1 Introduction

The Toolkit for Intrusion Alert Analysis (TIAA) was developed based on previous Intrusion Alert Correlator [3]. The primary goal of TIAA is to provide system support for interactive analysis of intrusion alerts reported by IDSs. Some of the existing utilities[2] were extended and more newly developed utilities were integrated into TIAA. In addition, an user friendly GUI interface was developed as part of TIAA.

To install and run TIAA, the Java 2 SDK Standard Edition V1.4 or higher is required. Also the Xerces Java Parser is needed to generate Knowledge Base, and GraphViz is needed to visualize analysis results.

2 System Requirements

- Pentium III, 500 MHz
- 256 MB ram
- 100 MB free space

2.1 Checklist

Here is the list of what you need to run this toolkit.

- Java 2 SDK Standard Edition V1.4 or higher
- Database Management System and corresponding JDBC Driver
- Database of raw alerts generated by IDS, Knowledge Base
- Xerces Java Parser v1.4.4
- AT&T GraphViz
3 Installing TIAA

3.1 Prepare JDBC Connection

All raw alerts and analysis results are under the management of one central Database Management System. In general, any DBMS with JDBC capability can be chosen to work with TIAA. However, due to the SQL syntax difference, minor changes on SQL statements might be needed for certain DBMS.

An appropriate JDBC driver is needed as well. More details can be found here at: http://servlet.java.sun.com/products/jdbc/drivers/browse_all.jsp

After installation of JDBC driver, please make sure its path is included in current CLASSPATH.

We developed and tested TIAA using Microsoft SQL Server and JDBC driver provided by Microsoft.

3.2 Xerces Java Parser

The newest version can be found here http://xml.apache.org/xerces2-j/. Follow the installation instructions and make sure to add its path to current CLASSPATH.

3.3 AT&T GraphVis

Download the newest version at http://www.research.att.com/sw/tools/graphviz/. After installation, make sure the path of dot executable is added to current PATH.

3.4 Download and Compile TIAA Source Code

The source code of TIAA can be downloaded at http://discovery.csc.ncsu.edu/software.html.

To compile TIAA, simply type `javac edu\ncsu\tiaa\gui\*.java` under the directory which contains downloaded source code.

4 How to Use TIAA

4.1 Key Concepts

This section explains some key concepts used in TIAA. For more detailed information, please refer to our technical report [5] and Yiquan’s Thesis.

4.1.1 Hyper-alert Type

A hyper-alert type $T$ is a triple $(\text{fact}, \text{prerequisite}, \text{consequence})$ where (1) fact is a set of attribute names, each with an associated domain of values, (2) prerequisite is the necessary condition for the attack, and (3) consequence is the possible outcome of an attack. Moreover, both prerequisite and consequence are represented in the form of logical formulas whose free variables are all in fact.

For example, a Sadmind_Amslverify_Overflow alert reported by RealSecure Network Sensor 6.0 [1] indicates a buffer overflow attack against the sadmind daemon. We can define a Sadmind_Amslverify_Overflow
4 HOW TO USE TIAA

A hyper-alert type for this kind of attack. The fact set will be \{VictimIP\}. The prerequisite can be formulated as \(\text{ExistHost(VictimIP)} \land \text{VulnerableSadmind(VictimIP)}\), while the consequence can be formulated as \(\text{GainRootAccess(VictimIP)}\).

Intuitively, the fact component of a hyper-alert type gives the information associated with the alert, prerequisite specifies what must be true for the attack to be successful, and consequence describes what could be true if the attack indeed happens.

4.1.2 Hyper-alert

A hyper-alert is an instance of an given hyper-alert type. It is a high-level intrusion alert based on raw alerts reported by IDSs. When created, a hyper-alert substitutes all attributes in fact set with actual values from raw alert. All logic formulas in prerequisite and consequence are evaluated using these values.

For example, if RealSecure Network Sensor reports a Sadmind_Amslverify_Overflow alert against IP address 172.16.112.50, we can create a hyper-alert with the Sadmind_Amslverify_Overflow hyper-alert type defined above. With the given raw alert attributes the new hyper-alert has the fact set of \{172.16.112.50\}, and its prerequisite now becomes \(\text{ExistHost(172.16.112.50)} \land \text{VulnerableSadmind(172.16.112.50)}\), and its consequence becomes \(\text{GainRootAccess(172.16.112.50)}\).

TIAA can convert raw alerts into hyper-alerts using customized Knowledge Base.

4.1.3 Knowledge Base

Knowledge Base is the "Hyper-alert Type Dictionary" where definitions of hyper-alert types are stored. It is customizable and expendable to work with different datasets and IDSs.

Knowledge Base is encoded in XML format. A sample segment for the Sadmind_Amslverify_Overflow mentioned above can be written as follows:

```xml
<hyper-alertType Name="SadmindOverflow">
  <Fact FactName="DestIPAddress" FactType="varchar(15)"/>
  <Fact FactName="DestPort" FactType="int"/>
  <Prerequisite>
    <Predicate Name="ExistHost">
      <Arg id="3" ArgName="DestIPAddress"/>
    </Predicate>
    <Predicate Name="VulnerableSadmind">
      <Arg id="22" ArgName="DestIPAddress"/>
    </Predicate>
  </Prerequisite>
  <Consequence>
    <Predicate Name="GainRootAccess">
      <Arg id="18" ArgName="DestIPAddress"/>
    </Predicate>
  </Consequence>
</hyper-alertType>
```
Please refer to Section 5 How to Write Knowledge Base File for more details on creating your own Knowledge Base.

4.1.4 Hyper-alert Collection

Hyper-alerts are organized into collections, which can be used as input of analysis utilities, or can be generated as the result of analysis.

For example, the initial hyper-alert collection contains all hyper-alerts converted from raw alert set generated by IDSs.

4.1.5 Analysis Utilities

Several interactive analysis utilities have been developed to refine the correlation results. These utilities can be divided into two categories: (1) hyper-alert generating utilities, including alert aggregation/disaggregation, clustering analysis, and focused analysis, and (2) feature extraction utilities, including frequency analysis, link analysis, and association analysis.

Here we only give brief descriptions of each utility. For more information, please refer to [5].

Focused Analysis

Intuitively, focused analysis is to concentrate on filtered alerts by specifying a focusing constraint. Focused analysis is particularly useful when we have certain knowledge of the alerts, the systems being protected, or the attacking computers. We expect an analyst to discover, or hypothesize and then verify, such knowledge while using the other utilities. For example, if we are concerned about the host at IP address 172.016.112.050, we can use this attribute as the focusing constraint to get all alerts targeting our host.

Clustering Analysis

Intuitively, clustering analysis is to partition a collection of hyper-alerts into different groups so that the hyper-alerts in each group share certain common features. One application of clustering analysis is to decompose a big hyper-alert collection into smaller ones with certain criteria, or clustering constraint in other words. Given two sets of attribute names $A_1$ and $A_2$, a clustering constraint $C_c(A_1, A_2)$ is a logical combination of comparisons between constants and attribute names in $A_1$ and $A_2$. A clustering constraint is a constraint for two hyper-alerts; the attribute sets $A_1$ and $A_2$ identify the attributes from two hyper-alerts $h_1$ and $h_2$, respectively. For example, we can set the clustering constraint as with the same source and destination IP addresses, or $(h_1.\text{SourceIP} = h_2.\text{SourceIP}) \land (h_1.\text{DestinationIP} = h_2.\text{DestinationIP})$ to be more formal. With such clustering constraint, our clustering analysis will do the partition accordingly.

Aggregation/Disaggregation Analysis

The purpose of alert aggregation is to generate a relatively concise view of current hyper-alert collection, while still keeping the structure of sequences of attacks informed by these hyper-alerts. By specifying a time interval constraint, all same type hyper-alerts happened within this interval will be aggregated together into one new hyper-alert of this hyper-alert type.

Disaggregation allows the user to expend an aggregated hyper-alert to display all hyper-alerts contained inside it. Thus users can inspect each hyper-alerts individually to gain more helpful information.
By combining aggregation and disaggregation, users can adjust the degree to which a hyper-alert correlation graph is to be reduced.

**Link Analysis**
Link analysis is intended to analyze the two-dimensional connection between hyper-alert attribute values. Examples include how two IP addresses are related to each other in a collection of alerts, and how IP addresses are connected to the alert types. Though link analysis takes a collection of hyper-alerts as input, it indeed analyzes the raw intrusion alerts corresponding to these hyper-alerts. Link analysis can identify candidate attribute values, evaluate their importance according to user defined metric, and rank them accordingly.

Link analysis can measure the connection among attribute values in either count mode or weighted mode. In count mode, once the attribute pair is specified (e.g., destination IP and destination port) the link measurement is simply the frequency of attribute value pairs (e.g., 172.016.112.050 and 80). In weighted mode, a more complex weighted method is adopted to quantify the link measurement.

If the specified attribute pair has the same domain (e.g., both are IP addresses or both are port numbers), we can treat these two attributes differently in link analysis. If an attribute value appears both as source IP/port and destination IP/port, we can treat them either as one single entity (i.e., unidomain link analysis) or two separate entities (i.e., dual-domain link analysis)

We will give a concrete example in Section 4.4.4 where we discuss how to apply link analysis.

### 4.1.6 GUI Components
The TIAA GUI is composed of four major parts, Main Window, Project Explorer, Workspaces and Log Panel.

**Main Window** Upon the starting of TIAA, the Main Window will be shown to the user as the primary graphical interface. In most cases, all analysis results and related information will be displayed in the Main Window. Some additional information might be generated on the run and can be displayed in Pop-up Windows and Floating Windows. All functionalities of TIAA can be accessed through Menus and Toolbars. Some frequently used menu items are included in the toolbars to allow fast access.

**Workspaces** One of the major objectives of GUI is to visualize the analysis results for users. For this reason, a majority of space is reserved to display graphical analysis results. All graphs are organized in the format of tabbed panels, allowing users to switch between different panels freely and conveniently.

Some graphical results are clickable, allowing users to query more detailed information if they are interested.

**Log Panel** Log Panel is used to display logging information and error messages.

**Project Explorer** Project Explorer is of great importance in helping users to browse analysis results and further apply appropriate analysis. It uses a tree structure to visualize the flow of analysis and to categorize different type of information. Upon the creating of a new analysis project, a node representing the project is created as the root node of the tree structure. Three immediate children are created as well at this time, representing knowledge base file, property file and raw alerts respectively.

**Task tree** is the sub-tree rooted at the raw alerts node. It provides an intuitive and structural view of all analysis applied. There are three types of nodes in the Task Tree. They are terminal nodes, alert collection nodes and utility nodes. Each of them represents a certain type of information stored in TIAA database, and has distinguishable image icon in the tree structure. Also, different types of nodes are subject of different type of operations.
The root node of task tree is a collection node representing the collection of raw alerts. Each collection node can be used as input and applied to any utilities provided. A utility node corresponding to the applied utility will be appended to the input collection node. The output of analysis utilities could be either collections of hyper-alerts, or visualized representation, represented by either collection nodes or terminal nodes. Resulting nodes are further appended to the utility node applied.

Collection node represents collection of hyper-alerts, which can be used as the input for all analysis utilities, or generated as the output of hyper-alert generating utilities. Such a node might be associated with more than one graph, depending on the utilities it has been applied to. For example, the correlation result of raw alert set might contain multiple graphs, identified by unique graph ID. To carry on another analysis, users can select one collection node by clicking it, and then choose from the list of available utilities applicable to the selected one. Terminal node represents the result of feature extraction utilities. Such a node can not be applied to any analysis utilities further more. Once clicked, the associated visualization file should be displayed in Workspaces. All files are in the format of html file. For example, the result of Link Analysis is visualized to a graph, and saved locally as an html file. All these information must be saved within one terminal node. Utility node connects input and output nodes, which could be either type of the other two types. It represents the utility has been applied and once clicked, all parameters specified for this analysis will be shown in the Log Window. For example, when applying correlation to the raw alert set, a new Utility Node is created as the child of raw alerts node. Once correlation finished, one or more Collection nodes might be generated and appended to the Utility Node as its child(ren).

**Pop-up Windows and Floating Windows** Pop-up Windows are used to display information requiring users’ immediate respond. Floating Windows are used to display more static information, allowing
users to switch focus between them and the main window.

4.2 Creating a New Project

Before starting using utilities, you need to create a new project to save your configurations and results.

1. Click File→New Project or New Project button to open the New Project dialog, as shown in Figure 2.

![New Project dialog](image)

Figure 2: Creating a New Project

2. Type in a name for your new project and specify your working directory by clicking the "Change" button. All analysis result files and save files will be stored under this directory.

3. Choose your knowledge base file by clicking "Browse" button.

4. Specify the JDBC driver and database URL will be used.

5. Enter the name of your working database name. You need to create such a database in your DBMS before creating a new project.

6. Choose preferred correlation method, either DBMS based or In-Memory correlation.
Figure 3: Creating a New Project (Cont’d)
7. Click Next to the next step (Figure 3).

8. Choose source of raw alerts.
   
   (a) If import from IDMEF file, specify IDMEF file and mapping file paths.
   
   (b) If import from database, specify corresponding database information such as JDBC driver, database URL, database name and login information. Also a mapping file is needed.

9. Click Finish to create your new project.

4.3 Open and Save a Project

1. Click File → Open Project to load a previously saved project.

2. Click File → Save Project to save current project to the default save.tiaa file under your working directory.

4.4 Applying Analysis Utilities

4.4.1 Focused Analysis

Here is an example of how to apply Focused analysis. Figure 4 shows the original correlation graphs generated from DAPRA 2000 dataset. By clicking on node Sadmin_AmSVerify_Overflow955 we can find its destination IP address: 172.016.112.050. Now let's use this IP address as our focused analysis constraint.

1. Right click on the collection node upon which you wish to apply focused analysis. Select Focused Analysis from the pop-up menu list (Figure 4).

2. Select "DestIPAddress" as attribute name and "=" as comparison operator. Enter "172.016.112.050" into attribute value field then click OK to start (Figure 5). You can specify up to three conditions and their combinations using "AND" or "OR". For IP address attributes, you can choose comparison operators from "=" "\neq" "<" "\leq" ">" and "\geq". If more complicated focusing constraints are desired, you can click "Add more" check box to manually enter more conditions.

3. Figure 6 shows the result of focused analysis. All hyper-alerts remaining here share the same destination IP address as we specified earlier in Step 2.

4.4.2 Clustering Analysis

Here is an example of how to apply Clustering analysis. Figure 7 shows the original correlation graphs generated from DAPRA 2000 dataset.

1. Right click on the collection node upon which you wish to apply Clustering analysis. Select Clustering Analysis from the pop-up menu list (Figure 7).

2. Select "with same source IP address" from pre-defined constraint list (Figure 8). You can specify up to three conditions and their combinations using "AND" or "OR". For more complicated clustering
Figure 4: Applying Focused Analysis

Figure 5: Applying Focused Analysis (Cont’d)
Figure 6: Applying Focused Analysis (Cont’d)

Figure 7: Applying Clustering Analysis
constraints, you can click “Add more” check box to manually input.

3. Figure 9 shows the clustering result. All remain hyper-alerts in this new graph share the same source IP address.
4.4.3 Aggregation/Disaggregation Analysis

Here is an example of how to apply aggregation analysis. Figure 10 shows one of correlation graphs from DAPRA 2000 dataset.

1. Right-click the collection node upon which you wish to apply aggregation analysis. Select Aggregation Analysis from the pop-up menu list (Figure 10).

2. Specify time interval for the analysis. A value of -1 means infinite interval. Chose aggregation level. Click OK to start (Figure 11).

3. Figure 12 shows the aggregation results. Since we use -1 as time interval in Step 2, all hyper-alerts with the same type are aggregated into one node.

Disaggregation can be only applied on aggregated hyper-alerts, which are the results of previous aggregation analysis. Taking Figure 12 as example, if we’d like to see all hyper-alerts contained in Sadmind_Amlverify_Overflow1010 we can disaggregate it by right-clicking on aggregated hyper alert and select "Disaggregation" from the pop-up menu. Figure 13 shows the disaggregation results. All aggregated hyper-alerts remains unchanged except for Sadmind_Amlverify_Overflow1010, which is expanded into four hyper-alerts.
Figure 11: Applying Aggregation Analysis (Cont’d)

Figure 12: Applying Aggregation Analysis (Cont’d)
4.4.4 Link Analysis

Here is an example of how to apply Link Analysis. Suppose that we want to see attribute connections of initial correlated hyper-alerts.

1. Right click "Correlated Alerts" and select "Link Analysis" from pop-up menu (Figure 14).
2. In the pop-up dialog, enter required parameters (Figure 15).
   - Enter logarithm base value. By default it’s set to be 2.
   - Enter threshold. By default it’s set to be 0.
   - Chose Link Analysis mode. Currently, only count mode is available.
   - Specify attribute pair. Here we choose destination IP address and source IP address.
   - Chose either uni-domain or dual-domain (Dual-domain is only selectable when two chosen attributes are in the same domain, i.e., both are IP addresses or both are port numbers). Here we choose dual-domain.
   - Click OK to start Link Analysis.

3. Figure 16 shows the analysis result. In the result graph, attribute values are represented as nodes. Since we chose dual-domain in step 2, two attributes are treated differently and represented in different shapes. Destination IP addresses are all of circle shapes, while source IP addresses are all in diamonds. The size of nodes indicates the weight of the corresponding attribute values. The bigger a node is, the more weight its attribute value has. Each node is clickable and detailed information regarding attribute
value represented by this node will display in the pop-up window. The link between two attribute values are represented by the color of the edge. A legend explaining each color’s weight will be shown when link analysis is finished. Or, you can right click any link analysis utility node to open legend for that analysis.

5 How to Write Knowledge Base File

The knowledge base contains the necessary information about hyper-alert types as well as relationships between predicates. To simplify the implementation, we assume that each hyper-alert type is uniquely identified by its name, and there is no negation in the prerequisite nor the consequence of any hyper-alert type.

In the XML file, there are basically three sections: Predicates, Implications and HyperAlertTypes.

In the Predicates section, the predicate name which works as the key of the predicate and the arguments of the predicate are specified. In order to make the validation strict and easier, a unique argument id is needed for each argument for all the predicates. Here is an example,

```xml
<Predicate Name="ExistService">
  <Arg id="14" Pos="1" Attr="varchar(15)"/>
  <Arg id="15" Pos="2" Attr="int"/>
</Predicate>
```

In this example, the predicate ExistService has two arguments, one is of char, the other is integer. Each of them should have an unique id. The Pos specifies that the char is the first argument and the integer is
Figure 15: Applying Link Analysis (Cont’d)
the second argument. The whole predicate is \texttt{ExistService(varchar, int)}.

Each predicate which appears in the \texttt{Implications} section or \texttt{HyperAlertTypes} section must be declared here. This part goes into the \texttt{Predicate} table in the database.

In the \texttt{Implications} section, there are two kinds of implications: normal and phantom. The phantom implications mean that the implied predicate is unclear to us, but it is clear to the attacker. For example, the consequence of \texttt{FTP\_Syst} is \texttt{GainOSInformation}. Through it, the attacker knows what operating system is in the target machine, either \texttt{OSLinux} or \texttt{OSWindows}, etc. But we, the correlator, cannot get this detailed information. So, we call this kind of implication, \texttt{GainOSInformation} implies \texttt{OSLinux}, phantom implication. Phantom implications and normal implications have the same effect in the correlation process. Having phantom can correlate more related alerts, which may be missed otherwise; however, it may also increase the false correlation rate.

Here is an example of implication:

```
<Implication Phantom="Yes">  
<ImplyingName>GainOSInfo</ImplyingName>  
<ImpliedName>OSLinux</ImpliedName>  
<ArgMap>  
<ImplyingArg id="23"/>  
<ImpliedArg id="39"/>  
</ArgMap>  
</Implication>
```
It represents $GainOSInfo(GainOSInfoArg)$ implies $OSLinux(OSLinuxArg)$. The $ImplyingName$ and $ImpliedName$ are obvious. $ArgMap$ represents the argument mapping relationship. In this example, the argument id of $GainOSInfoArg$ defined in the $Predicates$ section is 23 and the argument id of $OSLinuxArg$ is 39. They should match the id which is defined in the $Predicates$ section.

This part goes into the $Implication$ table in the database.

The hyper alert types are defined in the $HyperAlerTypes$ section. Each $HyperAlertType$ may consist several parts: $Fact$, $Prerequisite$ and $Consequence$ if it has. Among them, $Fact$ is a must of a hyper alert type; $Prerequisite$ and $Consequence$ are optional. The $HyperAlerTypes$ section will be mapped into several tables: $HATFact$ to store the fact information, $HATPrereq$ to store the prerequisite information and $HATConseq$ to store the consequence information.


We provide a module to parse this knowledge base XML file and put them into database.

6 Acknowledgement

This work is partially supported by the U.S. Army Research Office under grant DAAD19-02-1-0219, by the National Science Foundation under grant CCR-0207297, and by NCSU CACC.

7 Contributors

Dr. Ning, Peng
Cui, Yun
Hu, Yiquan
Peng, Pai
Xu, Dingbang

References


