot be contained within both an Alert element and a Heartbeat element, one of the two keys will always be blank. This can create wasted space depending
on the database’s storage and compression policies. To alleviate this problem, rule 10 was created. Under this rule four relations are created instead of three—one Alert relation, one Heartbeat relation, and two AdditionalData relations. One Addition Data relation will correlate with the Alert relation and the other with the Heartbeat relation via the technique described by rule 9.

Then enactment of these rules creates a relation database schema consisting of 39 relations. The database tables derived from this schema can be found in Appendix B. When the database import tool creates these tables, they all contain “IDMEF_” as a prefix to distinguish them from other database tables that may exist in the IDS.

4.3.1. Normalization

It is also important to note that all 39 relations are in Third Normal Form (3NF). A relation R is in 3NF if an only if, for every functional dependency \( A_1 A_2 \ldots A_n \rightarrow B \) over R:

- \( B \in A_1 A_2 \ldots A_n \) or
- \( A_1 A_2 \ldots A_n \) is a superkey for R or
- B is a member of a key

By ensuring that all the database tables are in 3NF, unnecessary redundancy is eliminated and the database uses less storage space on the server. Also, depending on the RDBMS storage structure, I/O performance is increased as individual records use less disk space. Finally, the structure is very similar to the IDMEF XML version 14 structure, allowing the
database structure to be extensible. Foregoing a major restructuring of the IDMEF structure, future versions will only require incremental modifications to the database structure.

4.4. Table Population

Once the database connection is configured and all necessary tables are constructed, the database import tool loads the information from the specified input file and stores it in the RDBMS. As the creation of the RDBMS tables provided a clear map the data migration, the process of loading data from the IDMEF version 14 XML file and storing it in the specified RDBMS structures is quite a simple matter. First, data is read from the specified IDMEF version 14 XML file into a corresponding IDMEF structure. This is the same structure used within the upgrade tool. Once the data is loaded into the common structure, it is inserted into the RDBMS via JDBC brokered SQL statements. This process is governed by the mapping procedure created in section 4.3.
5. Conclusions

To compare data migration techniques between formats we constructed Digital Rosetta Stones for different versions of a single evolved format and for two formats not directly linked by an evolutionary progression. These Digital Rosetta Stones resulted in two different processes for data migration: sequential migration and direct migration. Within the sequential migration process, three techniques (data elimination, external retention, and specification transgression) were utilized during situations where data could not be mapped to the next version. The results of this study provide interesting insight as how to create a Digital Rosetta Stone and formats should be evolved to simplify the creation of a Digital Rosetta Stone.

5.1. Digital Rosetta Stone Creation

First, when creating a Digital Rosetta Stone to migrate information between different versions of a single evolutionary format, a sequential data migration process should be utilized. The sequential process will minimize maintenance needs as new versions of the format emerge. However, if the number of versions is small and the format will not evolve beyond its current stage or performance of the process is an issue, a direct migration process should be considered as there are fewer actions to perform.

Second, when migrating data between formats not linked by a direct evolutionary progression, the direct data migration process should be utilized. This technique will provide a single mapping that needs to be maintained as each format evolves. This single mapping
should be the only means of migration between the formats. If data needs to be migrated to or from any version other than the mapped versions, a sequential migration process should be used in conjunction with the direct migration process. The sequential migration process handles information within a single evolutionary format, while the direct migration process handles the information between the unrelated formats. In this scenario two Digital Rosetta Stones are joined together to process the data.

Third, when developing mappings between versions in a sequential data migration process, external retention techniques should be utilized over specification transgression wherever possible. When utilizing specification transgression, structures are created that violate documented specifications. Although, these violations are resolved before the process completes, they inherently restrict the process to only output formats before or after the specification transgression process. The illegal structures created during specification transgression cannot be converted into their final form as the information contained within the structures is invalid. A true complete and effective Digital Rosetta Stone for an evolved format should be able to translate between any two versions of a single evolved format. Specification transgression prevents this from occurring without requiring additional processing and complexity. External retention does not require additional processing as the data contained within the structure is valid.

5.2. Directed Format Evolution

When developing a data storage format that is expected to evolve, there are some techniques that should be used to aid in data translation between versions. The first technique is to
maintain detailed documentation for each version of the format. According to Rothenberg [23], if the behavior of a system that views the document can be adequately described, then future generations will be able to recreate the system and view the document. By thoroughly documenting the format itself, all systems that can view the document are described. This documentation should include all relevant syntax and semantics of the format. The IDWG does not maintain documentation for each IDMEF format. However, every released IDMEF internet draft is archived at http://www.watersprings.org. Within the IDMEF specifications, every element, attribute, and data type is thoroughly described. Thereby, mappings between versions could be created.

A second technique is to avoid major reconstructions within the format and terminology used to describe the format. By evolving through incremental changes in format and description, differences and mappings between versions can be more easily determined. If major modifications to format or description is required, the other should remain as static as possible. That is, if format changes then description should remain constant. Similarly, if the description changes then the format should remain constant. When both format and description change migration paths between versions are very difficult or impossible to create.
6. Future Work

Currently, the upgrade tool does not provide a complete Digital Rosetta Stone. It uses any version of IDMEF XML file as input, processes the data contained within the XML file through structures representing consecutive IDMEF versions and outputs an IDMEF version 14 XML file. However, if an XML file pertaining to a specific IDMEF version other than IDMEF version 14 is needed, the upgrade tool will not suffice. For example, assume a DIDS contains a single analyzer and a single manager. The analyzer can only output IDMEF XML as designated by version 12 and the manager can only input IDMEF XML as specified by version 08. The upgrade tool would not be able to accommodate this situation. To address this shortfall, additional work on the upgrade tool needs to be completed, allowing it to accept any version of IDMEF XML and output any other version of IDMEF XML. This could be done addressing the mapped out differences between IDMEF version and allowing the tool to not only perform upgrades, but downgrades as well. Structures simulating every IDMEF version are already present within the upgrade tool. Therefore, it is possible to use these structures to create XML files for any IDMEF version.

Similarly, the database import tool only stores IDMEF version 14 XML data into a RDBMS. A complementary database extraction tool could be created allowing data within the database to be extracted and saved in an IDMEF version 14 XML document. This tool would be very beneficial as new IDMEF specifications are released. When a new IDMEF version is released, the information within the RDBMS can be extracted, then upgraded via an upgrade tool, which includes the new IDMEF specification. After the data is upgraded, it can be translated back to RDBMS using mappings from the new IDMEF version.
7. References


Appendix A - RDBMS tables

Table `Message` {
    Internal_id int;
    Version float;
}

Table `Alert` {
    Message_id int;
    Internal_id int;
    Analyzerid Sting;
    Messageid String;
    Analyzer_id int;
    Classification_id int;
    CreateTime_datetime timestamp;
    CreateTime_ntpstamp String;
    DetectTime_datetime timestamp;
    DetectTime_ntpstamp String;
    AnalyzerTime_datetime timestamp;
    AnalyzerTime_ntpstamp String;
    Confidence_rating String;
    Confidence_numericRating Float;
    Impact_completion String;
    Impact_severity String;
    Impact_type String;
}

Table `Heartbeat` {
    Message_id int;
    Internal_id int;
    Analyzerid String;
    Messageid String;
    Analyzer_id int;
    CreateTime_datetime timestamp;
    CreateTime_ntpstamp String;
    AnalyzerTime_datetime timestamp;
    AnalyzerTime_ntpstamp String;
    heartbeatInterval String;
}

Table `Alert_Source` {
    Alert_id int;
    Source_id int;
}


Table Alert_Target {
    Alert_id int;
    Target_id int;
}

Table ToolAlerts {
    Internal_id int;
    Alert_id int;
    Name String;
    Command String;
}

Table ToolAlert_Alert {
    ToolAlert_id int;
    Alert_alertident String;
    Alert_analyzerid String;
    Alert_id int;
}

Table OverflowAlert {
    Internal_id int;
    Alert_id int;
    Program String;
    Size int;
    Buffer String;
}

Table CorrelationAlert {
    Internal_id int;
    Alert_id int;
    Name String;
}

Table CorrelationAlert_Alert {
    CorrelationAlert_id int;
    Alertident String;
    Analyzerid String;
    Alert_id int;
}

Table Alert_AdditionalData {
    Alert_id int;
    Type String;
    Meaning String;
    Data String;
}
Table Heartbeat_AdditionalData {
    Heartbeat_id int;
    Type String;
    Meaning String;
    Data String;
}

Table Alert_Action {
    Internal_id int;
    Alert_id int;
    Action_category String;
    Action_description String;
}

Table Analyzers {
    Internal_id int;
    Analyzerid String;
    Name String;
    Manufacturer String;
    Model String;
    Version String;
    Class String;
    Ostype string;
    Osversion string;
    Node_id int;
    Process_id int;
    Analyzer_id int;
}

Table Source {
    Internal_id int;
    Analyzerid String;
    Ident String;
    Spoofed String;
    Interface String;
    Node_id int;
    User_id int;
    Process_id int;
    Service_id int;
}

Table Node {
    Internal_id int;
    Analyzerid String;
}
Ident String;
Category String;
Name string;
Location string;

} Table Node_Address {
  Node_id int;
  Address_id int;
}

Table Address {
  Internal_id int
  Analyzerid string;
  Ident string;
  Category String;
  Vlan_name String;
  Vlan_num int;
  Address String;
  Netmask String;
}

Table User {
  Internal_id int;
  Analyzerid String;
  Ident String;
  Category String;
}

Table User_UserID {
  User_id int;
  UserID_id int;
}

Table UserID {
  Internal_id int;
  Analyzerid String;
  Ident String;
  Name String;
  Number int;
  Type String;
  Tty String;
}

Table Process {
  Internal_id int;

Analyzerid String;
Ident String;
Pid int;
Name string;
Path String;
}

Table Process_Arg {
    Process_id int;
    Arg String;
}

Table Process_Env {
    Process_id int;
    Env String;
}

Table Service {
    Internal_id int;
    Analyzerid String;
    Ident String;
    Ip_version int;
    Iana_protocol_number int;
    Iana_protocol_name String;
    Name String;
    Protocol String;
    Portlist_id int;
    Port int;
}

Table SNMPService {
    Service_id int;
    Oid string;
    Community String;
    Command String;
    SecurityName String;
    ContextName String;
    ContextEngineID String;
}

Table WebService {
    Service_id int;
    url string;
    cgi string;
    http_method string;
}
Table WebService_Arg {
    Service_id int;
    Arg String;
}

Table Ports {
    Portlist_id int;
    Port_number int;
}

Table Portlist {
    Portlist_id int;
    Start_port_number int;
    End_port_number int;
}

Table Target {
    Internal_id int;
    Analyzerid String;
    Ident String;
    Decoy String;
    Interface String;
    Node_id int;
    User_id int;
    Process_id int;
    Service_id int;
}

Table Target_File {
    Target_id int;
    File_id int;
}

Table Classification {
    Internal_id int;
    Analyzerid String;
    Ident String;
    Text String;
}

Table References {
    Classification_id int;
    Name String;
    Url String;
    Origin String;
}
Table File 
{
    Internal_id int;
    Analyzerid String;
    Ident Sting;
    Name String;
    Path String;
    Create_time timestamp;
    Modify_time timestamp;
    Access_time timestamp;
    Data_size int;
    Disk_size int;
    Inode_c_major_device int;
    Inode_c_minor_device int;
    Inode_changetime timestamp;
    Inode_major_device int;
    Inode_minor_device int;
    Inode_number int;
    Checksum_algorithm String;
    Checksum_key String;
    Checksum_value String;
    Category String;
    Fstype String;
    File_type string;
}

Table File_FileAccess 
{
    File_id int;
    Internal_id int;
    UserID_id int;
}

Table FileAccess_Permissions 
{
    FileAccess_id int;
    Permission String;
}

Table File_Linkage 
{
    File_id int;
    Name String;
    Path String;
    Category String;
    File int;
}