2009

Intrusion Alerts Analysis Using Attack Graphs and Clustering

Hardik Patel
San Jose State University

Intrusion Alerts Analysis Using Attack Graphs and Clustering

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Hardik Patel

Fall 2009

ABSTRACT

Network and information security is very crucial in keeping large information infrastructures safe and secure. Many researchers have been working on different issues to strengthen and measure security of a network. An important problem is to model security in order to apply analysis schemes efficiently to that model. An attack graph is a tool to model security of a network which considers individual vulnerabilities in a global view where individual hosts are interconnected. The analysis of intrusion alert information is very important for security evaluation of the system. Because of the huge number of alerts raised by intrusion detection systems, it becomes difficult for security experts to analyze individual alerts. Researchers have worked to address this problem by clustering individual alerts based on similarity in their features such as source IP address, destination IP address, port numbers and others. In this paper, a different method for clustering intrusion alerts is proposed. Sequences of intrusion alerts are prepared by dividing all alerts according to specified time interval. The alert sequences are considered as temporal attack graphs. The sequences are clustered using graph clustering technique, which considers similarity in sequences as a factor to determine closeness of sequences. The suggested approach combines the concept of attack graphs and clustering on sequences of alerts using graph clustering technique.

ACKNOWLEDGEMENTS

I would like to sincerely thank my advisor Dr. Robert Chun, who has guided me throughout my project. I would like to thank my project committee members, Dr. Chris Pollett and Dr. Mark Stamp, for their direction.

The support and guidance from my advisor and committee members are invaluable. I am grateful for having this opportunity. It has been a challenging journey for me, and I am sure that I will remember the experience for the rest of my life.

Thank you.

Table of Contents

List of Figures

List of Tables

1.0 INTRODUCTION

Today, computers are widely used by most corporations, organizations and government agencies almost everywhere in the world. The tremendous growth in electronic infrastructure has increasingly attracted cyber criminals. The extremely important information which includes but is not limited to military secrets, corporate documents and sensitive personal information such as identification documents, bank accounts is stored on computers and servers. Sometimes in addition to stealing the information, the motive of the attacker is to disrupt the services which are backed by electronic infrastructure. Therefore, the task of keeping sensitive information safe and computing resources secure is of utmost importance for security experts and engineers. The security measures for network and independent computers are vital, but not ultimately sufficient, to protect the information and infrastructure from being attacked by hackers and compromised. The vulnerabilities present in application software and operating systems tend to assist attackers and malicious users.

1.1 Software Vulnerabilities

According to [10], "information security vulnerability is a mistake in software that can be directly used by a hacker to gain access to a system or network". One very common class of vulnerabilities is exposed because of buffer overflow. Buffer overflow occurs when memory write operation in a computer program exceeds the allocated range of the memory. A buffer overflow can be used to execute malicious programs by altering the flow of execution of the original program. Additionally, some programming language constructs are prone to errors and leaks, and using such constructs carelessly opens up opportunities to attackers. According to [10], each year thousands of vulnerabilities are discovered, and keeping all systems up-to-date with patches remains a challenging task. Though security patches may be published shortly after vulnerability is reported, the patches are not immediately applied to the systems for various reasons. Sometimes, the patches which are written in response to emergency may contain other bugs. The patches released for an operating system may require restarting the system. Administrators may not restart the systems where the availability and continuity of business is critical.

A computer system consists of hardware and software, which are built on different architectures and packages and provide various ways of interaction. It is impossible to eliminate all security vulnerabilities of a system considering complexity in building those systems and the interactivity between its components. Therefore, analyzing system globally is important to find possibility and severity of attacks on the system [2]. To determine the security of a system, a security analyst needs to appropriately model the system that takes into account local vulnerabilities and their global effects caused due to interaction [2].

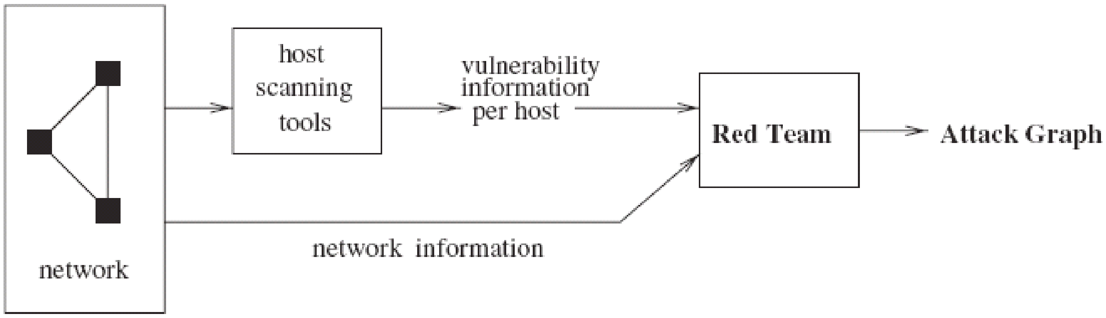


Figure 1. Vulnerability Analysis of a Network
Source: [1]

Figure 1 shows classical process of vulnerability analysis in a network. First, scanning tools are run against different hosts and determine individual vulnerabilities of hosts. Then, local vulnerability information combined with information about connections in a network can be used by a security analyst to prepare an attack graph.

1.2 Attack Graphs

First, let us consider an example. An attacker tries to exploit vulnerability in particular version of ftp and succeeds in overwriting .rhosts file on a victim machine. The attacker can remotely login to the victim machine in the subsequent action. The compromised victim machine can be used by the attacker to maliciously capture another system in the network. Researchers and analysts have given different approaches to model such chain of actions in graphs or trees [3].

According to Sheyner et al. [4], a network attack graph is a state transition diagram in which each state represents a state of the attacker, the defender and the system. Transitions in an attack graph represent actions taken by an attacker which results in a change in the state of the entire system [5]. An action from the attacker may lead to the state which is the attacker's goal. If the system is in a final state, it means that the attacker has succeeded in exploiting the system. An action from a defender or a red team can void the actions of the attacker and can lead to previous safer states. Sheyner et al. [4] formally define an attack graphs as:

"An attack graphs or AG is a tuple $G = (S, T, S_0, S_s)$, where S is a set of states, $T \subseteq S \times S$ is a transition relation, $S_0 \subseteq S$ is a set of initial states, and $S_s \subseteq S$ is a set of success states." S_s denotes states which mean that an intruder has successfully achieved his goal. Attack graphs serve as an important technique for understanding and analyzing vulnerabilities in networks.

Attack graph is a tool that facilitates analysis of system security in a global perspective. A node in an attack graph represents state of the system and an edge represents an action taken by an attacker. Traditionally, attack graphs are prepared by red teams manually and it is a tedious task. Researchers have suggested different approaches to generate attack graphs automatically. Section 2.0 contains more information about different techniques for automated generation of attack graphs. Data mining is one of the techniques to generate attack graphs from intrusion alerts.

1.3 Intrusion Alerts Analysis

As mentioned earlier, attackers attempt to exploit vulnerabilities in operating systems and applications to gain unauthorized access to the system and/or perform denial of service (DOS) attack on the system. As it is unrealistic to build a perfectly secure system, Intrusion Detection System (IDS) has an important role to play in identifying threats and malicious attempts. IDS can be deployed as a network based or a host based system. IDS recognizes intrusion attempts and raises alerts containing information such as source IP, destination IP, port numbers and the service being attacked. Though it is possible and practical to prevent the attacks when they are detected, the analysis of intrusion alerts is equally important to determine the level of damage done to the system for recovery operation. The analysis of intrusion alerts done afterwards is known as post-analysis or post-mortem of intrusion alerts.

1.4 Problem Addressed

Intrusion alert analysis is performed to investigate security incidents and estimate the damage due to intrusion. An IDS may raise thousands of alerts even in a day, which makes manually analyzing those alerts "tedious, time-consuming and error-prone" [14]. One of the useful techniques for alerts analysis is alert correlation. Alert Correlation System (ACS) performs post-analysis on intrusion alerts, presents high-level understanding of the security state of the system and also filters false positives and redundant alerts. Alert clustering method for correlation is helpful to identify the groups of alerts and reduce the number of alerts.

The ordinary alert clustering technique only tries to group the alerts with similar attribute values. For example, the alerts with same source IP address, destination IP address, source port and destination port are likely to be assigned to a single cluster. In addition to such clustering, it is necessary to analyze an intrusion alert in context of other alerts occurring before and after it. In this research, a graph-based clustering technique is suggested for alert correlation. First, the intrusion alerts are pre-processed to form alert sequences and temporal attack graphs. Then, the temporal attack graphs are clustered by graph clustering technique discussed later in this paper. The suggested approach gives a security expert an opportunity to analyze alert sequences instead of individual alerts, hence, alert patterns instead of individual alerts. The goal of this paper is to demonstrate a graph-based clustering technique for clustering intrusion alert sequences.

In the following sections of this paper, the graph-based clustering approach is discussed in detail.

2.0 RELATED WORK

This section briefly describes work done by researchers in the field of network security, attack graphs and analysis of intrusion alerts through correlation.

2.1 Attack Graphs Generation by Model Checking

Sheyner et al. [4] have done pioneer work of modeling global view of network security by constructing attack graphs. Traditionally, attack graphs are prepared manually by red teams. But, this process is highly error-prone, time-consuming and impractical when the network is larger than hundred nodes [4]. Sheyner et al. constructed attack graphs automatically using symbolic model checking algorithms. The steps they followed to construct an attack graph are:

1. Model the network – Sheyner et al. model the network as a finite state machine and state transitions denote atomic actions performed by attackers. Also, desired security property is specified, for example, an intruder must not be able to FTP to machine A.
2. Produce an attack graph – The model from step 1 is used by modified NuSMV [8], a model checker, to automatically generate attack graphs. GraphViz visualization package is used to render the graphs [9].

Figure 2 shows architecture of tool suite used in [4]. A user may specify network model and security properties in XML. A special purpose compiler built by Sheyner et al. converts XML specifications into the input language of NuSMV. They provide an algorithm for attack graph generation. The techniques are highly dependent on underlying data structures and algorithms of model checking in NuSMV.

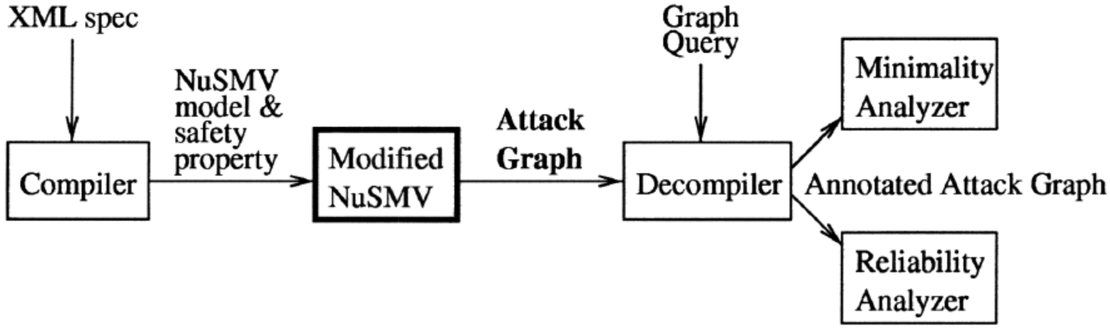


Figure 2. Tool Suite Architecture
Source: [4]

In this approach, the nodes represent the security state of the system as a set of properties, and edges represent actions of an attacker which change the state of the system. This approach is not scalable because the number of states can grow out of control with increase in set of properties.

2.2 Logical Attack Graphs

The concept of logical attack graphs has been proposed by Ou et al. [6]. The model checking system discussed has scalability issues and does not perform efficiently when the size of network is larger than one hundred nodes. Importantly, the size of the logical attack graph is polynomial in size of a network, whereas the size of the attack graphs mentioned in section 2.1 is exponential [6]. The authors of [6] have used MulVAL system, which is a reasoning system to identify security vulnerabilities in enterprise networks automatically. Figure 3 shows Logical Attack Graph Generator designed by Ou et al. The MulVAL reasoning engine is based on XSB, which is a Prolog system developed by Stony Brook. Logical attack graphs have a direct relation between logical dependencies and configuration of individual nodes.

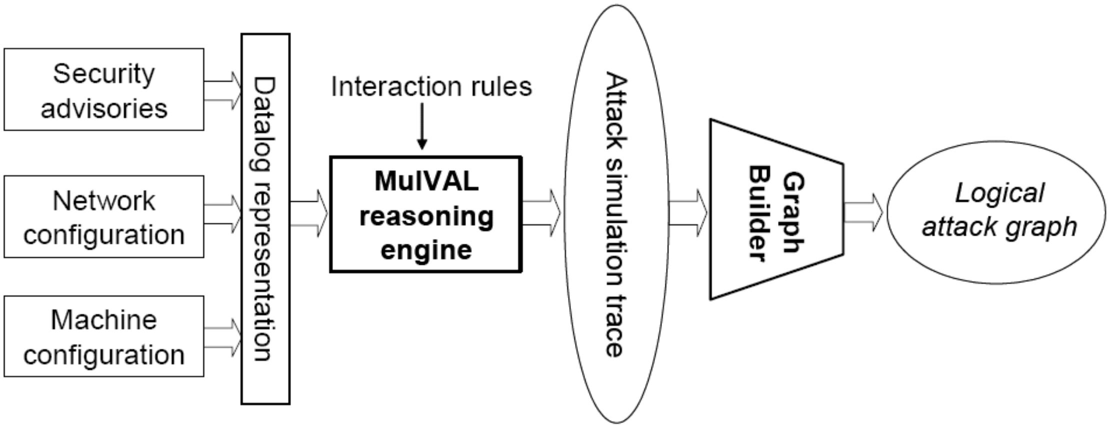


Figure 3. Logical Attack Graph Generator
Source: [6]

The approach of logical attack graphs improves scalability significantly over original attack graphs because using MulVAL has performance advantage over using model checking systems.

2.3 Data Mining Approach for Generating Attack Graphs

Li et al. [5] have done research to suggest a data mining based approach to generate attack graphs. The purpose of their work is to use data mining to construct attack graphs. There is a casual relationship between an attacker and the alert raised by IDS. Therefore, the behaviors of alerts and attacks may be mined from the intrusion alerts database [5]. They perform data mining on intrusion alerts to find association rules from the dataset. The association rules extracted from the alerts database gives information about the alert sequences which are most frequent. An example of association rule is mentioned below:

$[A \rightarrow B] \rightarrow C$

Support = 60, Confidence = 80

This rule indicates that every time A → B are present in a sequence, the sequence is followed by C. The Support value suggests that 60 percent of all the sequences being examined have A, B and C alerts present. The Confidence value suggests that 80 percent of all the sequences that have A → B also contain C.

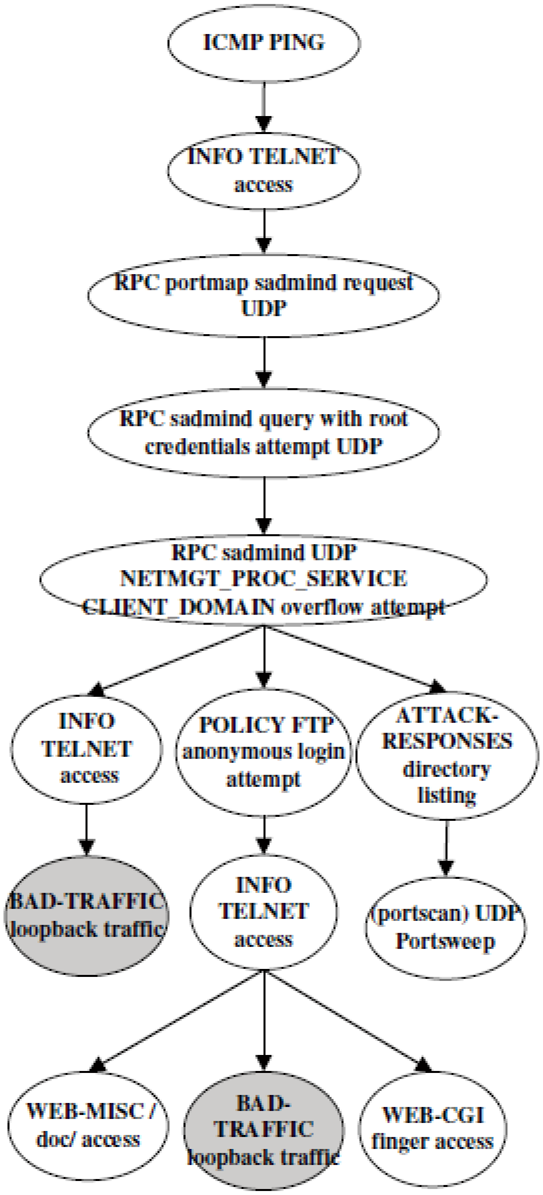

ICMP PING
INFO TELNET access
RPC portmap sadmind request UDP
RPC sadmind query with root credentials attempt UDP
RPC sadmind UDP NETMGT_PROC_SERVICE CLIENT_DOMAIN overflow attempt
INFO TELNET access
POLICY FTP anonymous login attempt
ATTACK-RESPONSES directory listing
BAD-TRAFFIC loopback traffic
INFO TELNET access
(portscan) UDP Portsweep
WEB-MISC / doc/ access
BAD-TRAFFIC loopback traffic
WEB-CGI finger access

Figure 4. Exploit-oriented attack graph, Source: [5]

An algorithm to find frequent alert sequences from the dataset, based on association rule mining, has been proposed in this work. The frequent alert sequences are combined together to form an attack graph.

DARPA 1999 and DARPA 2000 Intrusion Detection Evaluation Dataset have been used for this experiment [18]. This approach is different from the previous ones in a way that it does not require defining system states and security properties in advance. Therefore, there is no need for having prior knowledge of the system and maintaining the knowledge.

2.4 Graph-Based Vulnerability Analysis

In [4], the analysis about attack graphs mentions: For 5 hosts and 8 exploits, and the vulnerabilities associated with those exploits, NuSMV took 2 hours to execute and most of that time was spent on graph manipulation. The attack graph generated as a result had 5948 nodes and 68364 edges. The state space for that example was encoded with 229 bits.

In [3], the authors have suggested an approach in which the problem can be encoded with 229 nodes, one node for each bit in the state representation. They suggest a compact structure for an attack graph claiming that no information is lost in such compact encoding. The algorithm explodes the compact structure into an attack graph for an exhaustive analysis. To simplify the approach, they group together "attacker access privileges, network connectivity and vulnerabilities into generic attributes" [3]. An exploit is modeled as a transformation that, given a set of pre-conditions, produces a set of post-conditions. They define the concept of "monotonicity" which means that once a pre-condition of an exploit is satisfied, it never becomes unsatisfied. Because of their concept of monotonicity, they can identify minimal attack chains without doing exhaustive analysis on an attack graph. The representation here does not have to encounter the exponential explosion problem and therefore, this approach is more realistic and practical for networks with hundreds of hosts according to the authors.

2.5 Intrusion Alerts Analysis

In [15] Siraj et al. have suggested an approach to improve offline analysis of intrusion alerts. Offline analysis of intrusion alerts is performed to investigate the system security breaches and determine the extent of intrusion. It differs from real time intrusion detection because offline analysis is not intended to detect and prevent intrusion as and when it happens. In [15], a hybrid clustering model based on Improved Unit Range (IUR) and Principal Component Analysis (PCA) is proposed. The structural correlation is performed by grouping the alerts with similar attribute values.

Network Intrusion Detection Systems (NIDS) raise multiple security alerts. Correlation of such alerts through clustering is important to improve detection and provide comprehensive analysis because the number of alerts raised is prohibitively large for a security expert to investigate each alert separately. Clustering of intrusion alerts can help filter false positives. The problem with intrusion correlation systems is that they require higher level of participation of security expert in developing and maintaining them as the patterns of attack often change [15]. The results of their clustering are compared with other similar clustering approaches, the clustering approach by the authors provides better results in terms of clustering quality and processing time.

3.0 INTRUSION DETECTION SYSTEMS (IDS)

In this section, Intrusion Detection Systems are discussed in detail. The understanding about the different types of IDS is important to understand the scope of this paper. According to [17], Intrusion Detection is defined as "the process of monitoring the events occurring in a computer system or network and analyzing them for signs of intrusions, defined as attempts to bypass the security mechanisms of a computer or network." Intrusion Detection System can be a combination of software and hardware to perform intrusion detection [17].

Basically, there are two types of IDS, Host Based IDS and Network Based IDS.

1. Host based IDS collects information such as system API calls, resource utilization, logs and the lists of running processes from the local system to detect any malicious activities.
2. Network based IDS, also known as NIDS, monitors and analyzes the traffic of the network.

Classification of IDS is also done based on the underlying methods and techniques used for detection:

1. Anomaly based detection – the normal behavior of the system is predefined and any activity not conforming to normal behavior is considered as suspicious.
2. Signature based detection – The patterns of malicious activities/attacks are developed. Traffic or activity that matches with the patterns is reported as intrusion.

From [17], the IDS can be different depending upon the time aspect.

1. Real-time IDS – The alerts are raised immediately after an attack is detected.
2. Off-line IDS – It serves as an analysis module for the IDS. The alerts are already collected and post-analysis is performed on alerts. It is useful to understand strategy and behavior of an attacker.

The related work discussed in this paper is not directed towards improving Intrusion Detection System. The research is more inclined towards performing off-line analysis of intrusion alert information. The approach suggested in this paper, combines different topics such as attack graphs and clustering on sequences of alerts by graph clustering technique to perform off-line analysis on intrusion alerts.

4.0 DARPA DATASET AND SNORT IDS

In this paper, DARPA 1999 datasets have been used to perform experiments. These datasets have been published by MIT Lincoln Laboratory for research and "evaluation of Intrusion Detection Systems" [18]. These datasets are prepared by capturing TCP traffic of an experimental network when subjected to a systematic attack. The DARPA 1999 dataset contains data for 20 days.

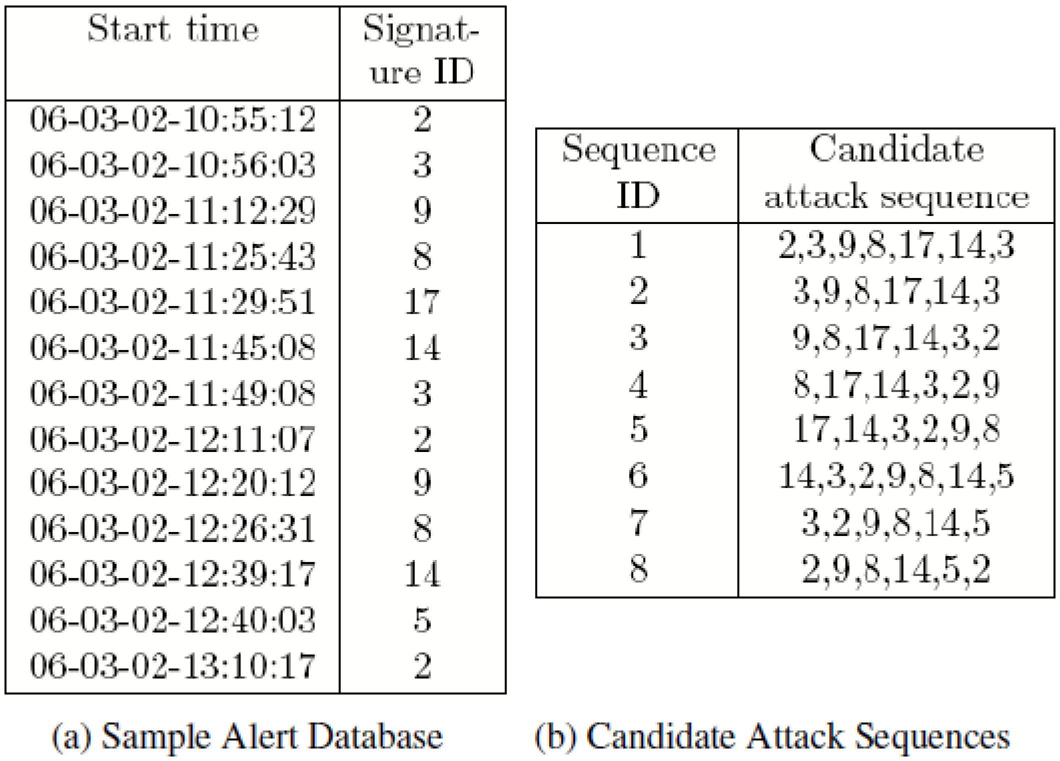
Snort [16] is a popular and powerful open source IDS which is freely available. Snort uses signature based detection techniques and it relies on "rules" to detect the intrusion. The rules of Snort IDS are maintained and regularly updated by Snort team.

The traffic was replayed from the DARPA dataset. The Snort IDS raised alerts from the TCPDUMP files. The command to replay TCPDUMP traffic and record the alerts is as below:

```
$ snort –c snort.conf –r tcpdumpfile
```

With appropriate configuration of Snort IDS, the alerts were directly recorded in MySQL database. The alerts have attributes associated with them which carry important security information such as source IP address, destination IP address, port numbers etc. Timestamp of each alert mentions the exact time when the attack was detected and the alert was raised. The part of a database table containing intrusion alerts is shown in Figure 5(a).

4.1 Alert Sequences



Start time	Signature ID
06-03-02-10:55:12	2
06-03-02-10:56:03	3
06-03-02-11:12:29	9
06-03-02-11:25:43	8
06-03-02-11:29:51	17
06-03-02-11:45:08	14
06-03-02-11:49:08	3
06-03-02-12:11:07	2
06-03-02-12:20:12	9
06-03-02-12:26:31	8
06-03-02-12:39:17	14
06-03-02-12:40:03	5
06-03-02-13:10:17	2

(a) Sample Alert Database

Sequence ID	Candidate attack sequence
1	2,3,9,8,17,14,3
2	3,9,8,17,14,3
3	9,8,17,14,3,2
4	8,17,14,3,2,9
5	17,14,3,2,9,8
6	14,3,2,9,8,14,5
7	3,2,9,8,14,5
8	2,9,8,14,5,2

(b) Candidate Attack Sequences

Figure 5. Alert Sequences from alerts
Source: [5]

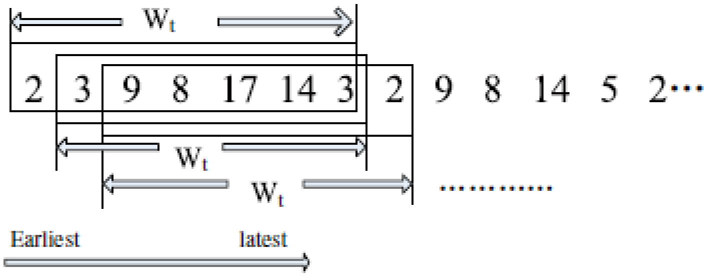


Figure 6. Attack Sequence
Source: [5]

The number of IDS alerts raised by Snort is large. The actual sequence of alerts in the database is considered as a global sequence of alerts [5]. In Figure 5 (a), the intrusion alerts are listed as they appear in the alerts database. Signature ID represents the type of an alert. Start time indicates the time of the attack. The alerts are divided according to time window and alert sequences are formed. Since multistage attacks occur in a certain time frame, the alerts are grouped into sequences according to the time interval chosen as "Time Window" W_t. When W_t is 1 hour, the "Candidate Attack Sequences" will appear as they are in Figure 5(b).

5.0 GRAPHCLUST

Once the alerts are preprocessed and organized into attack sequences, the attack sequences can further be processed to prepare directed graphs. The directed graphs are Temporal Attack Graphs prepared from attack sequences. The graphs are clustered with the “GraphClust” [19]. Shasha et al. have done research to suggest a way to perform clustering on the graphs. “GraphClust” can be used for any application which can benefit from finding relationships among graphs. GraphClust works in three phases [19]:

1. Finds highly connected substructures in each graph.
2. Substructures serve as attributes of each graph.
3. Normal clustering algorithms are applied to cluster graphs.

GraphClust can work for any application which has data to be represented in form of graphs. GraphClust is scalable when working with many large sized graphs [19].

5.1 LNE Format

To be able to use GraphClust on Attack Sequences, it is important to convert the attack sequences to graphs. GraphClust expects input graphs in “LNE” [19] format. LNE (“List of Nodes and Edges ids format”) format is a way to represent a graph in a text file. The specifications of LNE format are as mentioned below:

1. The first line of each graph has a label of the graph and it should start with character ‘#’. The label must be a string containing numbers and alphabets.
2. Second line indicates the number of nodes in the graph.
3. Following lines contain labels of the nodes.
4. Next line indicates number of edges in the graph.

5. Following lines are the node id pairs which mention edges in the graph.

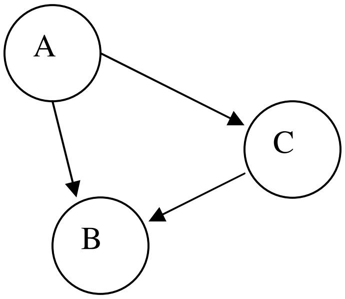


Figure 7. Sample Directed Graph named graph1

The figure above represents a small graph. The LNE representation for the graph is given as:

```
#graph1
3
A
B
C
3
1 2
1 3
3 2
```

LNE representation of graph in Figure 7

Therefore, the attack sequences are converted into directed graphs and directed graphs are written in a single file. The file is an input file for GraphClust.

5.2 Graph Clustering

Once attack sequences are featured as directed graphs, it can be provided to GraphClust for performing clustering on the temporal attack graphs. It is important to study the technique used for graph clustering.

5.2.1 Attribute Selection

Most clustering algorithms deal with a dataset which is comprised of a set of objects or items. The objects in the dataset are assigned to smaller subsets called clusters where all objects of a single cluster are similar to some degree. The similarity of objects is decided based upon their features or attributes. For example, every person has attributes such as height and weight. When a group of people is clustered, the people with similar values for height and weight will be assigned to the same cluster. Therefore, a graph must have a set of attributes to perform clustering on a set of graphs. GraphClust does not have prior knowledge about the attributes of the input graphs. The clustering process for the graphs suggested by GraphClust is explained in detail here.

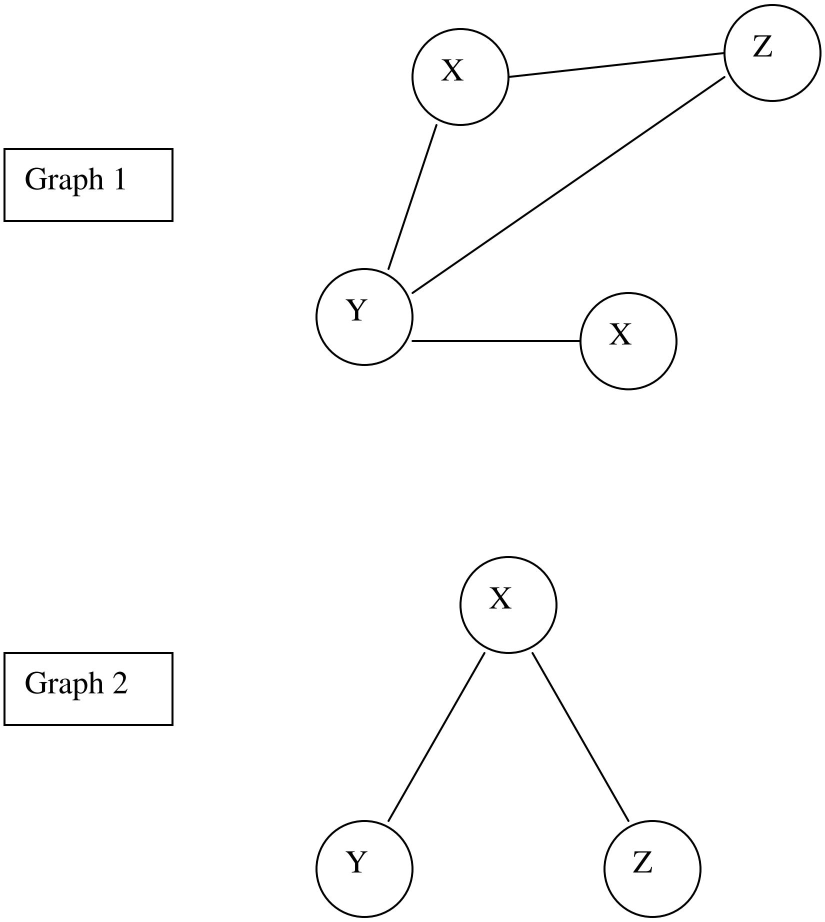


Figure 8. Undirected graphs example

Consider the graphs shown in Figure 8. First, the substructures are found for each graph in the input.

Graph 1: substructures of graph up to length 3 are

From upper node X: {X, XY, XZ, XYX}

From node Y: {Y, YX, YZ, YX}

From node Z: {Z, ZX, ZY, ZYX}

From the other X node: {X, XY, XYZ, XYX}

Similarly for graph 2,

Graph 2: substructures of graph up to length 3 are

From node X: {X, XY, XZ}

From node Y: {Y, YX, YXZ}

From node Z: {Z, ZX, ZXY}

Once all substructures are found, the matrix A is created. The columns of matrix A contains the union of substructures found from all the graphs. Matrix A has one row for each graph in the input. A[i, j] represents the number of times substructure j is present in graph i. In case of undirected graph, XYZ and ZYX represent the same substructure. The matrix A created from the given graphs, graph 1 and graph 2 in Figure 9, will appear as below.

Matrix A:

Graph	Z	ZX	ZY	ZYX	X	XY	XYX	Y	YXZ
Graph1	1	2	2	2	2	4	2	1	0
Graph2	1	2	0	0	1	2	0	1	2

Table 1. Correlation Matrix A for graph1 and graph2

Finding substructures from large sized graphs can be very expensive. Therefore, GraphClust only supports substructure length up to 4. The length limit of 4 is significantly restrictive. Hence, GraphClust provides a way to eliminate this limitation. GraphClust provides an option to use “Subdue” [20] to find “common substructures” from the input graphs. Subdue, developed by University of Texas at Arlington, is a "graph-based knowledge discovery system" that can perform mining on data represented in graphs and find structural patterns [20]. GraphClust can delegate work of finding common substructures in graphs to Subdue. Shasha et Al., the creators of GraphClust, recommend using Subdue to find common patterns in graphs.

Once the common substructures are found and matrix A has been created, the clustering algorithms supported by GraphClust can begin the task of clustering the graphs.

Finding distance between two items is very important in clustering. The distance between two items actually determines similarity among the items in an input set. Two items can be close to each other according to one function can be far from each other based on another distance function. A commonly used distance function is Euclidean distance. The Euclidean distance between two n-dimensional vectors (X_1, X_2, …, X_n) and (Y_1, Y_2, …,Y_n), where n > 1, is calculated as square root of

$$(X_1 – Y_1)^2 + (X_2 – Y_2)^2 + \ldots\ldots + (X_n – Y_n)^2.$$

In GraphClust approach, for each graph, the frequency of each common substructure is recorded as an integer number. Therefore, it is possible to find Euclidean distance between graph 1 and graph 2 based on values in matrix A. Also, GraphClust provides choice of inner product of two vectors as a distance metric. The inner product of two vectors V1 = (X_1, X_2, …,X_n) and V2 = (Y_1, Y_2, …,Y_n) is calculated as

Inner product (V_1, V_2) = $X_1Y_1 + X_2Y_2 + \ldots\ldots + X_nY_n$

For the purpose of clustering graphs, Euclidean distance is better choice for a distance metric since it is calculating less distance between similar graphs.

6.0 GRAPH-BASED INTRUSION ALERT CORRELATION

In this paper, a technique to perform analysis on intrusion alert sequences, which is different from clustering individual alerts, is proposed. Intrusion alert correlation is a powerful way of analyzing intrusion alerts as discussed in section 2.0. Though clustering on alerts provides insight about the security of the system, additional analysis on alerts in necessary. Many times, vulnerability is exploited only after all its preconditions are satisfied. For example, an attacker has to gain a root level access to modify administrator level settings of the system. Therefore, this paper proposes an approach about alert correlation which combines the concepts of attack graphs and graph-based clustering for performing clustering on alert sequences.

6.1 System Architecture

The diagram of system architecture is shown in Figure 9.

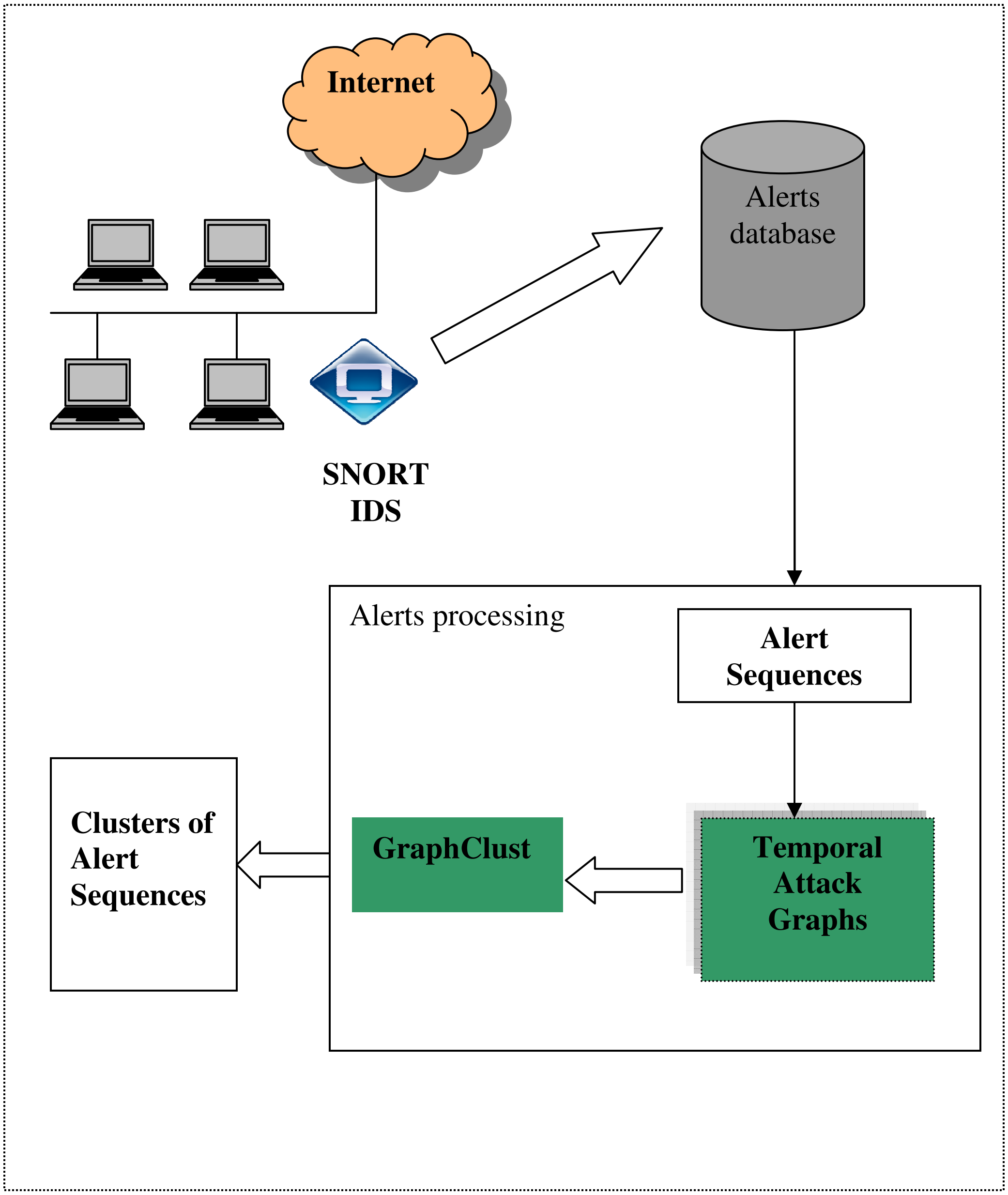


Figure 9. System Architecture Diagram

As shown in the diagram, the processing and analyzing the alerts is done in steps:

1. Collect alerts from the network.
2. Perform preprocessing on the alerts to get attack sequences.
3. Prepare temporal attack graphs from the attack sequences.
4. Perform clustering on temporal attack graphs using GraphClust.

The approach suggested here can perform clustering on alert sequences. Alert sequences contain the alerts which are recorded by IDS in specified time interval. The alert sequences are represented as directed graphs, and the graphs are clustered using GraphClust, a tool for performing graph-based clustering.

6.2 Alert Collection from Network

For the experiments in this paper, Snort IDS has been used. Snort is a popular and widely used Network Intrusion Detection System. Snort can monitor traffic and raise alarms/alerts in real time. To collect the intrusion alerts, the simulation of attack on the network is necessary. Then, the alerts raised by the intrusion detection can be used for further processing and analysis of attacks.

DARPA 1999 dataset, published by MIT Lincoln Laboratory, are used to collect alert information. These dataset contains TCPDUMP files recorded for 3 weeks from a simulated network. TCPDUMP is a popular tool which can capture and display packets from the traffic in the network.

To collect intrusion alert information, the TCPDUMP files from the first week of data were replayed. The Snort IDS was used to monitor the recorded traffic and the Snort

IDS raised alerts which were stored into the MySQL database. The command to raise alerts using snort from the TCPDUMP file is as follows:

```
$ snort –c snort.conf –r tcpdumpfile
```

The alerts raised are stored in the database along with other attributes of the alerts. The most important information about an alert is the timestamp associated with it. The timestamp value gives the exact information about the time when the attack was detected and the alert was raised by Snort. Therefore, after performing this step, the alert information is extracted from the DARPA dataset and the alerts, along with their timestamp and signature ID, are recorded in the database. The timestamp of each alert is useful to sort the alert in the exact order the attacks were detected and alert were raised by IDS.

6.3 Preprocessing the Alerts and Preparing Attack Sequences

Once the alerts are recorded in the database, the next step is to process the alert information and prepare it for the next step of the processing. The alerts have signature id and timestamp information with it. Each signature id is an integer number and indicates a different type of an attack. The whole set of intrusion alerts is sorted in ascending order by their timestamp is called global attack sequence.

The “Attack Sequence Time Window” [5] is the time interval which is moved along the global attack sequence. Attack Sequence Time Window is shown previously in Figure 5. In this step, the global sequence of attacks is broken down into smaller attack sequences. The time interval selected for the Attack Sequence Time Window is 4 minutes based on the experiments in [5]. If the chosen time interval for Attack Sequence Time Window is long, the generated alert sequences will appear as a long series of alerts. The temporal attack graphs generated from such long sequences can be large in size. GraphClust can take a long time to perform clustering on temporal attack graphs when the graphs are huge in size.

Many times, the attacks are carried out in a multistep fashion. An attacker has to have certain privileges to perform certain attacks. When the attack sequences are created, an alert appears between other alerts which were raised shortly before and after it. Therefore, Attack Sequence Time Window can divide global sequence of attacks into sequences which may have alerts with their prerequisite alerts as well.

For the experiments, the Attack Sequence Time Window was chosen as 4 minutes according to [5]. From the whole set of sequences, 1000 sequences were chosen for the experiment.

6.4 Temporal Attack Graphs

From the previous step, the attack sequences are extracted from the global sequences of alerts. The relationship existing between alerts in attack sequences can be used to form directed graphs. In this paper, such directed graphs are referred to as Temporal Attack Graphs. Attack graphs represent the security of the system in a way that a node represents a state of the system and an edge represents an action of an attacker. There are variations of attack graphs, called exploit-oriented attack graphs, where every node in a graph represents an action of an attacker. Temporal attack graphs share similar property and every node is an alert raised by the IDS within the time window.

The technique of creating temporal attack graphs from attack sequences is simple but useful. The sequences of alerts will be processed and the corresponding temporal attack graph will have a node for each kind of an alert. There will be a directed edge between the alerts occurring consecutively.

The temporal attack graphs have to be represented in a form that can easily be processed by the next step. GraphClust is used to perform the clustering on the temporal attack graphs. GraphClust accepts a single text file as an input file and all the input graphs must be represented in LNE format inside that file. LNE format is a mandatory format for GraphClust. LNE format is a specification for representation of the undirected and directed graphs in plain text. The first line of a graph must contain a label of the graph and it has to be preceded by '#' character. The next line indicates the number of nodes in the graph. The next lines indicate a label for every node, one label in a single line. The following line must indicate the number of edges in the graph. The set of lines in the end represent the edges in the graph. The edges are marked as a pair of node labels.

6.5 Clustering Temporal Attack Graphs

The temporal attack graphs found in previous step are formatted in LNE format and the input file is created which contains all the graphs to be clustered. The GraphClust tool is run to perform clustering on the graphs. GraphClust tries to find common substructures from all the graphs provided as inputs. The discovery of common substructures is done better when it is done by Subdue.

The common substructures extracted from the graphs are treated as attributes of a graph. The number of times a substructures is present becomes a value of the corresponding attribute of a graph. So, every graph has a non-negative integer value for each attribute. Therefore, for every graph, a non-negative integer valued vector is available. Since attributes of every graph can be represented as vectors, it is possible to compare two graphs for similarity. Distance metric can be used to find similarity between two graphs. If the distance between two graphs is smaller, then the graphs are close to each other. If the distance between two graphs is a bigger value, then the graphs are far from each other. Distance metric is very important for clustering algorithm. A clustering algorithm should build the clusters in a way that all the graphs in a single cluster are close to one another.

GraphClust provides two types of distance functions, one of them is Euclidean distance and the other is inner product of two vectors. For the purpose of clustering temporal attack graphs, Euclidean distance has been used as a distance metric. Because, when the Euclidean distance between two graphs is smaller, it means that they have a lot more substructures in common and they have similar topology of the graph. For the purpose of clustering on graphs based on their similarity, Euclidean distance turns out to be a better choice than inner product.

The output of GraphClust is an output file which clearly indicates the number of clusters formed based upon the input graphs provided. Each cluster has one of the graphs as its centroid. A centroid of a cluster is basically a center of a particular cluster. Other graphs which are part of a single cluster are listed together. Also, a distance of a graph from the center of a cluster is indicated. GraphClust supports two basic algorithms for performing clustering, K-means neighbor algorithm and Antipole algorithm. While using K-means neighbor, number of clusters has to be explicitly mentioned. Antipole algorithm is faster than K-means [19]. In case of Antipole clustering algorithm, value between 1 and 4 has to be provided as a tightness measure. The radius of the cluster becomes smaller if higher value of tightness is selected and the number of clusters is bigger. Antipole algorithm decides the number of clusters dynamically and not from the user input.

7.0 EXPERIMENTS

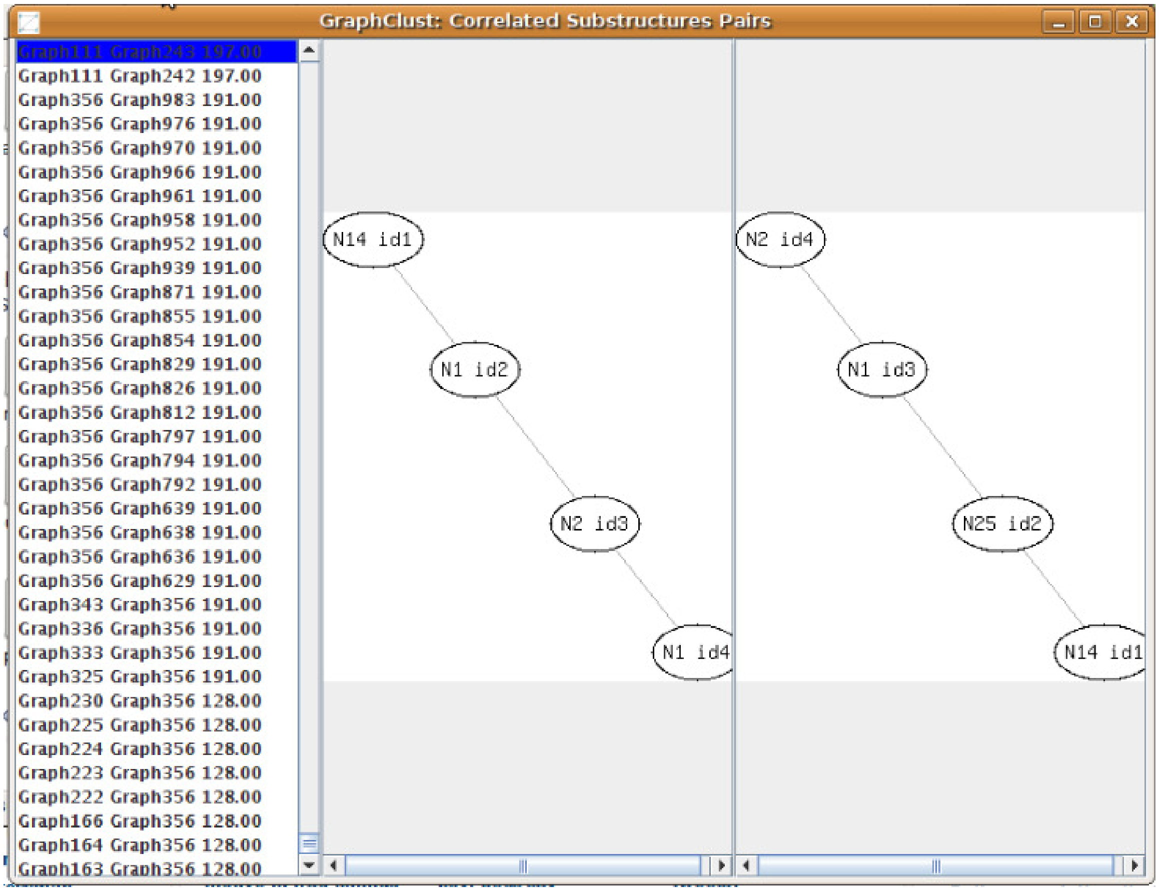
The system architecture and steps for performing graph-based clustering on intrusion alerts is explained earlier. In this section, the details about the experiments on the intrusion alert data is published.

The intrusion alerts were collected from the first week of DARPA 1999 dataset. After replaying TCPDUMP on Snort IDS, the alerts were collected. The intrusion alerts were preprocessed to form smaller alert sequences based on 4 minute window. Out of all the sequences, 1000 alert sequences were selected. These 1000 alert sequences were converted to directed graphs and these directed graphs were denoted in LNE format and written to a single file.

The command used to run GraphClust is as follows:

```
./graphclust_main -tightness 3 -s S -m E –g D LNEgraph_1000.txt
```

Where, LNEgraph_1000.txt is input file which contains 1000 graphs in LNE format, "-s S" option commands GraphClust to use Subdue to find common substructures from 1000 graphs, "-m E" asks GraphClust to use Euclidean distance function as a distance metric for clustering and "–g D" option is used to indicate the graphs in input file are directed graphs.



```
Graph111 Graph243 197.00
Graph111 Graph242 197.00
Graph356 Graph983 191.00
Graph356 Graph976 191.00
Graph356 Graph970 191.00
Graph356 Graph966 191.00
Graph356 Graph961 191.00
Graph356 Graph958 191.00
Graph356 Graph952 191.00
Graph356 Graph939 191.00
Graph356 Graph871 191.00
Graph356 Graph855 191.00
Graph356 Graph854 191.00
Graph356 Graph829 191.00
Graph356 Graph826 191.00
Graph356 Graph812 191.00
Graph356 Graph797 191.00
Graph356 Graph794 191.00
Graph356 Graph792 191.00
Graph356 Graph639 191.00
Graph356 Graph638 191.00
Graph356 Graph636 191.00
Graph356 Graph629 191.00
Graph343 Graph356 191.00
Graph336 Graph356 191.00
Graph333 Graph356 191.00
Graph325 Graph356 191.00
Graph230 Graph356 128.00
Graph225 Graph356 128.00
Graph224 Graph356 128.00
Graph223 Graph356 128.00
Graph222 Graph356 128.00
Graph166 Graph356 128.00
Graph164 Graph356 128.00
Graph163 Graph356 128.00
```

Figure 10. Correlation substructures pairs by GraphClust

Figure 10 above shows a screenshot taken from GraphClust. The pairs of graphs are shown on the left side and their common substructure is shown. GraphClust shows the substructures pairs in correlated graphs. The chart in figure 11 shows the clustering on 1000 temporal attack graphs. The x-axis represents the clusters formed as the result. The height of each cluster represents the number of temporal attack graphs assigned to a particular cluster.

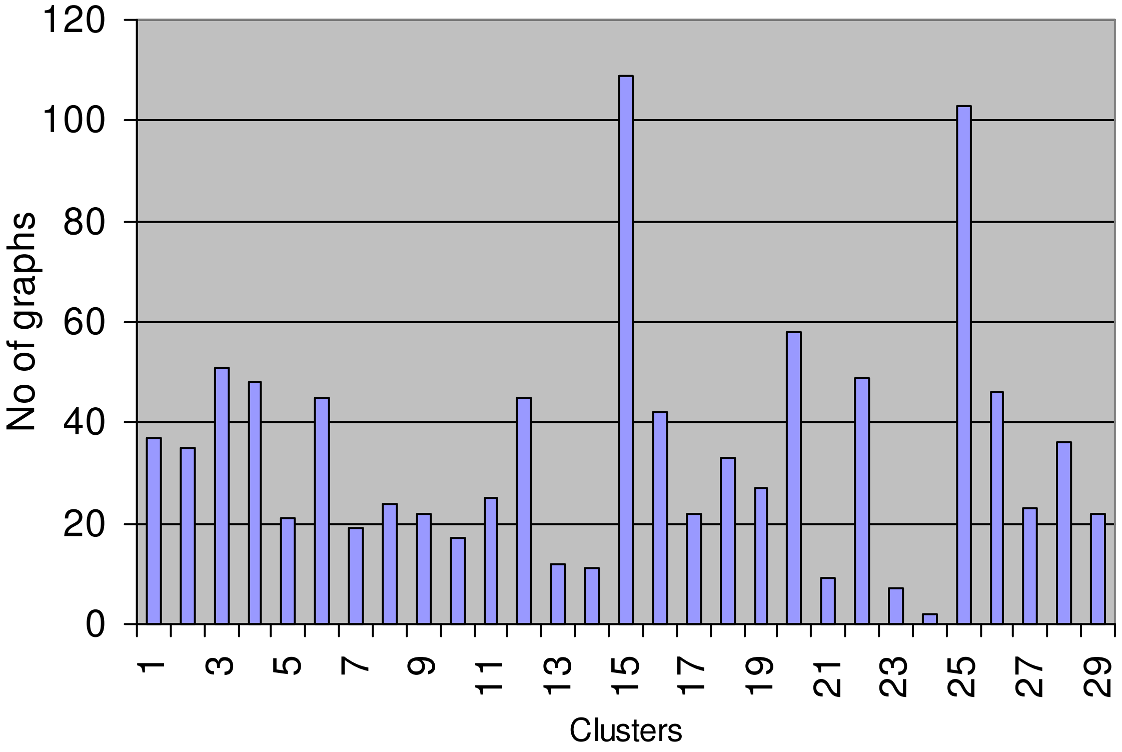


Figure 11. Clustering on 1000 graphs

GraphClust classified 1000 graphs into 29 clusters. To test the results, clustering was performed using different distance metric, which is Inner Product function. When inner product function was used as distance metric, there were total 158 clusters and most of the clusters were of really small size, containing only one item in the cluster. Whereas, in case when Euclidean distance was used as a distance, there were 29 clusters with no cluster having only single element, which is the cluster of size 1. So, Euclidean distance function is a better choice for graph clustering based upon similarity in substructures.

7.1 Test Cases

As discussed earlier, the alert sequences are represented in the form of temporal attack graphs and are clustered with graph-based clustering. For measuring quality of clustering, test cases were designed and results of clustering were observed. When all input graphs are very similar to one another, in other words, if all input graphs are close to one another based on their similarity, all input graphs should be classified in a single cluster in an ideal case. Accordingly, when the input graphs do not have common substructures, in other words, they are far from one another based on their similarity as distance among them, every input graph should be assigned to a different cluster in an ideal case. To test these two cases, two separate input sets were prepared and clustering was performed separately on each input set.

7.1.1 Clustering temporal attack graphs with a high degree of similarity

From the results of the experiment discussed in section 6.0, a cluster of considerably big size was chosen and all temporal attack graphs belonging to the cluster were added to an input file. Since all the temporal attack graphs which were chosen belonged to a same cluster, it was obvious that the temporal attack graphs were similar to one another. The selected graphs were close to one another based on Euclidean distance when common substructures of the graphs are designated as their features. It means that each alert sequence represented by a temporal attack graph contained common attacks or common attack paths. According to the test done for this case on the system, Sixty-five input graphs were provided as input to GraphClust. Fifty-nine of them were assigned to one single cluster. Six out of sixty-five were not clustered in the same cluster. The same results are shown in a chart in Figure 12.

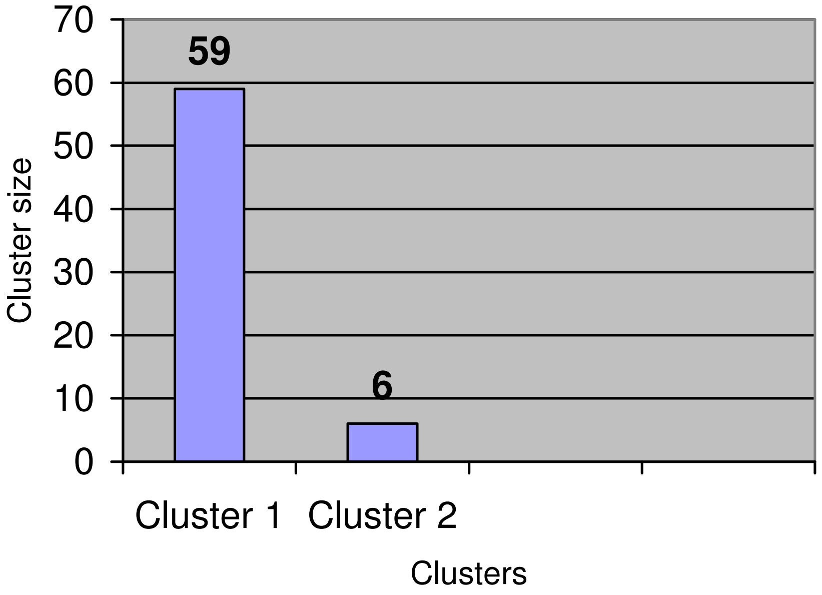


Figure 12. Clustering on 65 similar temporal attack graphs

Fifty-nine graphs were assigned to Cluster 1 and remaining six graphs were assigned to Cluster 2. Ideally, all sixty-five graphs should have been assigned to Cluster 1. But, in the previous experiment discussed in section 6.0, there were total 1000 temporal attack graphs to be clustered. In the test case discussed in this section, there were only 65 temporal attack graphs chosen as an input set. It shows that the distance between two items is considered relative and not absolute while clustering. Two points can be seen as close to each other in reference to 1000 other points, but the same points can be seen as far from each other if there are no points as reference.

7.1.2 Clustering temporal attack graphs with a high degree of dissimilarity

To perform this test, there were 65 temporal attack graphs chosen, and the graphs were different from one another to a great degree. The temporal attack graphs were generated from the alerts raised by SNORT on DARPA dataset as mentioned earlier. The graphs were chosen in such a way that they did not have common substructures. The absence of common substructures meant that the graphs did not have common attack or attack paths. The size of the input for this test was chosen as 65 graphs just to match it with the size of the input chosen for the previous test. The results of the test are shown in Figure 13.

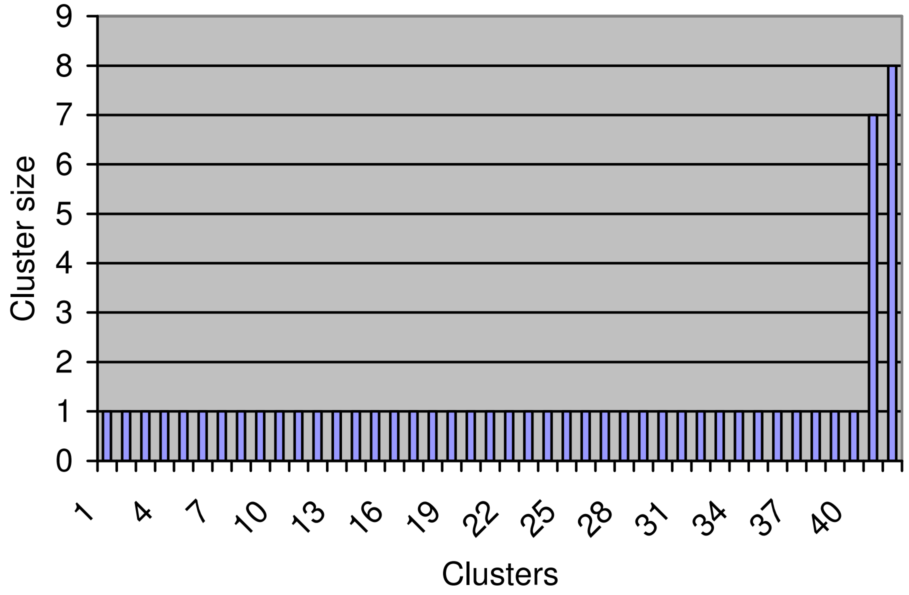


Figure 13. Clustering on 65 dissimilar temporal attack graphs

The x-axis of the graph shows the different clusters that were created from the input set of temporal attack graphs. The y-axis of the graph represents the size of clusters. There were 40 clusters of size 1, 1 cluster of size 7 and 1 cluster of size 8. The 40 clusters of size 1 indicate that each temporal attack graph out of 40 was assigned to a different cluster. This result supported the idea that the graphs should be assigned to separate clusters when they have a high degree of dissimilarity among them.

8.0 RESULTS

In “Intelligent Clustering with PCA and Unsupervised Learning Algorithm in Intrusion Alert Correlation”, Siraj et al. have proposed a clustering technique for clustering and correlation of alerts [15]. According to [15], the correlation of intrusion alerts through clustering can help reduce number of alerts and extract attack steps of an attacker. The system architecture diagram is shown below:

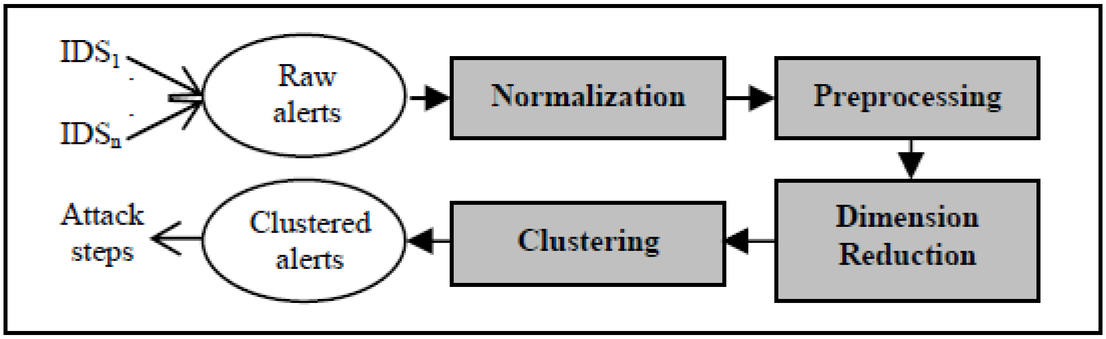


Figure 14. System Architecture Diagram by Siraj et al.
Source: [15]

The research and the approach to correlation of intrusion alerts in [21, 14] is similar to [15].

Before the results are presented, it is important to consider the scope of the work done in this paper. The suggested approach to process and analyze intrusion alerts is meant for post-analysis of intrusion alerts. Similar to the approach suggested by Siraj et al., the work presented in this paper is targeted to processing and analyzing intrusion alerts, and cluster alert sequences by forming temporal attack graphs and performing graph-based clustering. In following subsections, the comparison with the alert correlation technique by other researchers is presented.

8.1 Clustering on Alerts vs. Clustering on Alert Sequences

a). The alert correlation used by other researchers is only analyzing individual alerts instead of looking at an alert in the context of preceding and following alerts. The ordinary alert clustering works as follows. The alert correlation only finds the alerts with similar values for corresponding attributes and based on their similarities assigns them in clusters. Consider the below example as a database of alerts. Alert No. is just a way of representing a sequence number for each alert. Each row represents an alert raised by an IDS and each column is an attribute of an alert. The letters A, B, C, D, P, Q, R and S are only placeholders for real values of corresponding attributes.

Alert No.	**Alert Type**	**Source IP**	**Destination IP**	**Source Port**	**Destination port**
1	-	A	B	C	D
2	-	P	Q	R	S
3	-	A	B	C	D
4	-	A	B	C	D
5	-	P	Q	R	S
6	-	A	B	C	D
7	-	P	Q	R	S

Table 2. Example – sample alerts attribute values

From the above table, there will be two clusters. Since 1, 3, 4 and 6 (grey background) have similar values for all corresponding attributes, they will be assigned to one cluster. Similarly, 2, 5 and 7 will be assigned to another cluster. The alert correlation with such clustering is useful in reducing number of alerts and filtering the alerts.

b). The approach suggested in this paper applies a different technique to cluster intrusion alert information. A table of data is given below for example. The alerts are not clustered directly depending on their similarity. First, the sequences are formed from the alerts database. The sequences have timestamp values associated with it and it indicates the timing of the sequence of alerts. The sequence of alerts is represented in form of directed graphs which are called temporal attack graphs. The clustering is performed on temporal attack graphs created from alert sequences. The basic difference is that the previous approach considers an alert as an independent entity. It is also important to consider the alerts which are reported shortly before and after the alert. In other words, the proposed approach can cluster incidents rather than just alerts.

Sequence ID	**Timestamp**	**Attack Sequence**
101	08:23:11 9/9/2009	**1 2 25 31** 4 4 4
201	09:15:01 9/9/2009	1 1 **1 2 25 31** 0
301	11:20:23 9/9/2009	7 6 7 6 7 6 7 6 7
401	11:45:43 9/9/2009	2 **1 2 25 31** 4

Table 3. Example – sample alerts sequences

From the given table, the attack sequences with ids 101, 201 and 401 will be assigned to a single cluster since they have common substructure “1 2 25 31”. “1 2 25 31” sequence is given just for an example. Each number in sequence means a type of an alert. Therefore, if security expert finds out at the later stage the sequence “1 2 25 31” is malicious; all the sequences in the same cluster must be examined since they are most likely to match the pattern under scrutiny. Also, from the timestamps of each sequence, the timings of all such malicious attacks/probes can be analyzed. This way, a security expert can find out all the instances of the incident which might have happened at different times. Almost all the members of a cluster share common substructures. So, this work, the suggested approach of clustering the alert sequences, can act as additional analysis of intrusion alerts.

8.2 Results of Clustering Alert Sequences

Metric	**Clustering on temporal attack graphs/ alert sequences**			
# of items to be clustered	**400**	**600**	**800**	**1000**
# of attributes	Number of attributes is determined dynamically.			
Running time	**15 min.**	**27 min.**	**49 min.**	**60 min.**
# of clusters	**33**	**26**	**34**	**29**

Table 4. Results of Clustering Alert Sequences

In the above table, the results of graph-based clustering technique, suggested in this paper, are shown.

1. Number of Items to be clustered – This number simply mentions the number of items (temporal attack graphs) which were given as input to the graph clustering program. For graph-based clustering approach, the system has been tested by performing clustering on 400, 600, 800 and 1000 alert sequences to observe the performance trend with the increase in the size of input. First, 400 input alert sequences were selected for clustering. Then, 200 additional alert sequences

were added to prepare the next larger input set each time. Hence, the first 400 alert sequences were present in all input sets.

2. Number of Attributes – Since the ordinary alert clustering method clusters individual alerts, alert attributes are already known in that case. The alerts database used in individual alert clustering technique is in IDMEF format. IDMEF stands for Intrusion Detection Message Exchange Format. IDMEF is XML specification for storing intrusion alert information. Siraj et al. have used fixed number of attributes for clustering. While working with graphs, there is no previous knowledge about the attributes of the temporal attack graphs. Hence, the number of common substructures is determined after the input data is provided to GraphClust. Also, the number of attributes can change when another graph dataset is chosen for clustering. The common substructures from the graphs are discovered by “Subdue”. Therefore, the number of attributes depends upon input data and may vary for different inputs of the same size. Subdue discovered 400 substructures/attributes from the input file containing 400 graphs. In case of 1000 input graphs, it discovered 1000 common substructures.

3. Running Time – Clustering on individual alerts takes less time because every alert has a limited number of attributes and its size is always going to be bounded by the number of alerts and the number of attributes. In case of graphs, size of graph can not be bounded. A graph can have only a single node and it

can also have millions of nodes. Therefore, the time for clustering 1000 or 400 graphs can not easily be estimated or bounded. The time for calculation of substructures is added to the total time required for clustering. There is a huge difference between the numbers of attributes considered for clustering. Graph-based clustering has to deal with a lot bigger number of attributes. The graph-based clustering was performed separately on 400, 600, 800 and 1000 graphs. It took 15, 27, 49 and 60 minutes to cluster 400, 600, 800 and 1000 graphs respectively. From the table, it can be seen that the running time increases with the increase in the size of input.

4. Number of Clusters – In graph-based clustering, there were 33, 26, 34 and 29 clusters created from 400, 600, 800 and 1000 input graphs respectively. Since the first 400 graphs for input sets were kept as a part of all input sets, there is no drastic change in the number of clusters rendered. The number of clusters cannot simply increase or decrease with the size of the input, but it depends on the similarity present among the set of graphs given as input.

9.0 CONCLUSION

In this paper, the concept of graph clustering is studied and applied to intrusion alert data generated by Intrusion Detection System. The clusters of alert sequences which are computed by this approach can provide a ground for better analysis of the intrusion alerts than just correlation of individual alerts through clustering. As mentioned earlier, the proposed approach can cluster alert sequences, hence incidents, rather than just individual intrusion alerts. Since clustering is an unsupervised classification technique, training with the labeled data is not required. Few actions of an attacker can raise multiple alerts. Since the alert is examined along with the context, the strategy of an attacker can be mapped to a group of alerts. The clusters can help identify similar incidents from huge dataset.

The experiments and tests performed on the proposed system, discussed in previous sections, have shown that the alert sequences can be clustered based on their similarity in terms of steps taken for an attack. Clustering on the input set with all input graphs similar to one another assigned almost all the input graphs to a single cluster. In the same way, clustering on the input set with all input graphs dissimilar to one another assigned almost every input graph to a separate cluster. Hence, the proposed approach can perform clustering on alert sequences or temporal attack graphs considering common substructures as attributes. It is concluded that the graph-based clustering can be an important technique for analysis of intrusion alert sequences. The clusters generated can also be used for filtering false alarms and redundant alerts.

10.0 FUTURE WORK

In this paper, a technique to analyze the intrusion alerts by performing graph-based clustering is proposed and it differs from the work of other researchers who have proposed correlation of intrusion alerts by clustering. Though the clustering on the alert sequences can be more helpful than clustering on individual alerts, the system takes longer time to perform clustering. Improving the performance for large size datasets is next important step in the direction of taking this system to a practical solution for analysis on intrusion alerts. Once the system is scalable, the next step will be to test this system on real world data and conduct benchmarking.

REFERENCES

1. Sheyner, O., & Wing, J. (2004). Tools for generating and analyzing attack graphs. Formal methods for components and objects (pp. 344-371) Springer Berlin / Heidelberg.

2. Mehta, V., Bartzis, C., Zhu, H., Clarke, E., & Wing, J. (2006). Ranking attack graphs. Recent advances in intrusion detection (pp. 127-144) Springer Berlin / Heidelberg. doi:10.1007/11856214.

3. Amman, P., Wijesekera, D., & Kaushik, S. (2002). Scalable, graph-based network vulnerability analysis. Paper presented at the 9th ACM Conference on Computer and Communications Security, New York, NY, USA. 217-224.

4. Sheyner, O., Haines, J., Jha, S., Lippmann, R., & Wing, J. (2002). Automated generation and analysis of attack graphs. 2002 IEEE Symposium on Security and Privacy, Washington, DC, USA. 273.

5. Li, Z., Lei, J., Wang, L., & Li, D. (2007). A data mining approach to generating network attack graph for intrusion prediction. Fourth International Conference on Fuzzy Systems and Knowledge Discovery, 307-311.

6. Ou, X., Boyer, W., & McQueen, M. (2006). A scalable approach to attack graph generation. 13th ACM Conference on Computer and Communications Security, 336-345.

7. Feng, C., & Jin-Shu, S. (2008). . 2008 International Symposium on Electronic Commerce and Security 426-431.

8. NuSMV: A new symbolic model checker. Retrieved 11/10, 2008, from http://nusmv.irst.itc.it/

9. Graph vizualization tool graphviz., 2009, from http://www.graphviz.org/

10. Ou, X., Govindavjhala, S., & Appel, A. (2005). MulVAL: A logic-based network security analyzer. 14th Conference on USENIX Security Symposium, 8-8.

11. Common vulnerabilities and exposures. Retrieved 11, 2009, from http://cve.mitre.org/about/faqs.html#a2

12. Arbaugh, W., Fithen, W., & McHugh, J. (2000). Windows of vulnerability: A case study analysis.33, 52-59.

13. http://www.acm.org/crossroads/xrds2-4/intrus.html

14. Smith, R., Japkowicz, N., Dondo, M., & Mason, P. (2008). Using unsupervised learning for network alert correlation. Lecture Notes in Computer Science.

15. Siraj, M., Maarof, M., & Hashim, S. (2009). Intelligent clustering with PCA and unsupervised learning algorithm in intrusion alert correlation. 2009 Fifth International Conference on Information Assurance and Security, 679-682.

16. Snort IDS homepage., 2009, from http://www.snort.org
http://www.snort.org

17. http://www.cs.umn.edu/research/MINDS/talks/tutorial.pdf

18. http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/index.html

19. Reforgiato, D., Gutierrez, R., & Shasha, D. (2008). GraphClust: A method for clustering database of graphs. Journal of Information & Knowledge Management, 07(04), 231-241.

20. http://ailab.uta.edu/subdue/

21. Debar, H., & Wespi, A. (2001). Aggregation and correlation of intrusion-detection alerts. 4th International Symposium on Recent Advances in Intrusion Detection, 85-103.

22. Zhong S. & Khoshgoftaar T. M. (2007). Clustering-based network intrusion detection. International Journal of Reliability, Quality and Safety Engineering, vol 14. 169-187.