TIAA: A Toolkit for Intrusion Alert Analysis (Version 1.0) Installation and Operation Manual

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, a 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, and a DBMS (such as Microsoft SQL Server 2000) is necessary to manage alerts and store analysis results. Also the Xerces Java Parser is needed to parse Knowledge Base, GraphViz is needed to visualize analysis results, and Ethereal is needed to analyze the tcpdump files.

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
- Microsoft SQL Server 2000 and corresponding JDBC Driver
- Database of raw alerts generated by IDS, and the related Knowledge Base
- Xerces Java Parser v2.7.1
- AT&T GraphViz
- Ethereal

3 Installing TIAA

3.1 Prepare JDBC Connection

All raw alerts and analysis results are under the management of one Database Management System. In general, any DBMS with JDBC capability can be chosen to work with TIAA. However, due to the difference between SQL syntaxes supported by different DBMSs, minor revisions on SQL statments might be needed for certain DBMS. In TIAA 0.5, the default DBMS is Microsoft SQL Server 2000.

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
In TIAA 0.4, the default JDBC driver is Microsoft SQL Server 2000 JDBC driver.

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

3.2 Xerces Java Parser

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

3.3 AT&T GraphViz

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 Ethereal

Download Ethereal from http://www.ethereal.com/. In TIAA, Ethereal is used to analyze the tcpdump file, and the users can save the analysis result (packet summary) into a text file.

3.5 Download TIAA Package

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

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 technique papers [2–6].

4.1.1 Hyper-alert Type

A *hyper-alert type T* is a triple (*fact*, *prerequisite*, *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*

hyper-alert type for this kind of attack. The *fact* set will be {VictimIP}. The *prerequisite* can be formulated as *ExistHost(VictimIP)* $\wedge$ *VulnerableSadmind(VictimIP)*, while the *consequence* can be formulated as *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 a 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 *ExistHost(172.16.112.50)* $\wedge$ *VulnerableSadmind(172.16.112.50)*, and its *consequence* becomes {*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 extendable 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:

```
<hyper-alertType Name="SadmindOverflow">
  <Fact FactName="DestIPAddress" FactType="varchar(15)"></Fact>
  <Fact FactName="DestPort" FactType="int"></Fact>
  <Protocol ProtocolName="RPC"></Protocol>
  <Protocol ProtocolName="SADMIND"></Protocol>
  <Prerequisite>
     <Predicate Name="ExistHost">
      <Arg id="3" ArgName="DestIPAddress"></Arg>
    </Predicate>
    <Predicate Name="VulnerableSadmind">
      <Arg id="22" ArgName="DestIPAddress"></Arg>
    </Predicate>
  </Prerequisite>
  <Consequence>
    <Predicate Name="GainRootAccess">
      <Arg id="18" ArgName="DestIPAddress"></Arg>
    </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 the 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*, *focused analysis*, and *missed attack hypotheses*, and (2) *feature extraction utilities*, including *frequency analysis* (not integrated), *link analysis*, *association analysis*, and *attack strategy extraction*.

Here we only give brief descriptions of each utility. For more information, please refer to our technique papers.

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 $(h_1.SourceIP = h_2.SourceIP) \wedge (h_1.DestinationIP = h_2.DestinationIP)$ 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. Aggregation can also be performed through a user-defined abstraction hierarchy, which defines hyper-alert types in different abstraction levels.

Disaggregation allows the user to expand 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., *uni-domain 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.

Association Analysis

Association analysis is used to find out frequent co-occurrences of values belonging to different attributes that represent various alerts. For example, we may find through association analysis that many attacks are from source IP address 172.16.1.10 to destination IP address 172.16.1.31 at destination port 80.

Syntactically, association analysis takes a set S of categorical alert attributes and a support threshold t as parameters. Similar to link analysis, association analysis can be applied in both count mode and weighted analysis mode. In the count mode, given a set X of attribute values, the *support* of X in an alert set T is $s\%$ denoting that $s\%$ of the alerts in T contain attribute values X. In the weighted mode, association analysis requires a numerical attribute (also called a weight attribute) as an additional parameter. To facilitate the weighted analysis mode, we extend the notion of support to *weighted support*. Given a set X of attribute values and a weight attribute w, the *weighted support of X w.r.t. w* in the alert set T is

$$\text{weighted support}_w(X) = \frac{\text{sum of } w \text{ of all alerts in } T \text{ that contain } X}{\text{sum of } w \text{ of all alerts in } T}.$$

Thus, association analysis of a collection of alerts in count mode finds all sets of attribute values that have support more than t, while association analysis of a collection of alerts in weighted analysis mode returns all sets of attribute values that have weighted support more than t.

Attack Strategy Extraction

Attack Strategy Extraction is used to find attack strategies from correlation graphs. Attack strategies are represented by attack strategy graph. Informally, an attack strategy graph is a directed graph, where each node is a hyper-alert type, and each directed edge represents the equality constraints that the related nodes should satisfy.

Missed Attack Hypotheses

It is well-know that intrusion detection systems (IDSs) may fail to detect some attacks. The utility of missed attack hypotheses aims at hypothesizing possible un-detected attacks thus provoding better results than IDSs.

The key concept which can help hypothesize missed attacks is *equality constraints*, which captures the equality relations between consecutive hyper-alerts in correlation graphs. For example, a *Sadmind_Ping* alert and a *Samind_Amslverify_Overflow* alert always have the same destination IP address if the first one prepares for the second one. Based on equality constraints, we can hypothesize missed attacks, filter out incorrect hypotheses through raw audit data, and consolidate hypothesized attacks to get concise attack scenarios.

4.1.6 GUI Components

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

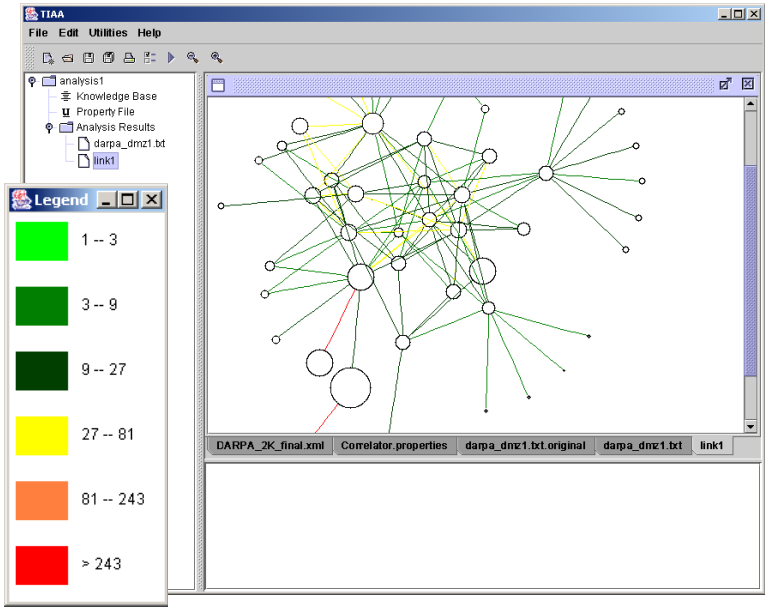


Figure 1: Major Components of TIAA GUI

Main Window Upon the starting of TIAA, the Main Window will be showed 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 will 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 loging 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 types of information. Upon the creation 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 analyses 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 types 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 an 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 alter 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 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 response. 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 an analysis 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.
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 which will be used.
5. Enter the name of your working database name. You need to create such a database in your DBMS before creating new project.
6. Choose preferred correlation method, either DBMS based or In-Memory correlation.
7. Enter preferred time interval for sliding window (The default value is -1, which means infinity. This time interval is usually used with the in-memory based correlation. For DBMS based method, usually we set it -1.).
8. Filter out uninteresting hyper-alert types through entering the type names (can be empty)
9. Click Next to the next step (Figure 3).
10. Choose source of raw alerts.
    (a) If import from IDMEF file, specify IDMEF file and mapping file paths.

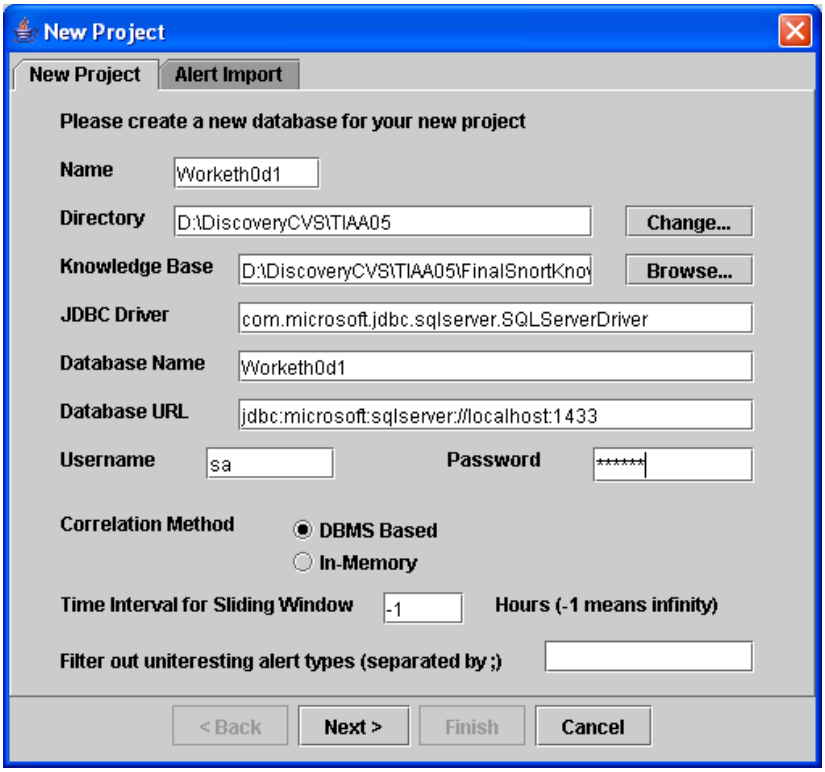


Figure 2: Creating a New Project

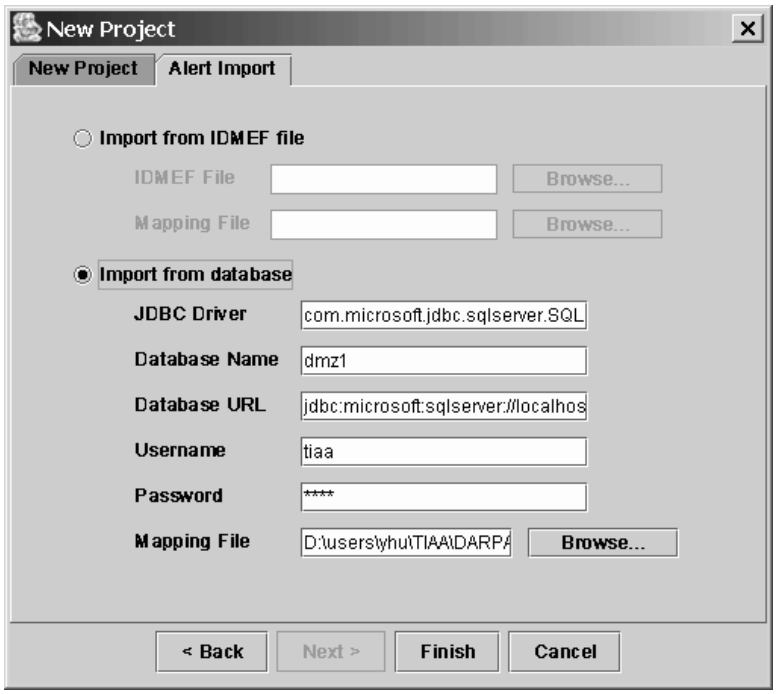


Figure 3: Creating a New Project (Cont'd)

(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.

11. 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 Sadmind_Amslverify_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** form the pop-up menu list (Figure 4).
2. Select "DestIPAddress" as attribute name and "=" as comparison operator. Enter "172.016.112.050" into

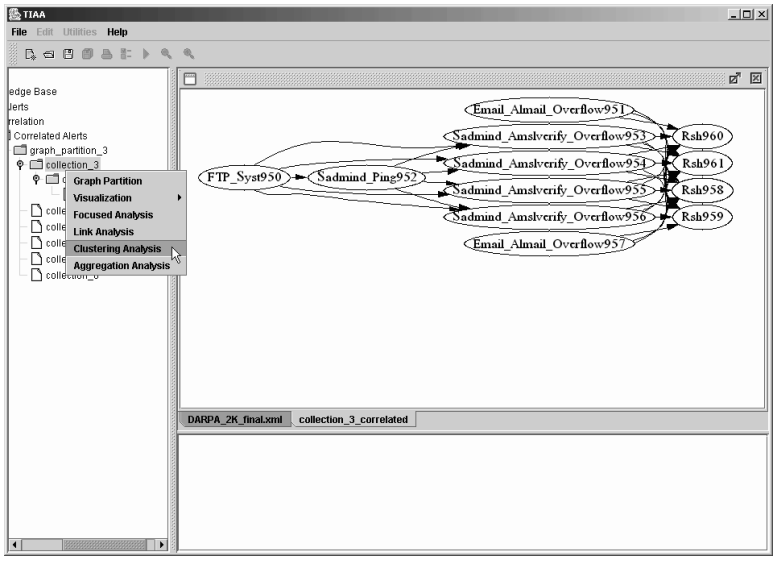


Figure 4: Applying Focused Analysis

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 "=", "≠", and "LIKE". With "LIKE" specified, you can input subnet information instead of exact IP addresses. For example, "DestIP LIKE 172.016.112.%" means all hosts within subnet of "172.016.112". As for port number attributes, you can choose among "=", "≠", "<", "≤", ">", and "≥". If more complicated focusing constraints are desired, you can click "Add more" check box to manually enter more conditions.

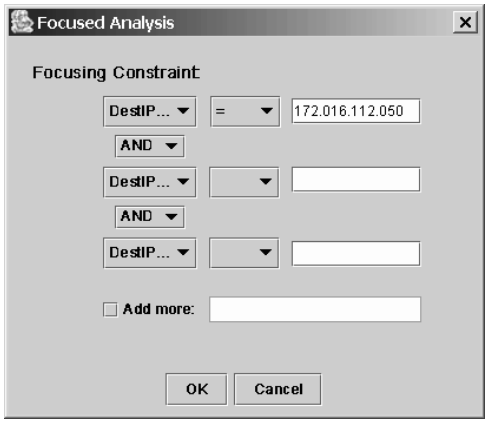


Figure 5: Applying Focused Analysis (Cont'd)

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

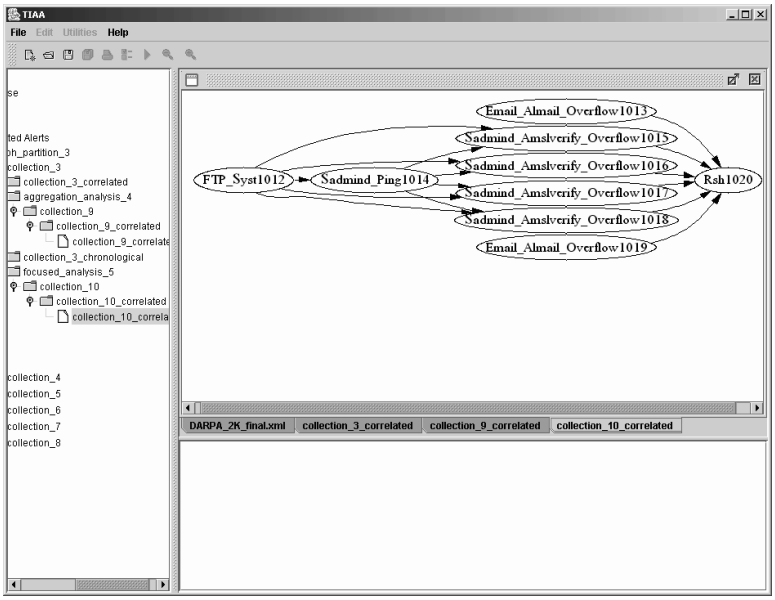


Figure 6: Applying Focused Analysis (Cont'd)

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** form 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

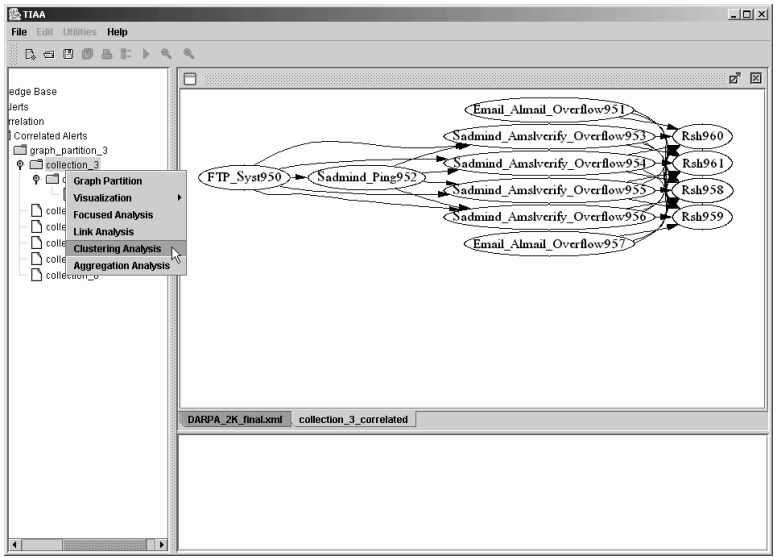


Figure 7: Applying Clustering Analysis

three conditions and their combinations using “AND” or “OR”. For more complicated clustering 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

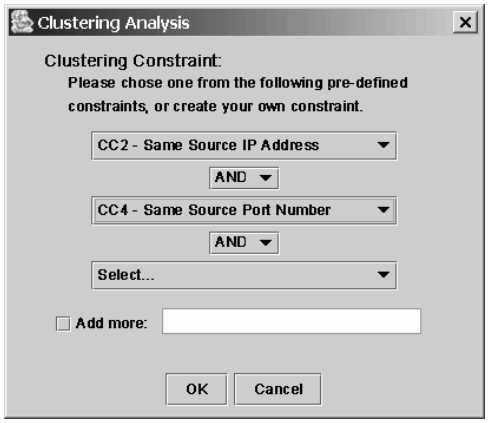


Figure 8: Applying Clustering Analysis (Cont’d)

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).

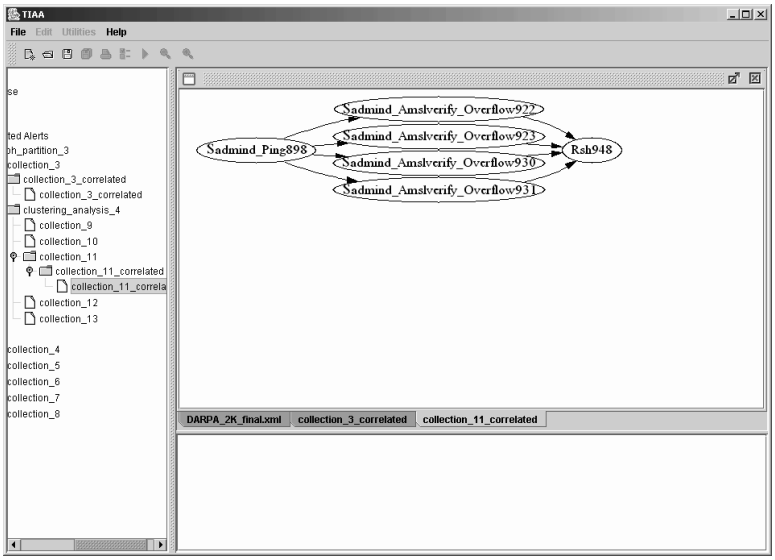


Figure 9: Applying Clustering Analysis (Cont'd)

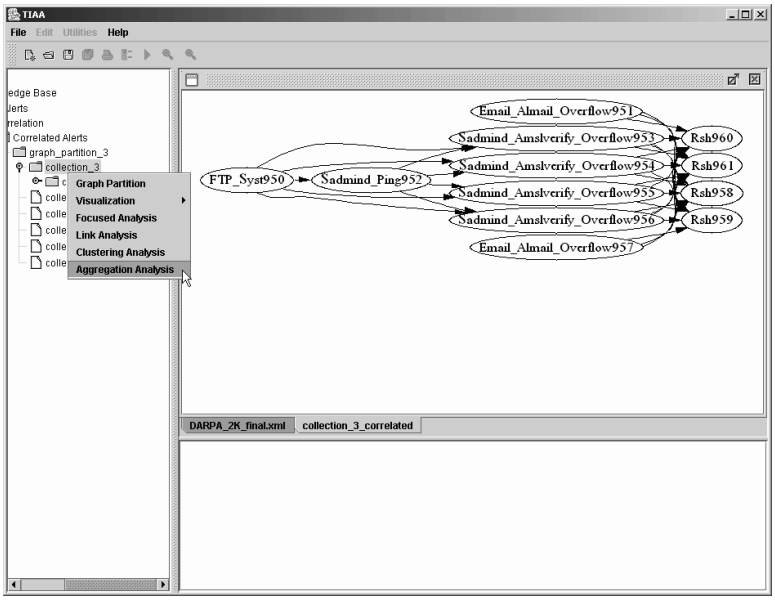


Figure 10: Applying Aggregation Analysis

2. In the pop-up dialog, enter the required parameters: time interval, which defines the time constaint (A value of -1 means infinite time interval), abstraction file, which is a xml file defining abstraction hierarchy between hyper-alert types, and the aggregation level, which defines the abstraction level along the hierarchy tree (The level of root node is 1, and more specific the hyper-alert type is, the higher the abstraction level is. For example, if from the root node to the leaf nodes, there exists a hyper-alert type path: "Attack" → "SerivePing" → "Sadmind_Ping", then the level of "Attack" is 1, "SerivePing" is 2, and "Sadmind_Ping" is 3). The input dialog is shown is Figure 19. If the user defines an abstraction hierarchy, then input the file path either through "Browse" button, or typing it in the text field. If the user does not define an abstraction hierarchy, simply make the text field empty. Click OK to start (Figure 11).

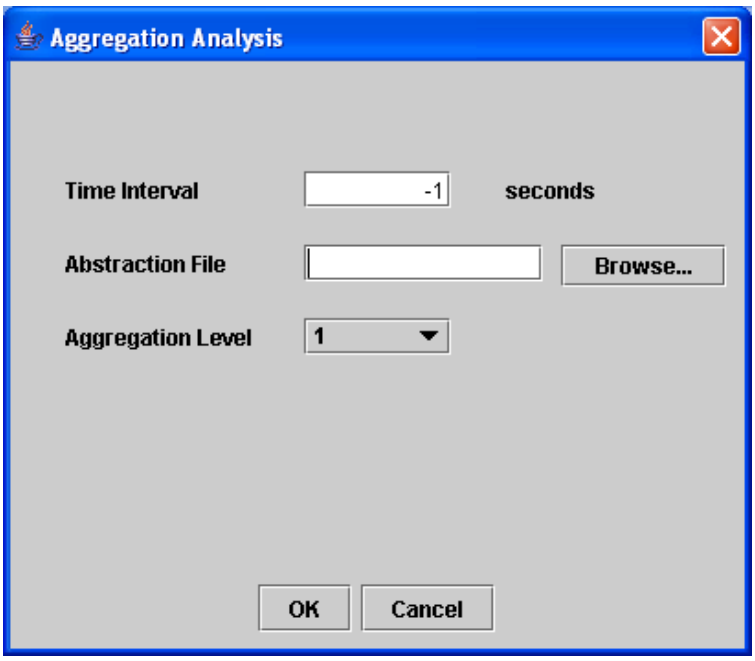


Figure 11: Applying Aggregation Analysis (Cont'd)

3. Figure 12 shows the aggregation results. Since we use -1 as time interval, 1 as aggregation level, and without abstraction file 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 an example, if we'd like to see all hyper-alerts contained in *Sadmind_Amslverify_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_Amslverify_Overflow1010*, which is expanded into four hyper-alerts.

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.

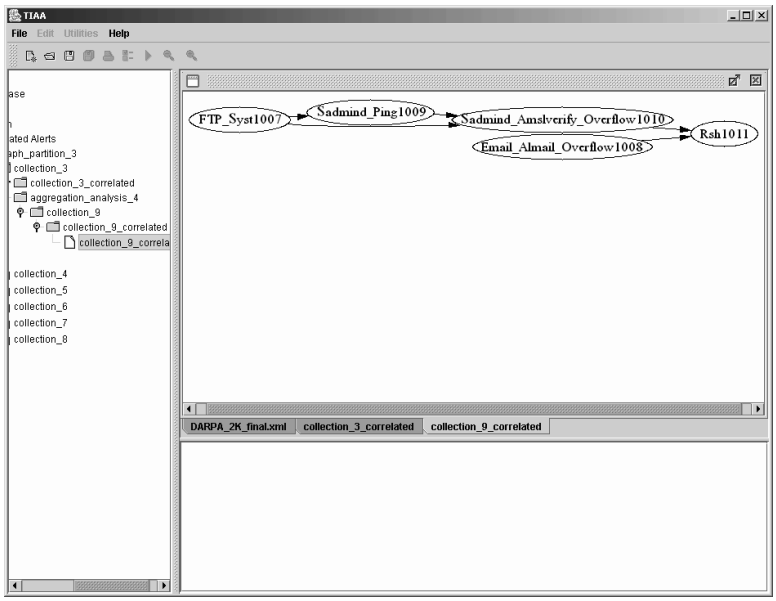


Figure 12: Applying Aggregation Analysis (Cont'd)

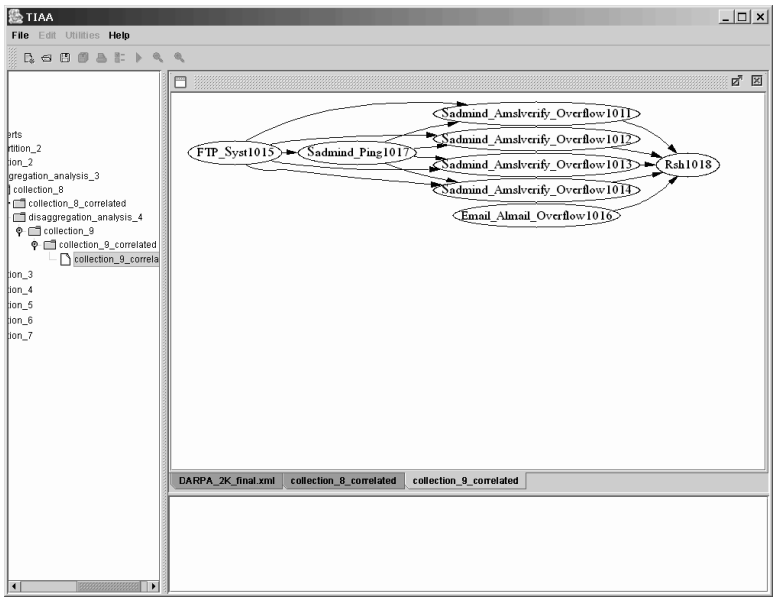


Figure 13: Applying Disaggregation Analysis

1. Right click “Correlated Alerts” and select “Link Analysis” from pop-up menu (Figure 14).

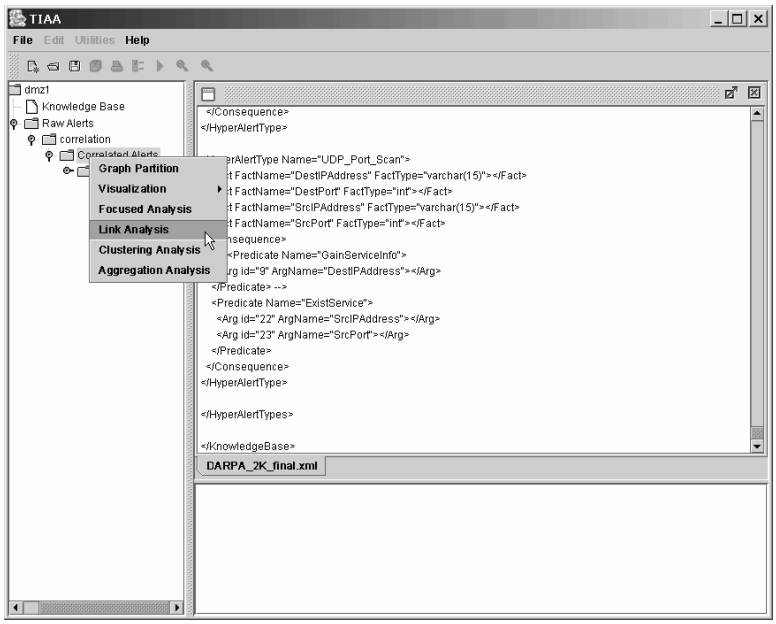


Figure 14: Applying Link Analysis

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.
- Choose Link Analysis mode. Currently, only count mode is available.
- Specify attribute pair. Here we choose destination IP address and source IP address.
- Choose 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 choose 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.

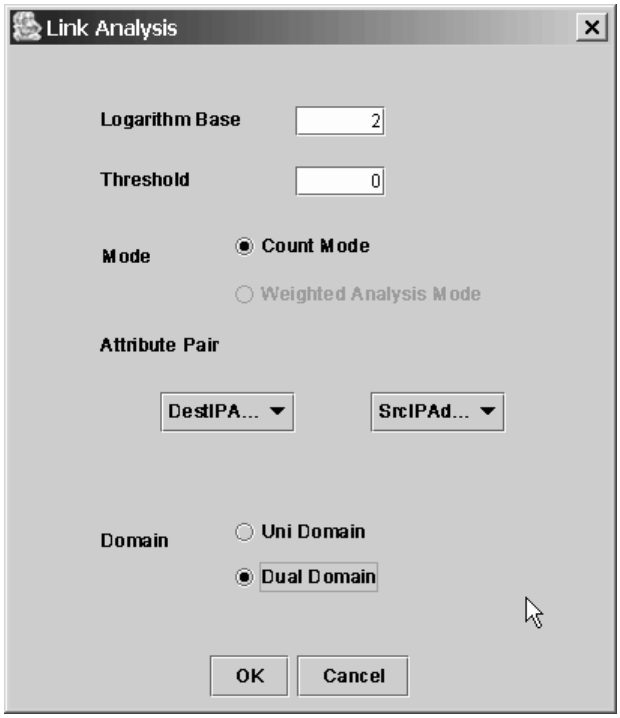


Figure 15: Applying Link Analysis (Cont'd)

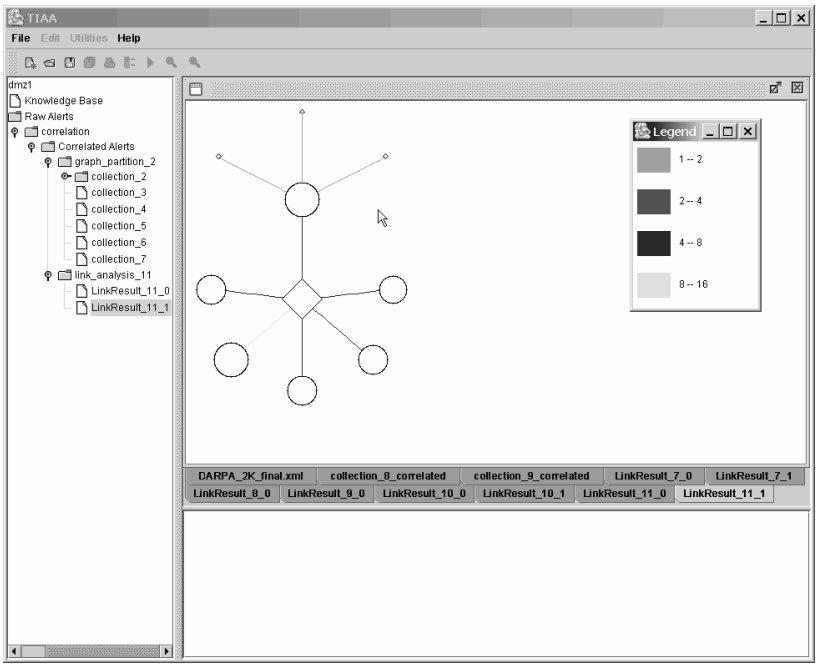


Figure 16: Applying Link Analysis (Cont'd)

4.4.5 Association Analysis

We present an example to show how to apply association analysis. Assume the alert dataset is from inside 1. After corrlation, we get one correlation graph, which is represented by collection_2. Suppose that we want to see attribute connections of the alerts in collection_2.

1. Right click "collection_2" and select "Association Analysis" from pop-up menu.

2. In the pop-up dialog, enter the required parameter: support threshold (Figure 17). This threshold should be a value between 0 to 100 (e.g., 30).

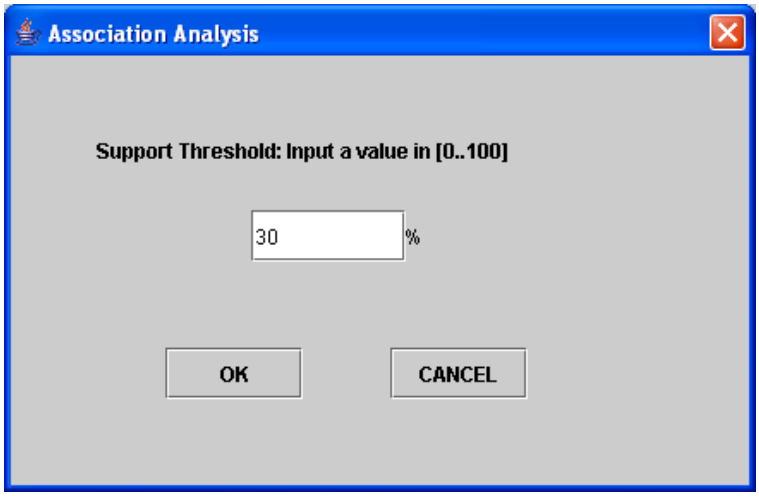


Figure 17: The Input Dialog of Association Analysis

3. Figure 18 shows the analysis result. Each row in the figure represents a frequent attribute set and the related support. For example, the last row means in collection_2, there are 31.818181818181817% of all alerts that their hyper-alert type is Sadmind_Amslverify_Overflow and their source IP address is 202.077.162.213.

4.4.6 Attack Strategy Extraction

We present an example to show how to extract attack strategies. Suppose that we want to extract attack strategies from collection_2.

1. Right click "collection_2" and select "Extract Strategy" from pop-up menu.

2. In the pop-up dialog, enter the required parameters: abstraction file and aggregation level, which are explained earlier.

3. Figure 20 shows two different results depending on the abstraction hierarchy. In Figure 20(a), we do not give any abstraction files. And in Figure 20(b), we provide a user-defined abstraction hierarchy file, and choose aggregation level with 2. These two graphs help us understand the adversaries' attack strategies.

4.4.7 Missed Attack Hypotheses

We present an example to show how to perform missed attack hypotheses.

For convenience and comparison purpose, in the utility of missed attack hypothese, we also provide users an option to partition correlation graphs through dropping one hyper-alert type. After dropping, a correlation graph may be partitioned into several pieces. We can integrate these pieces through missed attack hypotheses.

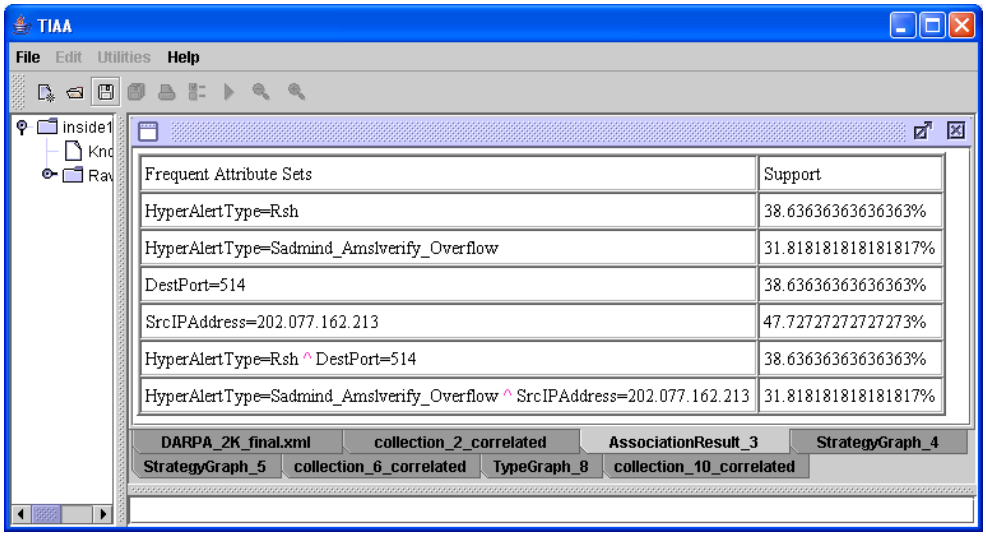


Figure 18: An Example Output of Assocation Analysis

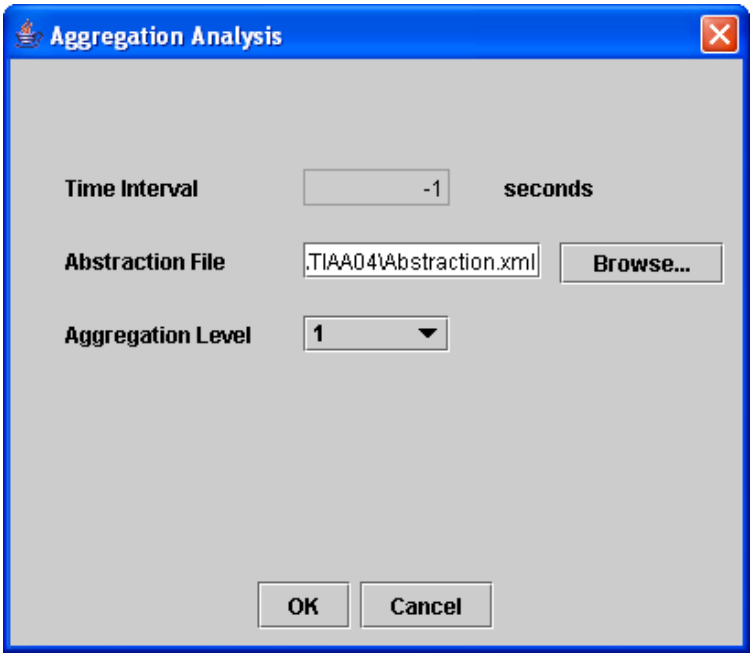
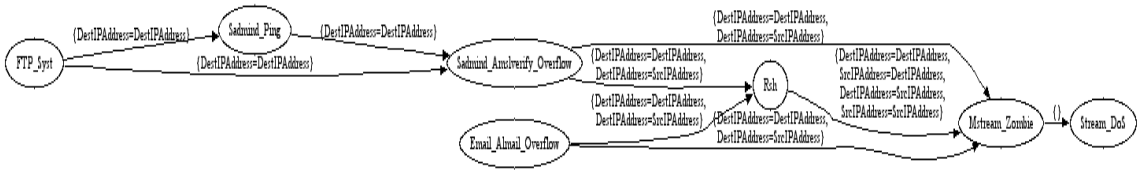
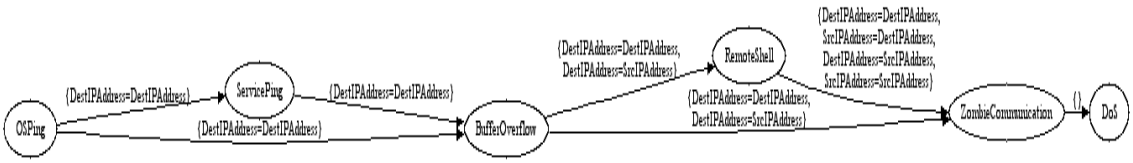


Figure 19: The Input Dialog of Attack Strategy Extraction



(a) Attack Strategy Graph without Abstraction Hierarchy



(b) Attack Strategy Graph with an Abstraction Hierarchy

Figure 20: Two Attack Strategy Graphs

In the following, we first present how to partition a correlation graph through dropping a hyper-alert type, then we talk about missed attack hypotheses. Suppose that we want to partition collection_2 through dropping "Sadmind_Amslverify_Overflow", and then we can integrate the remaining alerts in possible multiple correlation graphs.

1. Right click "collection_2" and select "Attack Hypotheses" from pop-up menu.

2. In the pop-up dialog (shown in Figure 21), click the radio button "Drop one hyper-alert type in the collection first", input the hyper-alert type name (e.g., "Sadmind_Amslverify_Overflow"), then click "OK" button. After this, we can get four collections (e.g., collection_6, collection_7, collection_8, collection_9). The four corresponding correlation graphs are shown in Figure 22.

Our following job is to integrate collection_6, collection_7, collection_8, and collection_9. We list the steps to integrate them as follows.

1. Right click "collection_6" and select "Attack Hypotheses" from pop-up menu.

2. In the pop-up dialog (shown in Figure 21), click the radio button "Integrate collections", then input all the necessary parameters. For "Specify other Collection IDs (separated by ,)", input all collection IDs except the collection node being selected. Here we input "7,8,9", denoting collection 6 will be integrated with collection 7, 8 and 9. If the user wants to output the type graph, check "Output Type Graph" checkbox. When TIAA makes attack hypotheses and filters out incorrect hypotheses, it needs audit information (e.g., the packets' source, destination IP addresses and related protocols) to help. Users can analyze the tcpdump files through Ethereal and save the packet summary information into a text file. TIAA can import the packet summary information into the database and save it for later analysis. Users can import packet summary information through specifying the correspoinding text file at "Audit Analysis File". Since importing these information is time-consuming, if the packet summary information is already saved in the database, users can "Use the pre-stored audit information for analysis". IDSs can generate alerts through analyzing the tcpdump file, or monitoring the replayed traffic (in an isolated network). For the replayed traffic, there may exist inconsistency between detection time and frame arrival time, thus it is necessary to specify the replay time to solve this inconsistency. If using our sample alert datasets (inside1 dataset and dmz1 dataset), the replay time for inside1 dataset is "2001-11-10 03:59:55", and the replay time for dmz1 dataset is "2001-11-09 23:06:18". Here assume that we want to output the type graph, we also specify the audit analysis file, un-check the check box "Use the pre-stored audit information for analysis", and specify the inside1 traffic

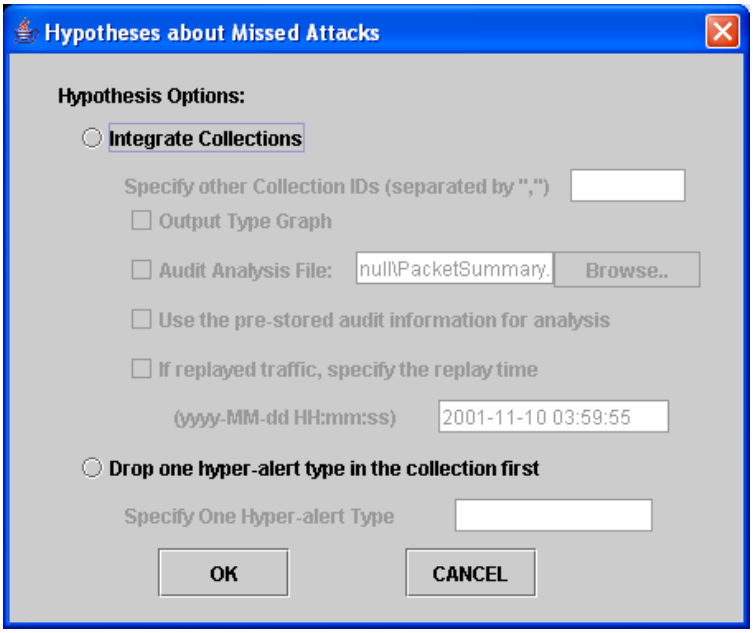


Figure 21: The Input Dialog of Missed Attack Hypotheses

replay time. After all these parameters are set, click "OK" button.

3. Figure 23 shows the example type graph, and Figure 24 shows the integrated correlation graph after missed attack hypotheses for collection_6, collection_7, collection_8, and collection_9.

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*.

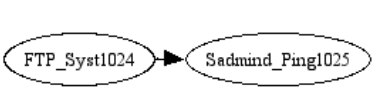
In the *Predicates* section, the predicate name which works as the key of the predicate and the arguments of the predicate are specified. In order to make the validation strict and easier, a unique argument id is needed for each argument for all the predicates. Here is an example,

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

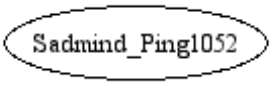
In this example, the predicate *ExistService* has two arguments, one is character, the other is integer. Each of them should have an unique id. The *Pos* specifies that the char is the first argument and the integer is the second argument. The whole predicate is *ExistService(varchar, int)*.

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

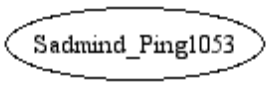
In the *Implications* section, there are two kinds of implications: normal and phantom. The phantom implications mean that the implied predicate is unclear to us, but it is clear to the attacker. For example, the



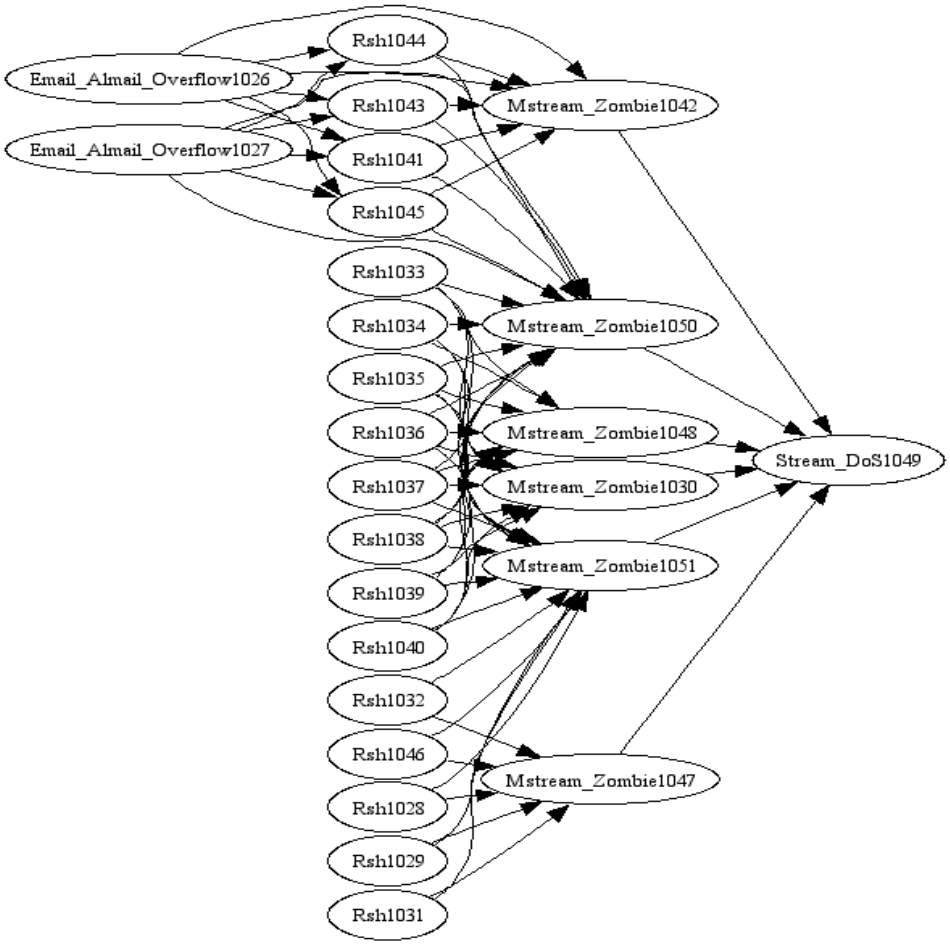
(a) Collection_6_Correlated



(b) Collection_8_Correlated



(c) Collection_9_Correlated



(d) Collection_7_Correlated

Figure 22: Four Correlation Graphs after Dropping One Hyper-alert Type

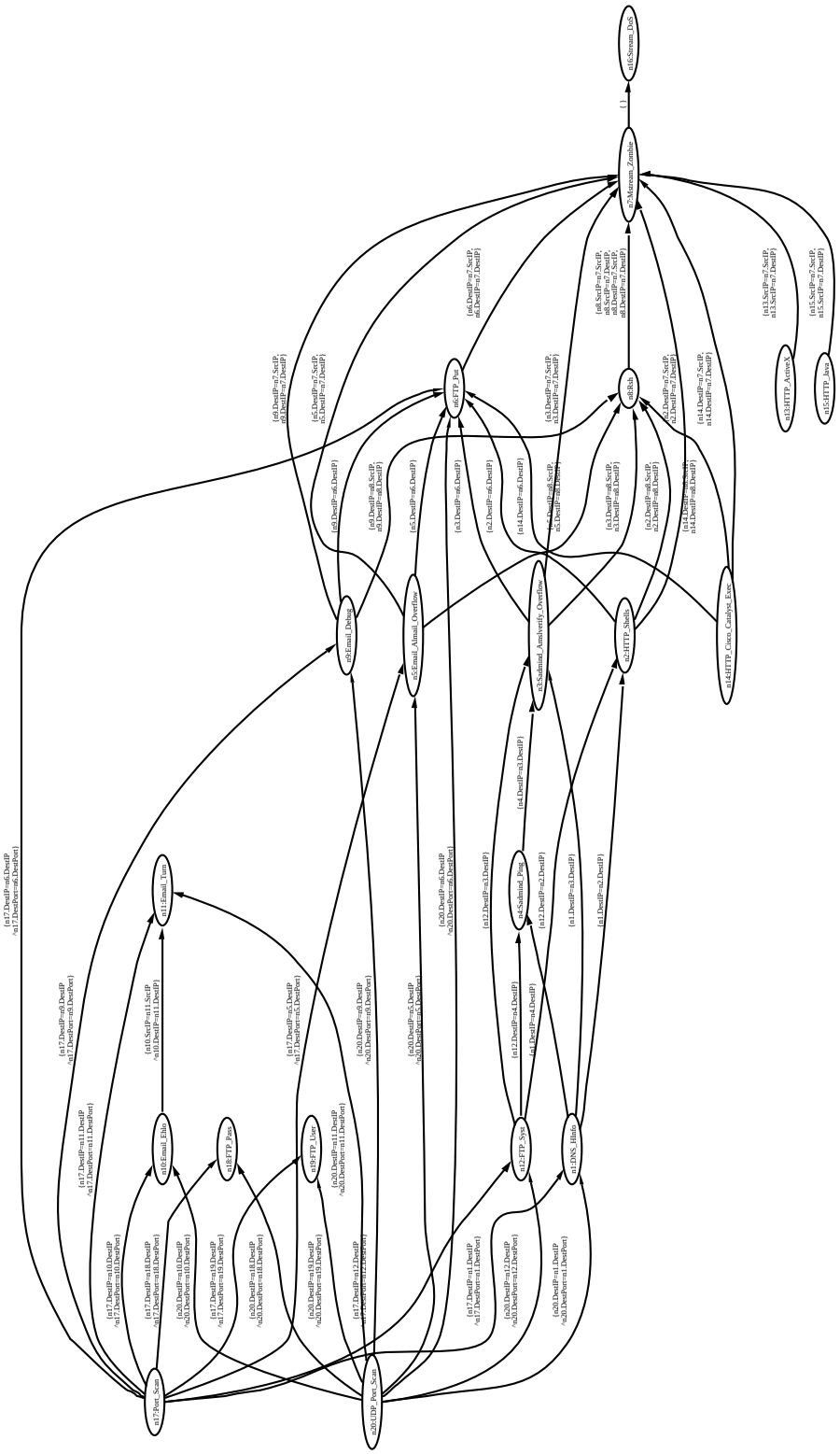


Figure 23: An Example Type Graph

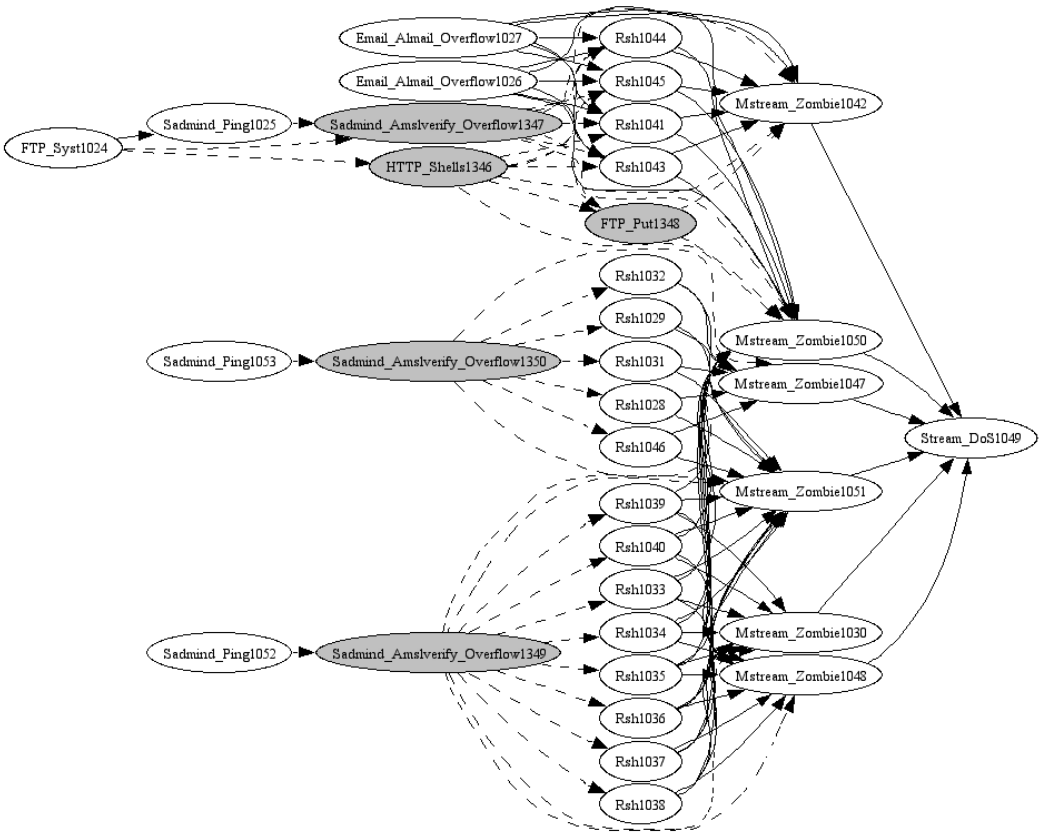


Figure 24: An Example Output of Missed Attack Hypotheses

consequence of *FTP_Syst* is *GainOSInformation*. Through it, the attacker knows what operating system is in the target machine, either *OSLinux* or *OSWindows*, etc. But we, the correlator, cannot get this detailed information. So, we call this kind of implication, *GainOSInformation* implies *OSLinus*, 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"></ImplyingArg>
<ImpliedArg id="39"></ImpliedArg>
</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* secion 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*, *Protocol*, *Prerequisite* and *Consequence* if it has. Among them, *Fact* is a must of a hyper-alert type; *Prerequisite* and *Consequence* are optional. In order to help missed attack hypotheses, we also associate *Protocol* with each hyper-alert type. Basically, *Protocol* tells us the protocols over which the corresponding attack occurs. The *HyperAlertTypes* section will be mapped into several tables: *HATFact* to store the fact information, *HATProtocol* to store the protocol information, *HATPrereq* to store the prerequisite information and *HATConseq* to store the consequence information.

Please refer to our papers [2, 3] for detailed table structures.

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

6 Acknowledgement

This work is partially supported by the National Science Foundation (NSF) under grants CCR-0207297 and ITR-0219315, by the U.S. Army Research Office (ARO) under grant DAAD19-02-1-0219, and by the NCSU Center for Advanced Computing and Communication (CACC).

7 Main Contributor

Xu, Dingbang

8 Other Contributors

Dr. Ning, Peng
Cui, Yun
Hu, Yiquan
Mahalati, Jaideep
Peng, Pai

References

[1] Internet Security Systems. RealSecure intrusion detection system. `http://www.iss.net`.

[2] P. Ning, Y. Cui, and D. S Reeves. Analyzing intensive intrusion alerts via correlation. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, pages 74–94, Zurich, Switzerland, October 2002.

[3] P. Ning, Y. Cui, and D. S Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 245–254, Washington, D.C., November 2002.

[4] P. Ning, Y. Cui, D. S. Reeves, and D. Xu. Tools and techniques for analyzing intrusion alerts. *ACM Transactions on Information and System Security*, 7(2):273–318, May 2004.

[5] P. Ning and D. Xu. Learning attack stratagies from intrusion alerts. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, pages 200–209, October 2003.

[6] P. Ning, D. Xu, C. Healey, and R. St. Amant. Building attack scenarios through integration of complementary alert correlation methods. In *Proceedings of the 11th Annual Network and Distributed System Security Symposium (NDSS '04)*, pages 97–111, February 2004.