**AFRL-IF-RS-TR-2005-353**
**Final Technical Report**
**October 2005**


# INTEGRATION INTO CYBER SECURITY MANAGEMENT SYSTEM

**Northrop Grumman Mission Systems**

*APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.*

**AIR FORCE RESEARCH LABORATORY**
**INFORMATION DIRECTORATE**
**ROME RESEARCH SITE**
**ROME, NEW YORK**

**STINFO FINAL REPORT**

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2005-353 has been reviewed and is approved for publication

APPROVED: /s/

BRIAN T. SPINK
Project Engineer

FOR THE DIRECTOR: /s/

WARREN H. DEBANY, JR., Technical Advisor
Information Grid Division
Information Directorate

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 074-0188*



| 1. AGENCY USE ONLY (Leave blank) | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | OCTOBER 2005 | Final May 04 – May 05 |

**4. TITLE AND SUBTITLE**
INTEGRATION INTO CYBER SECURITY MANAGEMENT SYSTEM

**5. FUNDING NUMBERS**
C - F30602-03-D-0026/0017
PE - 33140F
PR - WISE
TA - QP
WU - 17

**6. AUTHOR(S)**
Tom Daley

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
Northrop Grumman Mission Systems
Beeches Technical Campus
Route 26 North
Rome New York 13440

**8. PERFORMING ORGANIZATION REPORT NUMBER**
N/A

**9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
Air Force Research Laboratory/IFGA
525 Brooks Road
Rome New York 13441-4505

**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
AFRL-IF-RS-TR-2005-353

**11. SUPPLEMENTARY NOTES**

AFRL Project Engineer: Brian T. Spink/IFGA/(315) 330-7596/ Brian.Spink@rl.af.mil

**12a. DISTRIBUTION / AVAILABILITY STATEMENT**
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** ***(Maximum 200 Words)***

The purpose of this task was to integrate technology developed as part of the World Infrastructure Security Environment (WISE) program by Orincon Information Assurance into the Air Force Research Laboratory (AFRL) Cyber Security Management System (CSMS). The intended capabilities of WISE are to monitor host activity in real-time, alert analysts about information attacks utilizing a cost-benefit model, and recommend information-based countermeasures. Collection of WISE data into CSMS would enable quicker and more accurate response to threats against enterprise networks. However, at the time that this system integration work was performed, WISE and CSMS were not available. Therefore, since Orincon's Distributed Agent Information Watch (DAIWatch) is the precursor to WISE and AFRL's Air Force Enterprise Defense system (AFED) is the forerunner to CSMS, it was decided to enhance the method of data transfer from DAIWatch to AFED, which, at some future point in time, could be adapted for use by the WISE and CSMS programs, with a modicum of effort.

**14. SUBJECT TERMS**
DAIWatch, eXtensible Markup Language, XML, Transmission Control Protocol/Internet Control Protocol, TCP/IP

**15. NUMBER OF PAGES**
31

**16. PRICE CODE**

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| UNCLASSIFIED | UNCLASSIFIED | UNCLASSIFIED | UL |

## Table of Contents

**SUMMARY** ..... **1**

**1. INTRODUCTION** ..... **3**

**2. METHODS, ASSUMPTIONS, AND PROCEDURES** ..... **5**

**3. RESULTS AND DISCUSSION** ..... **7**

**4. LESSONS LEARNED** ..... **9**

**5. CONCLUSIONS** ..... **10**

**6. RECOMMENDATIONS** ..... **11**

**APPENDIX A - INSTALLATION AND MAINTENANCE GUIDE** ..... **12**

**APPENDIX B - SOFTWARE USER'S MANUAL** ..... **16**

## SUMMARY

A Java and XML-based interface was developed which provides the means for integration of host Intrusion Detection System (IDS) events from the Oracle database of the Distributed Agent Information Watch (DAIWatch) into the Oracle database populated by the Air Force Enterprise Defense system (AFED).

The data is sent from one system to another over a Transmission Control Protocol / Internet Protocol (TCP/IP) socket in the form of eXtensible Markup Language (XML) objects that conform to the Internet Engineering Task Force (IETF) Intrusion Detection Message Exchange Format (IDMEF). Security of the data transmission is assured by a Java Cryptography Extension (JCE) implementation of the triple Data Encryption Standard (DES) process.

Files in W3C XML Document Object Model (DOM) format are transformed into IDMEF XML objects, and vice-versa, via specific eXtensible Style Language (XSL) files and are validated against specific Document Type Definition (DTD) files. HIDS events are passed between the interface and database, using Java Database Connectivity (JDBC), Structured Query Language (SQL) statements and the Oracle XML SQL Utility (XSU) Application Programming Interface (API).

The purpose of this task was to integrate technology developed as part of the World Infrastructure Security Environment (WISE) program by Orincon Information Assurance into the Air Force Research Laboratory (AFRL) Cyber Security Management System (CSMS). The intended capabilities of WISE are to monitor host activity in real-time, alert analysts about information attacks utilizing a cost-benefit model, and recommend information-based countermeasures. Collection of WISE data into CSMS would enable quicker and more accurate response to threats against enterprise networks. However, at the time that this system integration work was performed, WISE and CSMS were not available. Therefore, since Orincon's DAIWatch is the precursor to WISE and AFRL's AFED is the forerunner to CSMS, it was decided to enhance the existing method of data transfer from DAIWatch to AFED, which, at some future point in time, could be adapted for use by the WISE and CSMS programs, with a modicum of effort.

During the early stages of a previous related development task, the foundation and framework of the design of an interface between DAIWatch and AFED was verbalized in meetings and through email correspondence between representatives of AFRL, Orincon, and Northrop Grumman. This initial design was later written into a draft version of an overview Interface Design Document (IDD), by Orincon. Non-applicable functionality was stripped out of the previous task's source code, to produce a stream-lined interface that transmits DAIWatch event data to AFED, on a periodic basis.

The design approach taken by the sole interface developer was one of getting a code module to perform a specific function, modify it and tweak it, then add more pieces of code in order to handle additional functionality. Since both DAIWatch and AFED have Oracle databases and the interface code was slated to be Java-based; it made sense to the interface developer to use Oracle XML APIs for Java.

Utilization of XSL and DTD files, by the interface, allows for minimal source code updates, when database tables and fields are modified or a new IDMEF version is issued, thereby reducing the need for re-compilation. Also, implementing the XML-based IDMEF as the common data transmission medium allows for inter-operation with other IDMEF-compliant systems and provides for an easier transition into compatibility with XML databases. Most importantly, the aggregate of the DAIWatch data integrated with the AFED data can be correlated and consolidated to reduce the information overload of network security personnel.

## 1. INTRODUCTION

Information Warfare (IW) attacks against complex distributed information systems have been increasing steadily over the last several years. These hostile actions have become possible as a result of evolving networks environments: advances in hardware, software, and communications technology; as well as high accessibility to these environments and their technologies. However, it has always remained imperative that system users can access and obtain information as needed and that the information remains free from corruption and theft. To maintain user confidence in this information, continuing upgrade and development has been necessary to ensure the integrity and availability of information that resides on and is transmitted among these systems. Such resulting technology integrates network security, adaptability, and survivability into existing and evolving architectures and systems at minimal impact to system performance.

Deployed Department of Defense (DOD) systems are increasingly complex aggregates of systems, networks, and infrastructure. Likewise, the DOD is increasing its dependence on Information Technology (IT) in all aspects of its operations. It is also holding down costs by relying on Government-Off-The-Shelf (GOTS) and Commercial-Off-The-Shelf (COTS) products and services. These two key trends result in increased dependencies on operational items the DOD does not own or develop which introduces additional vulnerabilities and risk to programs it cannot control.

The Automated Intrusion Detection Environment (AIDE) Advanced Concepts Technology Demonstration (ACTD) has developed a solution using current and maturing IW technology and has developed a multi-tiered integration environment. This solution combined current IW technology in an improved architecture for automated threat detection. Current capabilities include: correlation and user display of sensor data over regions of deployed systems at local agency, Commander-In-Chief (CINC), and global command levels. The automated warning capability of the IW threat would then be available to not only the war fighter, but also the joint commander of DOD operations, as well as Law Enforcement, Operations and Intelligence. AFED builds on AIDE's core intrusion detection capability and adds security policy enforcement, network configuration management, and vulnerability assessment capabilities. AFED is an integral piece of the IDS portion of the perimeter defense component of the AFRL Cyber Security Management System.

DAIWatch is a program supported by AFRL that applies distributed intelligent software agents to create mobile distributed intrusion detection and response systems to protect against sophisticated information attacks. The intelligent software agents perform tasks assigned to them to achieve specified goals. Intelligent agents are capable of modifying their behavior as dictated by roles assigned to them in performing a given task. These agents can also communicate with other agents to form communities of cooperating agents. Intelligent agents can be made mobile, resulting in distributed communities of intelligent agents to solve a given problem. This distributed intelligent agent construct employs automated reasoning technologies distributed over computer networks locally or around the world. The intelligent agents can analyze message context for network traffic, fuse diverse inputs, resolve conflicting information, detect

coordinated distributed assaults, and adapt to new types of network incursions. The agents can also provide information that determines when there is normal traffic that is triggering sensors, which can help reduce the numerous false positives from these sensors.

WISE is a program supported by AFRL that attempts to protect resources by verification of the integrity of transactions within the global infrastructure. Current systems attempt to provide security by building "barriers" and monitoring the characteristics of network traffic which actually prohibit transactions and limit interaction. WISE facilitates global interaction by individually verifying all transactions at several levels. Only valid transactions are allowed. WISE relies on advances in several technology areas including context interactive intelligent sensors and intelligent context recognition (both of which are inherited from DAIWatch), biometrics and participant recognition, fusion, transaction meta-data, data mining and decision support, dynamic integration and control systems and secure network technologies that allow independent collection, monitoring, and response management at the single transaction level.

One intention of Task 17 is for the intelligent agent software technology, developed as a part of DAIWatch, to be integrated into AFED. This effort includes receiving data in AFED from either DAIWatch agents independently, from a central management council, or from the systems database depending on the technical and architectural requirements. Therefore, the main purpose of this task is to create an interface between DAIWatch and AFED. This interface should allow collaboration of specified event data, such as intelligent agent data and IDS data. AFED should be able to query the DAIWatch database and insert the data into its database. The aggregate of this data could be correlated and consolidated to reduce the information overload to network security personnel.

The remaining body of this report will be portioned into four major sections. The first section will be titled Methods, Assumptions, and Procedures, which will describe the design criteria and development measures that were used in creating an interface between DAIWatch and AFED. Results and Discussion will be the subject of the second section. Within this section, the author will document all technical work accomplished and information gained, including processes developed, pertinent observations made, nature of problems seen, positive and negative results noted and lessons learned. Conclusions will be presented in the third section. At this point in the report, substantial findings from the second section will be interpreted regarding their implications, complete with the author's opinion. The fourth major section of the report body will be titled Recommendations. Here, a course of action for the intended audience - the customer and other representatives of applicable government agencies, will be presented, by identifying potential transition opportunities and vehicles. This final section also may include additional areas for study and alternate design approaches.

## 2. METHODS, ASSUMPTIONS, AND PROCEDURES

During the early stages of a previous related development task, the foundation and framework of the design of an interface between DAIWatch and AFED was verbalized in meetings and through email correspondence. The main individuals participating in these communications were representatives from Orincon, the developer of DAIWatch; Northrop Grumman, the prime contractor for the development of AIDE; and AFRL, the government customer. The initial high-level design-by-consensus was later written into a draft version of an overview Interface Design Document (IDD), by Orincon. From these meetings and emails and the IDD, the following components of the interface design were chosen for implementation. It was decided that the interface be developed using Java, since that is the language which currently is making the most advances with XML, and because of code portability and reusability. Instead of just having DAIWatch feed AFED, as stated in the Statement of Work (SOW), it was decided that the exchange of data be bi-directional over a TCP/IP socket, with DAIWatch being the server. As stated in the IDD, the IDS data would be exchanged, as the specified events occur, using a common message format, IDMEF, which is XML-based. It was also decided that since the IDMEF model includes a provision for a heartbeat message, this message should be sent from AFED to DAIWatch. In addition to transmission of data due to event detection, it was decided that a means of performing queries, with multiple optional parameters, on another system's data, should be implemented using a newly-defined XML-formatted request message, designed by Orincon. In regard to data message security, the consensus was that JCE triple-DES encryption would be an adequate implementation in Java, for the interface.

However, for this task, many of the capabilities stated in the previous paragraph were not required by the Task 17 SOW. Therefore, the non-applicable functionality was stripped out of the previous task's source code, to essentially produce a stream-lined unidirectional interface that transmits every new DAIWatch event table record in IDMEF XML format to AFED, on a periodic basis. Some new code was added and changes were made, as needed, during this task, such as: increasing the size of the IDMEF object socket, limiting the number of DAIWatch event records returned per query, creation of a DAIWatch destination IP address lookup table, and generation of a DAIWatch heartbeat, in order to increase the performance of the data throughput and to supply additional, but critical, data points to AFED.

The design approach taken by the sole interface developer was one of researching on the internet and in books for code to perform a specific function, modify it and tweak it, then add more code modules in order to handle additional functionality. Thus, there was much trial-and-error coding and testing during development. Since both of the DAIWatch and AFED systems have Oracle databases; it made sense to the interface developer to use APIs such as the Oracle XML SQL Utility for Java, Oracle XML Parser for Java and XSLT Processor, and the Oracle XML Class Generator for Java. No effort was made to research or use generic APIs which could be used with other databases. DOM Parser APIs were used instead of the Simple API for XML (SAX) version because the Oracle XML Parser for Java documentation guidelines stated that the DOM is recommended for performing (XSL) transformations, even though the DOM consumes a lot of memory for large XML documents. In addition, most of the parsing and transforming examples

in the documentation used DOM, which made the decision to use those APIs easier. At the time of development, Sun Microsystems's Java API for XML Processing (JAXP) was not supported by the Oracle XML Parser for Java, therefore, that technology was not used. Another Sun package, called Java API for XML Binding (JAXB), was not used since the Oracle XML Class Generator APIs performed what was needed.

See the Appendix for information regarding the Development Environment and Test Bed Setup.

## 3. RESULTS AND DISCUSSION

The end result of this task is a Java and XML-based interface which provides the means for integration of host IDS events from the Oracle database of DAIWatch into the Oracle database populated by AFED. The data is sent from one system to another over a TCP/IP socket in XML objects that conform to the IETF Working Group IDMEF. Security of the data transmission is assured by a JCE implementation of the triple-DES process. Events are selected into the interface from a database, through JDBC, using SQL statements as inputs to the Oracle XSU API. The DAIWatch database is queried on a pre-set periodic basis for every new event record that has been stored since the last query was performed. The resultant sets of database records are stored in W3C XML DOM format, and then transformed into IDMEF XML objects via specific XSL files. Proper structure and content of the transformed XML files is validated against specific DTD files. The maximum event identifier from the query resultant set is extracted and stored on the hard drive for subsequent query reference. After the IDMEF XML objects are serialized and encrypted, they are sent across the socket to the other system, where the entire process is performed in reverse. The objects are decrypted, unserialized, validated against a DTD, transformed via XSL into the current system's database table schema, DTD-validated again, and then SQL-inserted into the database, via an Oracle XSU API.

See the Operation Details section of the Appendix for a step-by-step, in-depth description of the entire IDS data exchange process.

The sole developer, at AFRL, on this project, attempted to implement the design of an interface between WISE and CSMS, as closely as possible to the intent of the Task 17 SOW. However, at the time that this system integration work was performed, WISE and CSMS were not available. Therefore, since DAIWatch is the precursor to WISE and AFED is the forerunner to CSMS, it was decided to enhance the method of data transfer from DAIWatch to AFED, which, at some future point in time, could be adapted for use by the WISE and CSMS programs, with a modicum of effort. Other than that major discrepancy, the task objective was met, with just a couple of minor variations.

One minor issue is that AFED was programmed to be the socket server in the actual interface, instead of DAIWatch. There is no good reason that AFED must be the socket server, instead of DAIWatch, other than the fact that development of the interface began with the socket server and client both being on an AIDE box. However, according to the client/server concept, DAIWatch should be the server, since the purpose of the interface is for DAIWatch to serve data to AFED, which by definition would make AFED the client. On the other hand, an important point to bring to light here, is that the interface is currently coded in a manner that the server socket must be started first. Therefore, this condition could affect a site's preferred startup sequence of its systems.

Another minor issue related to insertion of data from one system's database into another systems' database, that has not been addressed, is that of the DAIWatch NTA_Alert table, which records the network IDS data observed by the four possible sensors: RealSecure, NFR, Snort,

and Cybercop. IDS event data currently flows only from the DAIWatch Event table to the AFED Event table. During the development phase of this task, there were no records available in the NTA_Alert table of the database of the DAIWatch machine, on which to test. Therefore, by default, all of the records stored in the DAIWatch Event table, during the testing phase of this task, are host IDS events. Unfortunately, there were no DAIWatch agents assigned to monitor any of the possible network sensors, in order to populate the NTA_Alert table. Therefore, there was no code created by the interface developer, to transfer network-based IDS events from DAIWatch to AFED. On July 2, 2003, a chart was received from Orincon that maps the data between the AIDE Event table and various DAIWatch tables. This data map chart states that the DAIWatch Event table has a one-to-many relationship with the DAIWatch NTA_Alert table, with the Event_ID field of the NTA_Alert table being a foreign key to the Event_ID field of the Event table. There are only three fields from the DAIWatch Event table that map directly to the AIDE Event table. On the other hand, there are 11 fields from the DAIWatch NTA_Alert table which map directly to the AIDE Event table. Apparently, the DAIWatch Event table provides a Machine (or Destname), Event_Type (or Signature) and Description for both host-based and network-based events, while the NTA_Alert table provides additional information only for network-based events, which have been observed by as many as four different sensors. This being so, leads to quite the complex and intricate coding situation for future development.

There were no performance metrics measured, per se; however, the interface was run over the weekend. All of the new event records appearing in the DAIWatch database had been transmitted to AFED and also appeared in its database, successfully. Although Java has a reputation of being slow, as compared to C and C++, it seems to be able to handle the low volume of data exchanges that were run during this weekend test and that are anticipated to be seen during actual usage. However, a couple of times during the initial phases of operation of the interface, the processing doesn't just slow down, it actually comes to a stop. The first time that the encryption method is called upon in the program to read the hard-coded secret key, there is a delay of up to 10 seconds due to the many security checks performed by the JCE. Once again, this delay occurs at the beginning stages of data exchange, just after startup, and therefore should not be an issue. Another time, the program stopped when a lack of memory error occurred during transfer of a very large IDMEF XML object containing multiple DAIWatch event records across the socket connection. This error condition was eliminated by limiting the number of records returned per query to 512 and by increasing the frequency of transactions from once every 5 minutes to every 30 seconds. Both of these parameters can be adjusted and set by the user in a configuration (properties) file, prior to interface startup.

## 4. LESSONS LEARNED

1) Using XML/XSL/DTD does not entirely prevent one from having to modify source code and re-compile when database changes are made. This is true because, if the XML element tag which is being searched for by the source code changes, then the source code must be updated, also.

2) Since this interface only deals with the exchange of database records between two databases, the server and client components of the interface are not bound to run on the same machines of either database, because of the capability of JDBC with remote login.

3) A majority of the source code is the same for both the server and client components of the interface. The only difference is the TCPServer and TCPClient modules, which contain the socket server and socket client code, respectively. The use of Java property files, functioning as configuration files, enabled the assignment of unique values at startup, which allows the source code to be generic.

## 5. CONCLUSIONS

Since WISE and CSMS were not available during performance of this task, and since Orincon did not have the time or manpower to create WISE or DAIWatch agents for AFED, this interface is essentially only half done. Therefore, one could label this interface as just a DAIWatch sensor tap for AFED. Taking this line of thinking even further, one could state that this interface is relegated to being a tap for host-based events only, since it has not been tested with any network-based sensor data from the DAIWatch database.

Even though the interface is essentially only half operational, it still demonstrates the versatility and power of the XML, XSL and DTD files that are built into it. Not only are two databases with different schemas allowed to exchange data without utilizing complex hard-coded conversion routines, the interface code also allows for minimal source code updates, when database tables and fields are modified or a new IDMEF version is issued, thereby reducing the need for re-compilation.

It is the author's opinion that if WISE and CSMS had been made available during this task, it would have been very interesting and beneficial to see the advanced technology capabilities of WISE to be integrated into an all-encompassing information protection concept, such as CSMS. Most importantly, since DAIWatch is the precursor to WISE and AFED is the forerunner to CSMS, this robust Java/XML interface between DAIWatch and AFED will provide a good foundation for the possibility of WISE integration into CSMS, in the future.

## 6. RECOMMENDATIONS

Listed below are recommendations for the enhancement of the current design.

- Create DAIWatch agents for AFED, with assistance from Orincon.
- Modify code to allow either the client or server to be started first. Currently, the server must be started first.
- Develop code to allow selection of Network IDS data from the DAIWatch NTA_Alert table into the AFED Event table.
- Upgrade all XML DTD files into XML Schema files.
- Upgrade GUIs to use Swing components, which are independent of the X-window server, instead of just AWT components, which are dependent.
- Incorporate Sun JAXP APIs into code to allow other XML parsers to be used, instead of just Oracle's, in case other databases are used.
- Incorporate more Oracle JAXB APIs into code to reduce/replace dependencies on org.w3c.dom Node method calls; if possible.
- Research the use of SAX vs. DOM to parse XML documents, to lessen memory usage.
- Develop makefiles to enable creation of classes.jar on a Windows OS box.

Listed below are recommendations for alternate designs.

- Create additional code for AFED that will programmatically send requests to DAIWatch in order to complete missing segments of a correlation scenario.
- Change tables in the conventional relational Oracle databases of AFED and DAIWatch to contain IDMEF elements in combination with an upgrade to Oracle XML DB, which provides specialized (hierarchical) indexing in order to support improved access to XML data.

As stated previously in this report, since DAIWatch is the precursor to WISE and AFED is the forerunner to CSMS, this robust Java/XML interface between DAIWatch and AFED will provide a potential transition opportunity and vehicle for the possibility of WISE integration into CSMS, in the future.

## APPENDIX A - INSTALLATION AND MAINTENANCE GUIDE

### UNIX Source and Binaries

Host: Rock
Operating System: Solaris 5.8
Java: j2sdk-1_4_1_02-solaris-sparc.sh from http://java.sun.com/j2se/downloads.html
installed in /usr directory

| | | | |
|---|---|---|---|
| java: | /usr/j2sdk1.4.1_02/jre/bin/java | with link to | /bin/java |
| javac: | /usr/j2sdk1.4.1_02/bin/javac | with link to | /bin/javac |
| jar: | /usr/j2sdk1.4.1_02/bin/jar | with link to | /bin/jar |

Oracle Database: /oracle/product/server/9i - Oracle9i Enterprise Edition Release 9.0.1.0.0 - Production
Oracle Database: ( Host: TryIt ) - Oracle9i Enterprise Edition Release 9.0.1.0.0 - Production

OracleXDKforJava: xdk_java_9_2_0_5_0.tar.gz from
http://otn.oracle.com/software/tech/xml/xdk_java/index.html
saved in /oracle/product/server/9i directory, extracted from ./lib directory only

classgen.jar, xdb.jar, xmlparserv2.jar, and xsu12.jar moved to
…/Task_17/shared/Oracle_XDK_92050_lib/ directory

Oracle JDBC: classes12.zip from http://otn.oracle.com/software/tech/java/sqlj_jdbc/index.html
saved in …/Task_17/shared/Oracle_JDBC_9203/ directory

IDMEF DTD: idmef-message.dtd from http://www.ietf.org/html.charters/idwg-charter.html
saved in …/Task_17/shared/ directory

Integration Code: …/Task_17/server/AFED/ directory
AFED-event-table.dtd , AFED-sensor-table.dtd, database_props.txt,
IDMEF-alert_to_AFED-event.xsl, IDMEF-heartbeat_to_AFED_sensor.xsl,
Makefile, query_index_props.txt, server_props.txt, start.sh

…/Task_17/client/ directory
client_props.txt, DAIW-alert-table.dtd, DAIW-destinationIPs.dtd,
DAIW-destinationIPs.xsl, DAIW-event-table.dtd,
DAIW-event_to_IDMEF-alert.xsl, DAIW-host_to_IDMEF-heartbeat.xsl,
database_props.txt, Makefile, query_index_props.txt start.sh

Integration Code: …/Task_17/shared/
DOM_API.java, idmef-message.dtd, keygen.sh, Makefile, OracleDatabase.java,
PwdCollector.java, PwdTxtFldFrameDrawer.java, QueryIndex.java,
ReturnXML.java, TCPClient.java, TCPServer.java, Tool.java, TripleDES.java

…/Task_17/shared/Return_Classes/
_ReturnClassGen.java, return.dtd, return.xsl, Makefile

Compiling Process: change directory to
…/Task_17/client,
…/Task_17/server/AFED, then
type “make all”

The following actions will occur:

1) directory location will change to: … Task_17/shared
2) “make all” will be called in the shared directory
3) all class files will be removed from the shared directory
4) directory location will change to … Task_17/shared/Return_Classes
5) “make all” will be called in the Return_Classes directory
6) all class files will be removed from the Return_Classes directory
7) shared/DOM_API.java will be compiled
8) Return_Classes/_ReturnClassGen.java will be compiled
9) ${JAVA} -classpath "$(MAKE_CLASSPATH)" _ReturnClassGen -root return return.dtd
   will be run, which produces java source files for each node and element in return.dtd
10) each java file, generated in step 9, will be compiled
11) each java file, generated in step 9, will be deleted
12) directory location will change to: … Task_17/shared
13) “make ReturnXML.class” will be called in the shared directory
14) ${JAVA} -classpath "$(MAKE_CLASSPATH)" ReturnXML output > return.xml
   will be run, which generates an XML document, return.xml that conforms to return.dtd
15) all java files in the shared directory will be compiled
16) all class files in the shared directory will be moved to the original directory, listed prior to step 1
17) all class files and Return_dtd.txt in Return_Classes directory will be moved to original directory
18) all class files and Return_dtd.txt in original directory will be archived into classes.jar
19) all class files and Return_dtd.txt in original directory will be deleted
20) keygen.sh and idmef-message.dtd in the shared directory will be copied to the original directory
21) return.dtd and return.xsl in the Return_Classes directory will be copied to the original directory

## Windows Source and Binaries

Host: d2101ra046234 (except for Oracle Database)

Operating System: Microsoft Windows XP Professional 5.1.2600 Service Pack 2

Java: jdk-1_5_0_04-nb-4_1-win.exe from http://java.sun.com/j2se/downloads.html
installed in C:\ directory
java: C:\Program Files\Java\jdk1.5.0_04\bin\java.exe
javac: C:\Program Files\Java\jdk1.5.0_04\bin\javac.exe
jar: C:\Program Files\Java\jdk1.5.0_04\bin\jar.exe

Oracle Database: Apex::C:\oracle - Personal Oracle9i Release 9.0.1.1.1 - Production

OracleXDKforJava: classgen.jar, xdb.jar, xmlparserv2.jar, and xsu12.jar copied from
Rock:: .. /Task_17/shared/Oracle_XDK_92050_lib/ directory
to … \Task_17\shared\Oracle_XDK_92050_lib directory

Oracle JDBC: classes12.zip copied from
Rock:: .. /Task_17/shared/Oracle_JDBC_9203/ directory to
… \Task_17\shared\Oracle_JDBC_9203 directory

Integration Code: same as Unix, except base directory structure is … \Task_17\ instead of … /Task_17/

Compiling Process: files in the client, server/AFED directory compiled on a Unix box
can be copied to the corresponding directories on a Windows box

Develop. Environ.: JCreator LE version 2.5 build 6 from http://www.jcreator.com

## DAIWatch -> AFED testing/operation

TCP socket server: resided on d2101ra046234 ( Windows XP )
TCP socket client: resided on d2101ra046234 ( Windows XP )

AFED Oracle JDBC settings: d2101ra046234:: … /Task_17/server/AFED/database_props.txt
HOST_IP_ADDRESS=XXX.XXX.XXX.217
SERVICE_NAME_OR_SID=SID=EPIC

AFED Oracle JDBC Thin client: resided on d2101ra046234 ( Windows XP )
AFED Oracle database server: resided on TryIt (.216) ( Solaris )

DAIW Oracle JDBC settings: d2101ra046234:: … /Task_17/client/database_props.txt
HOST_IP_ADDRESS=XXX.XXX.XXX.63
SERVICE_NAME_OR_SID=SERVICE_NAME=IADB

DAIW Oracle JDBC Thin client: resided on d2101ra046234 ( Windows XP )
DAIW Oracle database server: resided on Apex (.63) ( Windows XP )

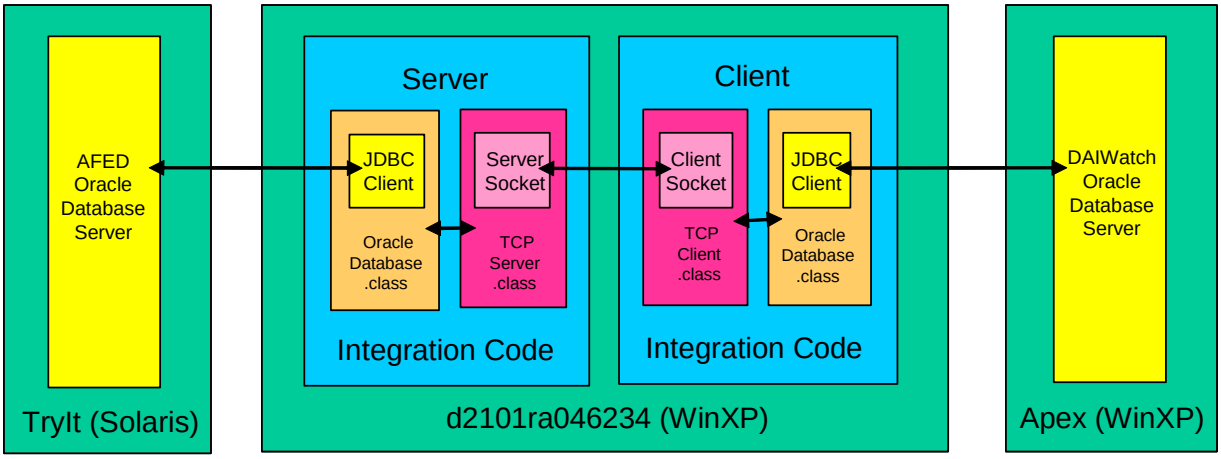


**Figure 1: AFED <-> DAIWatch Testing Scenario**

## APPENDIX B - SOFTWARE USER'S MANUAL

### Overview

The integration code interface between DAIWatch and AFED consists of a TCP/IP socket, where DAIWatch is configured as the client and AFED as the server. AFED establishes a server socket on a specified communication port and listens for incoming socket connection requests from the DAIWatch client. The interface is bi-directional with information flowing in both directions between the DAIWatch client and the AFED server. Specifically, DAIWatch sends heartbeat and alert messages in the common Intrusion Detection Message Exchange Format (IDMEF) to AFED for database insertion and AFED sends database record insertion status and query index data, in basic XML format, back to DAIWatch.

The DAIWatch component of the interface queries the database on a periodic interval for every new record stored in the Event table since the last query. These query results are put into XML format, then transformed into IDMEF for transmitting to the AFED component for processing. Another message that employs the IDMEF is the "heartbeat". To ensure that DAIWatch is still active and requires an interface, a "heartbeat" signal is sent from DAIWatch to AFED every 2 minutes. Since data is transmitted externally from each of the systems, security of the interface is an important issue. To establish security of the data, the communication between DAIWatch and AFED utilizes JCE tripleDES encryption.

### Pre-operation Setup

The first step to be performed before actually firing up the separate client and server components of the interface is to ensure that the respective property files: client_props.txt, server_props.txt and database_props.txt contain the correct settings.

Task_17/client/client_props.txt -> TCPClient.class in Task_17/client/classes.jar
Task_17/client/database_props.txt -> OracleDatabase.class in Task_17/client/classes.jar

Task_17/server/AFED/server_props.txt -> TCPServer.class in Task_17/server/AFED/classes.jar
Task_17/server/AFED/database_props.txt -> OracleDatabase.class in Task_17/server/AFED/classes.jar

For example, the server_props.txt file in the …/Task_17/server/AFED directory contains the settings for the TCPServer class in the …/Task_17/server/AFED/classes.jar executable file.
Whereas, the database_props.txt file in the …/Task_17/server/AFED/ directory contains the settings for the OracleDatabase class in the …/Task_17/server/AFED/classes.jar executable file.

Most of the settings need not be changed, such as the names of the *.dtd and *.xsl files. Two settings that definitely need to be assigned are the HOST_IP_ADDRESS and SERVICE_NAME_OR_SID properties in the database_props.txt file of each of the client and server directories. These JDBC settings dictate to the server and client the location of its associated database. The SITE_LOC and SITE_NAME properties in the database_props.txt file of the client directory should also be checked prior to startup. These two settings are needed in the AFED Event table for identifying the DAIWatch sensor. The QUERY_INTERVAL property in the client_props.txt file, which controls how often events are searched in the DAIWatch

database. This property should be set in conjunction with the NUMBER_OF_EVENTS property in the database_props.txt file, which is also located in the client directory. Both of these properties should be set to accommodate the rate of events being stored in the DAIWatch database.

**Server Property File** - AFED server_props.txt file contents

```
# type of system:  AFED or DAIW
SYSTEM_TYPE=AFED

# stylesheet to transform IDMEF Alerts to AFED Events
IDMEF_ALERT_TO_EVENT_TABLE=IDMEF-alert_to_AFED-event.xsl

# dtd to validate IDMEF Alerts and Heartbeats
IDMEF_DTD=idmef-message.dtd
IDMEF_ROOT=IDMEF-Message

# dtd to validate Return XML after SQL insert
RETURN_DTD=return.dtd
RETURN_ROOT=return

# stylesheet to create Return XML after SQL insert
RETURN_XSL=return.xsl

# dtd to validate Event records after queries and before inserts
EVENT_DTD=AFED-event-table.dtd
EVENT_ROOT=ROWSET

# stylesheet to transform IDMEF Heartbeats to AFED Sensor records
IDMEF_HBEAT_TO_SENSOR_TABLE=IDMEF-heartbeat_to_AFED-sensor.xsl

# dtd to validate Sensor records before heartbeat inserts
SENSOR_DTD=AFED-Sensor-table.dtd
SENSOR_ROOT=ROWSET
```

**Database Property File -** AFED database_props.txt contents

```
# max. number of Event table records to be returned per query
NUMBER_OF_EVENTS=0

# IP address of host of AFED database - Tryit: 216, Devel: 217
HOST_IP_ADDRESS=XXX.XXX.XXX.216

# TCP connect data string of AFED database: SERVICE_NAME=? or SID=?
SERVICE_NAME_OR_SID=SID=EPIC

# table name and field name used for event queries
EVENT_TABLE=event
EVENT_ID_FIELD=event_id
```

```
# stylesheet to transform DAIW destination IPs - used by client ONLY
DESTINATION_IP=

# table name - field name used for destination IPs -used by client ONLY
# EVENT_TABLE is already defined above
HOST_IP_TABLE=
MACHINE_TABLE=
NET_DEV_TABLE=
MACHINE_FIELD=
MACHINE_ID_FIELD=
HOSTID_FIELD=
HOSTIP_FIELD=
NETDEVID_FIELD=

# stylesheet to transform host table data into IDMEF Heartbeat - used by client ONLY
HOST_TABLE_TO_IDMEF_HEARTBEAT=

# table names and field names used for DAIW heartbeat queries - used by client ONLY
HOST_TABLE=
HOST_IP_FIELD=
HOST_NAME_FIELD=
```

**Client Property File -** DAIWatch client_props.txt file contents

```
# type of system: AFED or DAIW
SYSTEM_TYPE=DAIW

# 2 minutes HeartBeat interval
PULSE_INTERVAL=120000

# 0.25 minute Query request interval
QUERY_INTERVAL=15000

# dtd to validate IDMEF Alerts and Heartbeats
IDMEF_DTD=idmef-message.dtd
IDMEF_ROOT=IDMEF-Message

# dtd to validate Return XML after SQL insert
RETURN_DTD=return.dtd
RETURN_ROOT=return

# stylesheet to create Return XML after SQL insert
RETURN_XSL=return.xsl

# dtd to validate Event records after queries and before inserts
EVENT_DTD=DAIW-event-table.dtd
EVENT_ROOT=ROWSET

# dtd to validate Destination IP records
DEST_DTD=DAIW-destinationIPs.dtd
DEST_ROOT=Destination
```

**Database Property File -** DAIWatch database_props.txt contents

```
# max. number of Event table records to be returned per query
NUMBER_OF_EVENTS=500

# IP address of host of DAIW database
HOST_IP_ADDRESS=XXX.XXX.XXX.63

# TCP connect data string of DAIW database: SERVICE_NAME=? or SID=?
SERVICE_NAME_OR_SID=SERVICE_NAME=IADB

# site location and site name used for AIDE/AFED Event table
SITE_LOC=DIW_LAB
SITE_NAME=AFRL

# stylesheet to transform DAIW Events into IDMEF Alerts
EVENT_TABLE_TO_IDMEF_ALERT=DAIW-event_to_IDMEF-alert.xsl

# stylesheet to transform site table data into IDMEF Heartbeat - NOT USED
SITE_TABLE_TO_IDMEF_HEARTBEAT=

# table name and field name used for event queries
EVENT_TABLE=event
EVENT_ID_FIELD=event_id

# stylesheet to transform DAIW destination IPs
DESTINATION_IP=DAIW-destinationIPs.xsl

# table names and field names used for destination IPs
# EVENT_TABLE is already defined above
HOST_IP_TABLE=host_IP
MACHINE_TABLE=machine
NET_DEV_TABLE=network_device
MACHINE_FIELD=machine
MACHINE_ID_FIELD=machine_ID
HOSTID_FIELD=hostID
HOSTIP_FIELD=hostIP
NETDEVID_FIELD=networkDeviceID

# stylesheet to transform host table data into IDMEF Heartbeat
HOST_TABLE_TO_IDMEF_HEARTBEAT=DAIW-host_to_IDMEF-heartbeat.xsl

# table names and field names used for DAIW heartbeat queries
HOST_TABLE=hostfileinfo
HOST_IP_FIELD=host_IP
HOST_NAME_FIELD=host_name
```

As previously mentioned in the overview, the communication between DAIWatch and AFED utilizes Java Cryptography Extension (JCE) tripleDES (or DESede) encryption. Therefore, the next step in the pre-start process is the generation of the secret shared keyfile. DESede is a

symmetric key encryption algorithm, which means that the key for encryption and decryption are identical. This is accomplished by changing directory to the client or server code and typing "keygen.sh". A 24-byte encrypted string will be generated and stored in a file named "keyfile. For example, if the keyfile is generated in the DAIWatch/server/AIDE directory on the host acting as the server, then the keyfile will have to be copied to the DAIWatch/client directory on the host acting as the client, and vice-versa.

**Key Generation Shell Script** - DAIWatch/server/AIDE/keygen.sh

```
JAVA=/bin/java
${JAVA} -cp "./classes.jar" TripleDES -g
```

Once the property file values are set and the encryption key is generated and copied, then it is time to start the server and client portions of the interface. The server code must be started first in order for the server socket to listen for a connection attempt from the client code. NOTE: If the client is started first, then the program will exit.

**Start Server Shell Script** – Task_17/server/AIDE/start.sh

```
SHARED= … Task_17/shared
ORACLE_JDBC=${SHARED}/Oracle_JDBC_92030
ORACLE_XDK=${SHARED}/Oracle_XDK_92050_lib

JAVA=/bin/java

${JAVA} -cp "./classes.jar:${ORACLE_JDBC}/classes12.zip:${ORACLE_XDK}/xsu12.jar:${ORACLE_XDK}
/xmlparserv2.jar:${ORACLE_XDK}/xdb.jar:${ORACLE_XDK}/classgen.jar" TCPServer $1
```

**Start Client Shell Script** – Task_17/client/start.sh

```
SHARED=… Task_17/shared
ORACLE_JDBC=${SHARED}/Oracle_JDBC_92030
ORACLE_XDK=${SHARED}/Oracle_XDK_92050_lib

JAVA=/aide/java/jre/bin/java

${JAVA} -cp "./classes.jar:${ORACLE_JDBC}/classes12.zip:${ORACLE_XDK}/xsu12.jar:${ORACLE_XDK}
/xmlparserv2.jar:${ORACLE_XDK}/xdb.jar:${ORACLE_XDK}/classgen.jar" TCPClient $1 $2
```

The one argument to "TCPServer" in server/AFED/start.sh is the port number on which the socket server will be listening for attempted connections from the client. If no port number is given, then it defaults to "55555". The two arguments to "TCPClient" in client/start.sh are the server host IP address and the socket server port number, in that order. Again, if no port number is given, then it defaults to "55555" however, if no IP address is given then it defaults to "localhost". In total, four connections are made from four separate client ports to the one server port. Each of the four connections handles a different type of data transmission: status strings, IDMEF objects, request objects, and menu (signature) objects. Segregating the data in this

manner, prevents a socket on the receiving end of a transmission from having to identify the data and its content in order to determine where it should be sent for decryption, XSL transformation, XML parsing and further processing, if any of these actions are at all applicable.

**Operation Flow Description**

**LEGEND**: Stars with numbers correspond to positions in the flowcharts on pages 24 - 26.

1 Upon startup of the client and server components, just prior to the socket connection process, the values in the three property files, database_props.txt, query_index_props.txt, and client_props.txt or server_props.txt are loaded into memory. The database_props.txt, client_props.txt and server_props.txt files were briefly explained earlier in this Operation Details section, but not the query_index_props.txt file. The purpose of the query_index_props.txt file is to keep track of the ID of the last event record that was accessed from the DAIWatch database, in order to prevent duplicate data retrieval. This entire process is handled programmatically, which is why query_index_props.txt was not mentioned earlier in the property files preparation text.

2 A small window entitled “Oracle Database Login”, preceded by the system type, i.e., AFED or DAIWatch, should appear on the screen. The TCPServer component of the interface will make a JDBC connection to an AFED or AIDE database, while a TCPClient component will make a JDBC connection to a DAIWatch database. Text inside the window prompts the user to enter a username/password string that includes the forward slash, without any spaces, i.e., “Username/Password”. The database login window will reappear on the screen if an incorrect username/password combination is entered and will continue to reappear, until the correct combination is entered. The user submits the entered text, by pressing the Enter key or by clicking on the OK button.

3 Once a connection is made to both of the databases associated with TCPServer and TCPClient, SQL queries can be performed. The first action to take place is to assure that the maximum record ID listed in the query_index_props.txt file is still relevant, in relation to the events stored in the database. For instance, if the database had recently been cleared, then none of the new events in the database will be accessed by the interface until their record IDs exceed the one currently listed in the query_index_props.txt file. The checkMaxQueryID method assesses the relevance of the query index property value by searching the database for a record containing

4 that event ID number. If the number does not exist, then the value of the property in the query_index_props.txt file is reset to zero, in addition to the query index property value currently stored in memory. However, if the record ID still exists in the database, then nothing is changed nor updated.

The next group of actions in the interface, following establishment of JDBC connections to both databases, deals with generation of a lookup table of available destination IP hostnames to destination IP addresses in the DAIWatch database. The destination IP lookup table is stored in memory by the DAIWatch client for future access during XSL transformation of event XML objects to IDMEF. The first step in this process is for a four table SQL join to be performed on the DAIWatch database. The resultant set of IP hostnames and IP addresses from the database

query is stored in an XML document object model (DOM), is transformed by the DAIW-destinationIPs XSL file, then is validated against a Document Type Definition (DTD) file, for proper structure and content. The IP hostnames and IP addresses are extracted from the XML object via DOM tree-traversal methods, and then deposited into a hash table map.

5

Another group of preliminary activities that must be performed prior to the exchange of data between the client and server is the starting of thread processes and the scheduling of timer tasks. The DAIWatch client first schedules the heartbeat task according to the PULSE_INTERVAL property value and schedules the event query task according to the QUERY_INTERVAL property value, then it starts the SQL insert status return receive thread. While these processes are begun, the AFED server starts the IDMEF alert/heartbeat receive thread.

6 Every QUERY_INTERVAL period of time, the qryTask in the DAIWatch client is executed. This entails envoking the "alert" portion of the method that creates Oracle SQL select statements. Within this same method, the latest record ID (index) that has been accessed, during prior queries, is obtained from the query_index_props.txt file. Once an index is obtained, it is added to an SQL select statement which searches the DAIWatch database event table, via the Oracle XML SQL Utility (XSU), for records that contain all events with an event ID greater than the index. Zero or more records are returned, which are put into one XML file, validated against

7 the DTD pertaining to the database table from which the data was queried, transformed into IDMEF according to the appropriate XSL file, then validated against the IDMEF DTD. Before the IDMEF object is sent back to the server component, it must first be serialized and encrypted, and then it is finally put on the client component idmefSocket for transmission.

13 Every PULSE_INTERVAL period of time, the hbTask in the DAIWatch client is executed. This entails envoking the "heartbeat" portion of the method that creates Oracle SQL select statements. Once the SQL select statement is executed and the data is returned, it is put into one XML file,

14 transformed into IDMEF according to the appropriate XSL file, then validated against the IDMEF DTD. Before the IDMEF object is sent back to the server component, it is first serialized and encrypted, and then it is finally put on the client component idmefSocket for transmission.

NOTE: The idmefReceive Thread loop, which is monitoring the idmefSocket of the server component for objects received from the client component, will send the object containing the DAIWatch event records or heartbeat data to the appropriate methods for decrypting, unserializing, parsing and DTD-validation. After the object is changed into XML DOM form, the data type is extracted.

8 If the XML DOM contains event data, then it will be transformed into an XML file that conforms as much original DAIWatch event table data into the AFED event table format, as it can. At this point, the XML DOM will be validated against the DTD pertaining to the database

9 table into which the data will be inserted. The next step in this process is for the Oracle XSU to insert the entire set of DAIWatch event records in the XML file into the AFED database event table in one batch. After the data is successfully inserted and committed, the count of the number of rows returned from the Oracle XSU insertXML API is added to a return XML object. The maximum event ID (or query index) is extracted from the XML file that was just inserted into the database, and it is also added to the return XML object. The return XML object is created

10 from the return DTD, transformed with the return XSL, validated against the return DTD, and then put on the server component returnSocket for transmission to the client. The returnReceive Thread loop, which is monitoring the returnSocket of the client component for objects received from the server component, will send the object containing the AFED database event record insertion status data to the appropriate methods for decrypting, unserializing, parsing and DTD-validation. As a final step, the query index properties list in memory is updated with the maximum event ID found in the return XML object and the query_index_props.txt file is updated and saved.

11

12

15 If the XML DOM contains heartbeat data, then it will be transformed into an XML file that conforms to the AFED Sensor table. At this point, the XML DOM will be validated against the DTD pertaining to the database table into which the heartbeat data will be updated. The final step in this process is for the Oracle XSU to update the set of DAIWatch host data in the XML file into the AFED database sensor table.

16

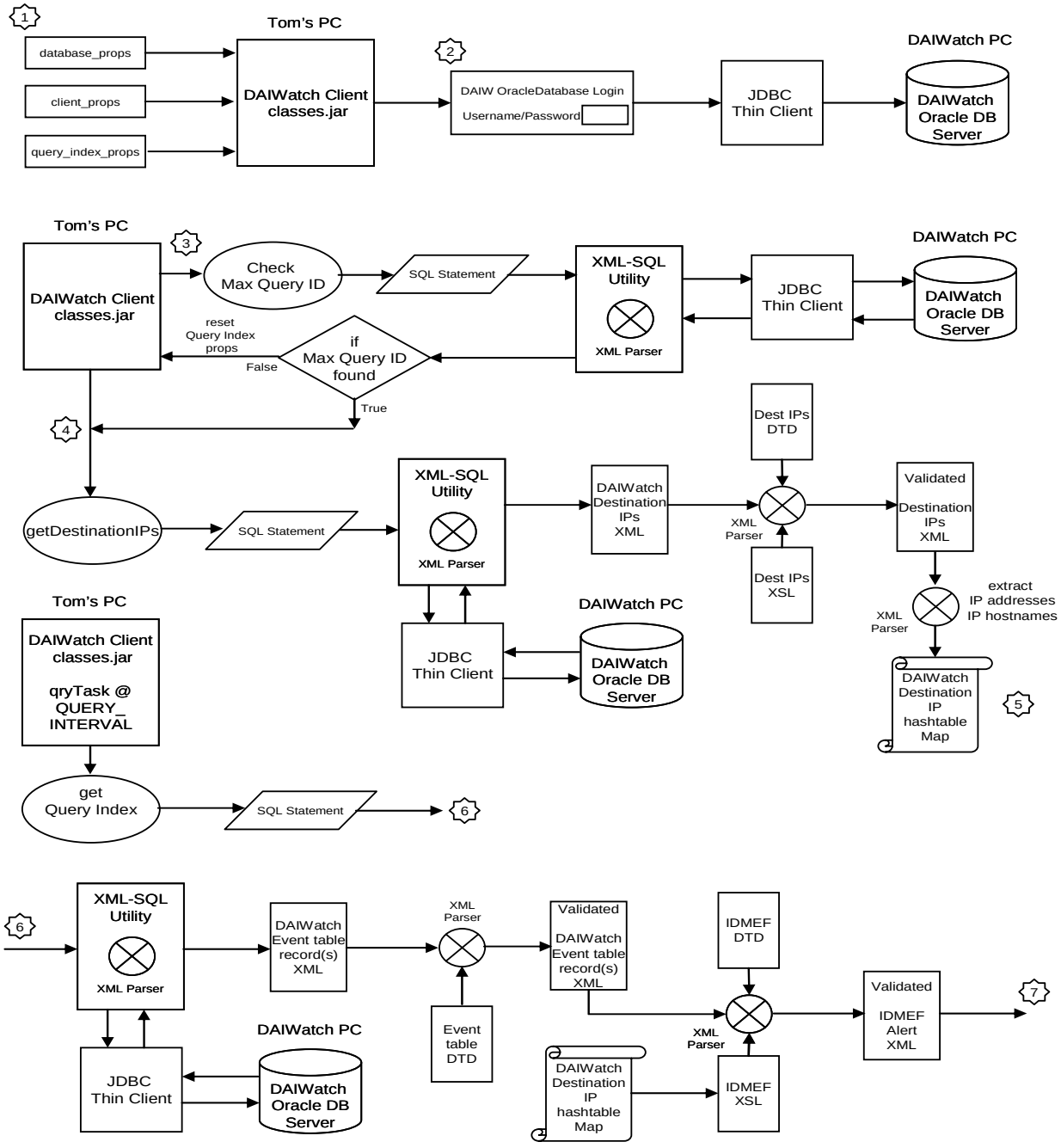


**Figure 2: DAIWatch to AFED Flowchart Steps 1 – 7**

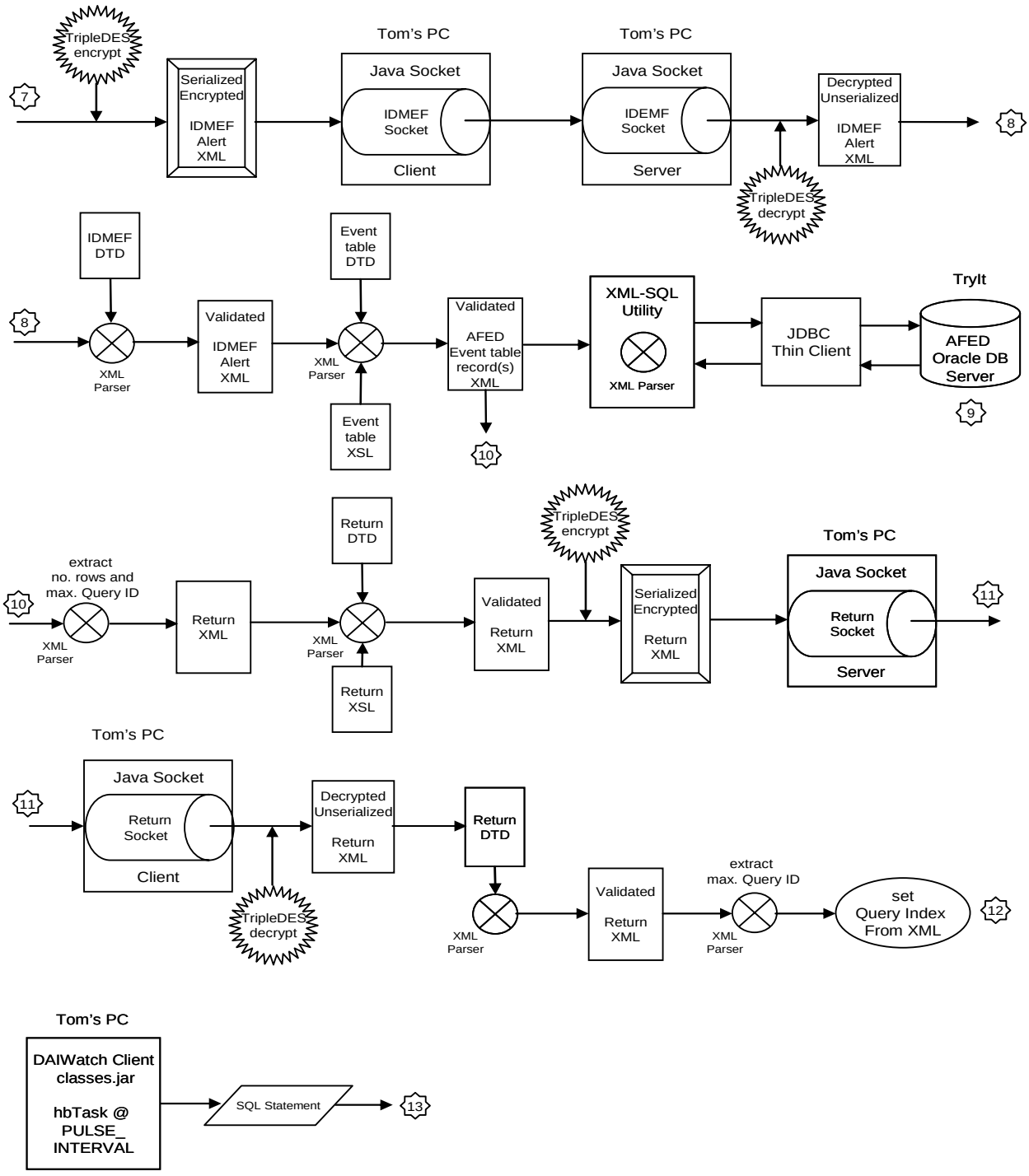


**Figure 3: DAIWatch to AFED Flowchart Steps 7 – 13**

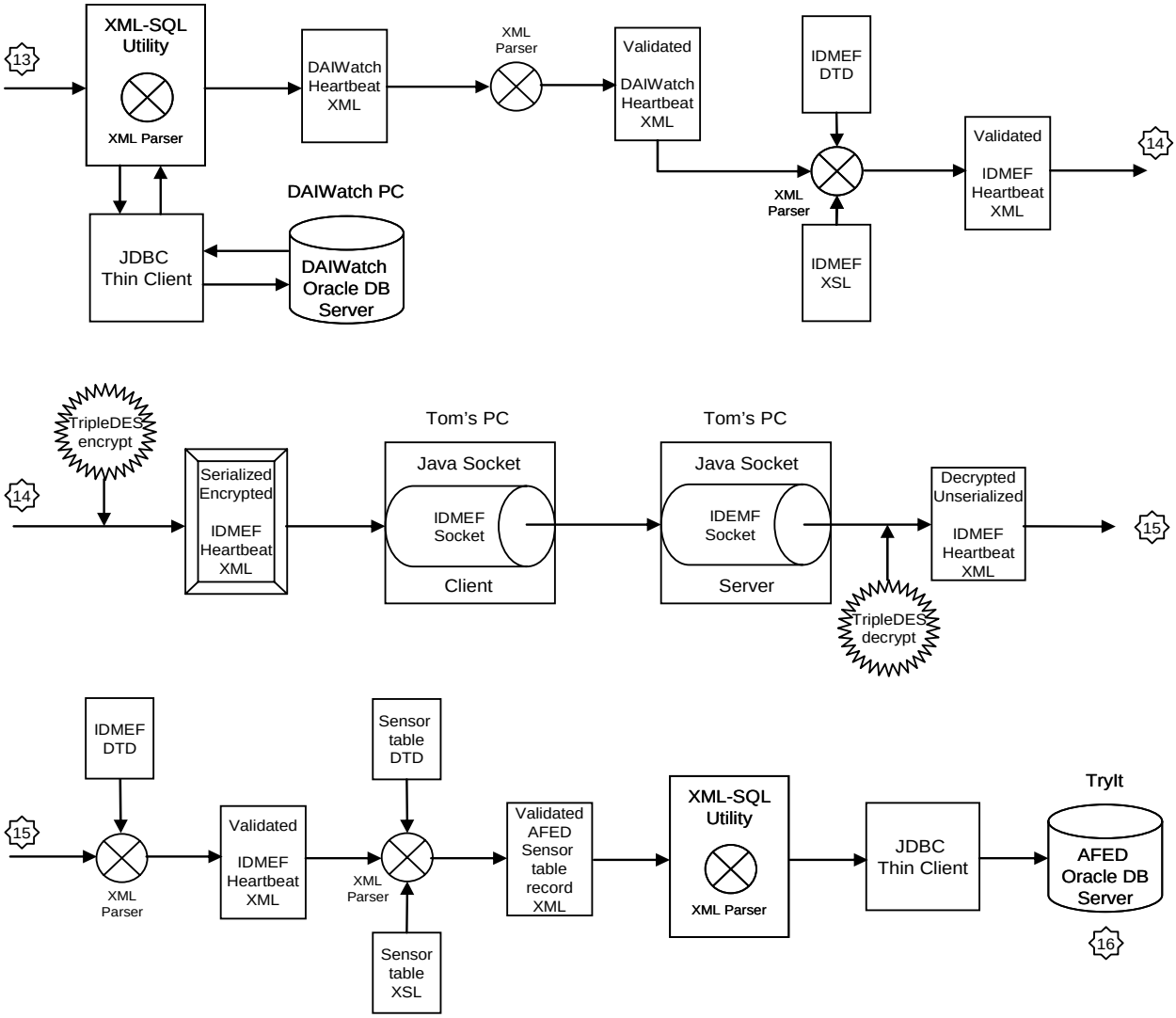


**Figure 4: DAIWatch to AFED Flowchart Steps 13 - 16**

| DAIWatch Event table | DAIW-event_to_IDMEF-alert.xsl Alert class | IDMEF-alert_to_AFED-event.xsl ROW class |
|---|---|---|
| EVENT_ID | ident | |
| EVENT_TYPE | Classification/name | SIGNATURE |
| REPORT_TYPE | AdditionalData type=string meaning=REPORT_TYPE | |
| MACHINE | Source/Node/name | SRCNAME |
| MACHINE | Target/Node/name | DESTNAME |
| MACHINE | Source/Node/Address/address = OracleDatabase.getNumericalDestIP( MACHINE ) | SRCIP = Tool.convertIpStringToDecimal( address ) |
| MACHINE | Target/Node/Address/address = OracleDatabase.getNumericalDestIP( MACHINE ) | DESTIP = Tool.convertIpStringToDecimal( address ) |
| AGENT_NAME | AdditionalData type=string meaning=AGENT_NAME | |
| DESCRIPTION | AdditionalData type=string meaning=DESCRIPTION | DESCRIPTION |
| AGENT_TAHITI_ID | | |
| TIME_STAMP | AdditionalData type=date-time meaning=TIME_STAMP | |
| RISK | Assessment/Impact | |
| ADDEDDT * | CreateTime | CREATE_SNSR_DT after removing 'T' and 'Z' |
| MODIFIEDDT * | AdditionalData type=date-time meaning=MODIFIEDDT | |
| ACTIVE | Assessment/Action | STAT = "PI" if Action='1' |
| REPORTID | AdditionalData type=string meaning=REPORTID | |
| DATAOBSOLETEDBYFUSION | AdditionalData type=boolean meaning=DATAOBSOLETEDBYFUSION | |
| CONFIDENCE | Assessment/Confidence | |
| | | |
| **DAIW-event_to_IDMEF-alert.xsl** | | |
| SENSOR_NAME | Analyzer analyzerid | SENSOR_NAME |
| | | |
| **client/database_props.txt** | | |
| SITE_LOC | Analyzer/Node/location = OracleDatabase.getDataBaseProp( SITE_LOC ) | SITE_LOC |
| SITE_NAME | Analyzer/Node/name = OracleDatabase.getDataBaseProp( SITE_NAME ) | SITE_NAME |
| | | |
| | | PARTITION_VAL = Tool.getNumericalDayOfWeek() |
| | | |
| | | EVENT_ID is created by an Oracle sequence trigger |

*OracleDatabase.createXSUqueryXML( Connection conn, … ) {
  qry.setDateFormat( "yyyy-MM-dd'T'HH:mm:ss'Z'" );

**Figure 5: DAIWatch to IDMEF to AFED data map**