

User Installation & Operation Manual

November 26, 2002

1 Introduction

This tool takes the alerts generated by Intrusion Detection System(IDS) as input and applies our correlation techniques upon them. The false alerts will be filtered out by the tool and a set of correlation graphs will be generated to show the attack scenarios hiding behind the alerts.

In a word, this tool aims to assist human user in analyzing the potentially large amount of intrusion alerts and uncovering the high-level attack strategies.

This tool is written using Java, with JDBC to access the database. The GraphViz package [1] is adopted to visualize the results.

The techniques that this tool was based on can be found in [4] and [5].

2 Installation

2.1 System Environment

This tool is developed with Java 2. A relational database is a must to use this tool. In order to support knowledge base generation, Xerces Java Parser is also needed. We use Java 2 SDK Standard Edition v1.3.1_04, and Xerces Java Parser v1.4.4 [3].

Our tests were conducted on Windows 2000 with Microsoft SQL Server 2000. The original alerts were generated by RealSecure Network Sensor 6.0 [2].

2.2 Checklist

Here is the list you need to run this tool.

- Java
- Database
- JDBC Driver
- Raw alerts generated by IDS which are stored in database
- Knowledge Base

- Xerces Java Parser v1.4.4
If you haven't a knowledge base ready, and you want to generate it through an XML file, you need it.
 1. Go to <http://xml.apache.org> and download a Xerces Java Parser;
 2. Install it and make sure the package is in your classpath;
- GraphViz
GraphViz is necessary to visualize the correlation results.

2.3 Download

The software package can be downloaded at <http://discovery.csc.ncsu.edu/software>. A sample knowledge base, a sample test database, and instructions to repeat our experiments are also available on the above page.

3 How to use this tool

The core of this tool consists of a knowledge base, an alert preprocessor, a correlation engine, a hyper-alert correlation graph generator, and three useful utilities: adjustable graph reduction, focused analysis and graph decomposition.

3.1 Knowledge Base

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 of char, the other is integer. Each of them should have a 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 consequence of *FTP_Syst* is *GainOSInfo*. 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, *GainOSInfo* implies *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">
<ImpliedName>GainOSInfo</ImpliedName>
<ImpliedName>OSLinux</ImpliedName>
<ArgMap>
<ImpliedArg id="23"></ImpliedArg>
<ImpliedArg id="39"></ImpliedArg>
</ArgMap>
</Implication>
```

It represents *GainOSInfo*(*GainOSInfoArg*) implies *OSLinux*(*OSLinuxArg*). The *ImpliedName* 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 *HyperAlertTypes* 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 *HyperAlertTypes* 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.

Please refer to our papers [4] [5] for detailed table structure.

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

3.2 Property File

There is a property file named "Correlator.properties". Here is an example:

```
#section 1: database connection
dbDriver = com.microsoft.jdbc.sqlserver.SQLServerDriver
dbURL = jdbc:microsoft:sqlserver://balisong:1433;DatabaseName=defcon8-vol1;SelectMethod=cursor

#newTable = insert into events select distinct e.sid, e.cid, e.signature, s.sig_name,
e.timestamp, i.ip_src, i.ip_dst, t.tcp_sport, t.tcp_dport from event e, signature s,
iphdr i, tcphdr t where e.signature=s.sig_id and e.sid=i.sid and e.cid=i.cid and
e.sid=t.sid and e.cid = t.cid;
```

```

#section 2: knowledge base
Generate_Knowledge_Base = true
Knowledge_Base_XML_File = DARPA_2K_final.xml

#section 3: correlation engine
AlertTable = events
# column names of RealSecure & MSSQL
# the following 4 are static keywords
AlertID = EventID
AlertName = OrigEventName
BeginTime = EventDate
EndTime = EventDate
# the following are "dynamic" keywords based on knowledge base
SrcIPAddress = SrcIPAddress
SrcPort = SrcPort
DestIPAddress = DestIPAddress
DestPort = DestPort

Original_Graph_Output = darpa_dmz1.txt

#Below are the three utilities
#section 4: graph reduction
Aggregation_Time_Interval = -1
Graph_Reduction_Output = dmz_a.txt

#section 5: focused analysis
Focused_Analysis_GraphID = 9
Focusing_Constraint = SrcIPAddress='152.14.53.39'
Focused_Aggregation = true
Focused_Aggregation_Time_Interval =
Focused_Output =

#section 6: graph decomposition
Decomposition_ID = 9
Clustering_Constraint = h1.SrcIPAddress=h2.SrcIPAddress
Decomposition_Aggregation = true
Decomposition_Aggregation_Time_Interval =
Decomposition_Output =

#section 7: mapping between original alert name and hyper alert name (OP-
TIONAL)
original_alert_name = hyper_alert_name

```

You can specify the parameter needed to communicate with database in section 1. The *dbDriver* is used as the jdbc driver name and the *dbURL* is used as the database url.

The knowledge base parameters are specified in section 2. If you want to generate a knowledge base through an XML file, please set *Generate_Knowledge_Base* to *true* and give the relative path of the XML file in *Knowledge_Base_XML_File*. If you don't want to generate the knowledge base, just set *Generate_Knowledge_Base* to *false*.

We assume that all original alerts generated by IDS are stored in a single table. So, if there are more than one table involved, please preprocessing them by merging into one table. You can use the *newTable* property to merge it if it can be done by SQL statements. If the *newTable* is set, we will process the SQL statements you have specified here before other processes.

The correlation engine parameters are specified in section 3. These properties are used to specify the alert table name and the column names. *AlertTableName* is a fixed keyword parameter to tell the tool which table stores the original alerts. *AlertID*, *AlertName*, *BeginTime* and *EndTime* are also fixed keyword parameters. They are used to tell the column names of original alert id, alert name, alert begin time and alert end time in the *AlertTableName*. The name and number of the other parameters in this section are based on knowledge base. The keywords are the attribute names used in knowledge base and the values are their column names in *AlertTableName*. In this example, we have attribute name *SrcIP*, *SrcPort*, *DestIP* and *DestPort* in the knowledge base, so we need to specify their corresponding column names in *AlertTableName*. They have the same names here.

Set a file name to *Original_Graph_Output* will generate a DOT formatted graph file based on the original prepare-for relationships which can be displayed by GraphViz.

Sections 4, 5 and 6 contain the parameters used for three utilities: graph reduction, focused analysis and graph decomposition. Please refer to the section "Execution" for detailed usages.

We assume that hyper alert has the same name as its original alert. But to accommodate the situation that hyper alert has the different name with its original alert, or several different alerts could be mapped to one hyper alert, we use section 7 to address it. You can write the name mapping relationship as: *ORIGINAL_ALERT_NAME = HYPER_ALERT_NAME*, such as:

Sadmind = Portmap request sadmind

The property keyword is the original alert name and the property value is the hyper alert name in knowledge base. If nothing is specified in section 7, we assume they have the same name.

Any line started with "#" is a comment line.

3.3 Execution

If you have configured the "Correlator.properties" file and have a knowledge base XML file, you are ready to go. What you need to do is to set property parameters in the property file and turn on or turn off the corresponding switch. The command line would be:

```
java Correlator [-Correlation] [-TransitiveExclusion] [-GraphReduction]
[FocusedAnalysis] [-GraphDecomposition] user [password]
```

After it executes successfully, you can expect some graph files output with the name you specified in the property file. Now, you can use GraphViz to generate the graph with the command line:

```
dot -Tps inputFileNames -o outputFileNames.ps
```

Let's explain the first command line in detail. When you put *-Correlation* in the command line, the correlation engine will be activated and it will search the correlation related parameters in the section 3 of the property file.

When *-TransitiveExclusion* appears in the command line, the tool will delete the transitive edges in the prepare-for relationship.

When *-GraphReduction* is in the command line, the graph reduction utility will be activated and it will search the graph reduction parameters in the section 4 of the property file.

When *-FocusedAnalysis* is in the command line, the focused analysis utility will be activated and it will search the needed parameters in the section 5 of the property file.

Similarly, when *-GraphDecomposition* is in the command line, the graph decomposition utility will be activated and it will search the needed parameters in the section 6 of the property file.

For example, the command line

```
java Correlator -Correlation -TransitiveExclusion -GraphReduction -GraphDecomposition
userName userPassword
```

will activate the correlation engine, the graph reduction utility and the graph decomposition utility, but not the focused analysis utility. And, keeps the transitive edges.

The user and password are used for database authentication. This user needs to have full rights in the database.

3.3.1 Correlation

Set *-Correlation* in the command line; Set the alert related parameters in the property file following the instructions in the section "property file", you can correlate the alerts as you want.

3.3.2 Graph Reduction

Graph reduction is to aggregate the alerts with same type in each graph. To activate it, you must set *-GraphReduction* in the command line. Also, you must give the aggregation time interval. It is in milli second unit. You can set *Aggregation_Time_Interval* to *-1* which will make the aggregation engine to aggregate all the alerts with same type no matter how far in time they are. Give a value to *Graph_Reduction_Output* will generate a DOT formatted graph file based on the aggregated prepare-for relationship, which can be displayed by GraphViz.

3.3.3 Focused Analysis

Focused analysis is implemented on the basis of focusing constraint. To activate it, you must set *-FocusedAnalysis* in the command line. Also, please specify which graph you want to analyze and what constraint you want to use. So you must have some knowledge of the graph and the alerts before you can do the focused analysis.

Focused_Analysis_GraphID is on which graph you want to do the focused analysis;

Focusing_Constraint is an SQL valid presentation that can be inserted into a query like “select alertID from alertTable where *Focusing_Constraint*”;

Focused_Aggregation will activate/inactivate the aggregation engine on the focusd analysis;

Focused_Aggregation_Time_Interval is similar to *Aggregation_Time_Interval* in section 4;

Focused_Output is used for the output graph.

3.3.4 Graph Decomposition

Graph decomposition is to decompose a big graph into several sub graphs based on a user-specified clustering constraint [4]. To activate graph decomposition, you must set *-GraphDecomposition* in the command line.

Decomposition_ID is on which graph you want to do the decomposition;

Clustering_Constraint is an SQL valid presentation that can be inserted into an SQL query directly. It must be in a relationship between *h1.attribute* and *h2.attribute* since we use *h1* and *h2* as two alerts in the query;

Decomposition_Aggregation will activate/inactive the aggregation engine on the decomposed sub graphs;

Decomposition_Aggregation_Time_Interval is similar as *Aggregation_Time_Interval* in section 4;

Decomposition_Output is used to specify the output graph.

4 Sample Execution Procedure

1. java Correlator -Correlation -TransitiveExclusion userName password
2. dot -Tps darpa.dmz1.txt -o outputFileName.ps

To generate the graph using GraphViz. Here, “darpa.dmz1.txt” is the file name we specified for the “Original_Graph_Output” in “Correlator.properties” file. The “outputFileName” is a ps formatted file which will be generated by GraphViz.

5 Trouble Shooting

1. Make sure the *dbDriver* and *dbURL* in the property file is set correctly. Remember the tool uses *DataManager.getConnection(url, user, password)* to

connect to the database.

2. If you use the JDBC-ODBC driver, it may take very long time to execute the SQL statement to delete the transitive edges. If this does happen, either you have patience to wait, or you just execute this SQL statement by hand.

3. If you use Java 2 SDK v1.4 and the JDBC-ODBC driver at the same time, you may have some problem to communicate with the database. You can use Java 2 SDK v1.3 to get around the problem.

References

- [1] <http://www.research.att.com/sw/tools/graphviz/>
- [2] <http://www.iss.net>
- [3] <http://xml.apache.org>
- [4] Peng Ning, Yun Cui, Douglas S. Reeves, "Analyzing Intensive Intrusion Alerts Via Correlation," To appear in *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, Zurich, Switzerland, October 2002.
- [5] Peng Ning, Yun Cui, Douglas S. Reeves, "Constructing Attack Scenarios through Correlation of Intrusion Alerts," To appear in *Proceedings of the 9th ACM Conference on Computer & Communications Security*, Washington D.C., November 2002.