2009

Intrusion detection and response for system and network attacks

Fred Philip Stanley
Iowa State University

Intrusion detection and response for system and network attacks

by

Fred Philip Stanley

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Johnny Wong, Major Professor
Samik Basu
Shashi Gadia

Iowa State University

Ames, Iowa

2009

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

This work focuses on Intrusion Detection System (IDS) and Intrusion Response System (IRS) model for system and network attacks. For decades, IDS has evolved tremendously and has become highly sophisticated. However, the response to an attack is still manually triggered by an administrator who relies on static mapping to counteract the intrusion. The speed of attack-spread and its increased complexities in recent years have shown that it is highly critical to develop an automatic IRS. Moreover, manual responses are not flexible and effective in distributed environment without infrastructure.

This work presents a cost based response model that is tightly coupled with multi-source IDS. It is a known fact that any system can be broken down into smaller granules of services and resources. A dependency graph is employed to describe the relations between services and resources in a system. This dependency graph is also used to propagate the total value of the system down to the service and resource levels. The damage cost of the intrusion and the response cost of the responses are evaluated using the dependency graph. Using several performance metrics, a response which brings the most benefit to the system is deployed.

We demonstrate the abilities of our model by using buffer overflow attack caused by a computer worm on Optimized Link State Routing (OLSR) protocol on a wireless ad-hoc network environment. Experimental results show that our model is effective and is highly practical.

CHAPTER 1. OVERVIEW

1.1. Introduction

Intrusion detection has been the focus of intense research for the last few decades and has become highly sophisticated. Typically *intrusion detection* is defined as the process of analyzing information about the system activity or its data to detect any malicious behavior or unauthorized activity. There are many types of intrusion detection systems which include Network based Intrusion Detection Systems (NIDs) and Host based Intrusion Detection Systems (HIDs).

Network layer attack can be defined as a packet or a series of packets that abuses the field of network layer header in order to exploit vulnerability in the network. NIDs look at the network traffic data and detect any malicious activity between any two interacting systems. Snort [17] is an example of a network based intrusion detection system. On the other side, HIDs monitor and analyse the internals of a computing system like memory, file system, user rather than the external interfaces to find abnormal behavior. OSSEC [23] is an example of a open source host based intrusion detection system and there are lots of open source HIDs available on market.

Typically, any type of intrusion detection system will trigger an alert or will log the malicious activity. After an intrusion detection system has detected a malicious activity it is desirable to take evasive or corrective actions to stop the attack and ensure safety of the computing environment. Such a counter-measure is called *intrusion response*.

Traditionally, when an intrusion detection system is triggering an alert it is the duty of the system administrator to go through every detail of the alert generated and then deploy a suitable response. Unfortunately, system administrators neither can keep up with the pace of the intrusion detection system nor can they react upon these alerts within a reasonable time limit. Moreover these manual responses are not flexible and are not very efficient. These manual responses totally depend on the expertise of the system administrator.

Automated response systems can take over the task which is required in case of many distributed systems to respond to an alert more quickly and accurately. Even though the

intrusion response system component is often integrated with intrusion detection, it receives considerably less attention than the intrusion detection research because of the complexity involved developing and deploying response in automated fashion.

An automated intrusion response is a mechanism to select a response without human intervention. The main advantage of automated response is reduction in the response latency from the time of detection. An automated response also provides consistent and accurate response across systems and organizations. These automated responses also eliminate the fact that many system administrators fail to consider the cost associated with the response deployed. Even though the automated responses have lot of advantages, it is slow to be adopted because its implementation is complex even for experienced professionals as it still needs standardized performance metrics and increased automation .

1.2. Motivation

A major concern of most of the existing automated intrusion response systems is the way to build a system model. This system model takes into account the resources of the system and factors like intrusion cost, response cost and response effectiveness. Many graph based system models have been proposed in the past. Some of the examples include a dependency graph [1], graph models [3] and hierarchical tree model [2].

Most of these approaches need the system administrator to assign values to specific resources of the system and then build the system model. This assigned value for the resources are used in the process of evaluating metrics like intrusion damage cost, response cost and response effectiveness. Any system can be divided into many components that may be a service or a resource. But it becomes intractable in general for the system administrator to assign values for a specific resource or a service in a system. It is not only a tough job for the system administrators but it may not also be an accurate estimate for that system.

On the other hand, the project managers or the business analysts are more comfortable in assigning the values just for the services of the system. This can be easily obtained from the cost incurred by the organization when that service stops functioning. The other problem is that most of the existing automated response systems assume that the damage cost of an intrusion is known and they only provide ways to estimate the response cost.

To solve all the aforesaid problems we have proposed and implemented a generic response model that will be a possible solution to these problems. We provide an accurate and a consistent way to evaluate the system and a procedural method to select a response. We have introduced a dependency graph that will help in evaluating values for all elements in the system, estimating the damage cost, response cost and response effectiveness of an intrusion.

1.3. Contribution

In this work we design and implement an intrusion detection and response framework. We introduce a dependency graph model which represents the interdependency between entities of a system. The system is divided into layers of different entities. An entity is a general name used to represent a resource or a service of a system. The dependency graph divides any system into four layers namely the application services, component services, system/support services and the resources of the system. The resources are further categorized into virtual resources and physical resources.

This dependency graph represents each of these entities as nodes connected by edges. These edges represent the dependencies between entities in terms of the security goals, namely confidentiality, integrity and availability. Each edge also has information that describes the dependency weight among services and resources. As a first step the overall value of the system is gauged from the business analyst or the manager. Then the total value of the system is shared among the top level application services. Then a value propagation method is used to propagate the value of the top level application services to all other entities of the system. This propagation method uses information provided by the dependency graph to do the value propagation.

Each entity in the system gets a value for its own after the propagation. Thus we do an accurate value estimate of all the entities in a system. Then we use the dependency graph constructed to find the Damaged Cost (DC) of an intrusion when an attack takes place. Along with that we provide functions to evaluate metrics like response cost and response effectiveness.

Finally we provide a method to select the best response for an intrusion and then deploy the selected response. We deploy a response only when that response is estimated to do more

benefit than damage to the system. We use Linux shell scripts to deploy responses such as adding a firewall rule, restart a process or suspend a process.

The main contributions of this project are the following:

Intrusion detection/response framework: A framework is implemented that works on the principle of proposed generic response model. The framework is adaptable to different environments. It has a multi-source intrusion detection module and a cost based automated intrusion response engine. The communication between the intrusion detection module and the intrusion response engine is based on the client server architecture.

Dependency graph: A dependency graph is designed and implemented using an XML technology. This dependency graph helps categorizing a system into different layers of entities. Then this graph is used to represent the interdependencies between entities.

Value propagation method: A value propagation method is implemented to propagate the whole value of the system to every entity of the system.

Response selection and deployment module: This provides a generic approach to measure metrics like damage cost and response cost. This module also helps to select and deploy the best response.

1.4. Road map

This thesis is organized as follows. Chapter 2 details some of the related work and highlights some interesting open source tools in the field of intrusion detection and response. Chapter 3 describes the intrusion detection module, focusing on Optimized Link State Routing (OLSR) protocol and an OLSR worm attack. In Chapter 4 introduces the cost based automated intrusion response engine that uses dependency graph to select the best response. Chapter 5 describes the implementation details of the intrusion detection and response framework. Chapter 6 presents the results, timing and efficiency of the framework. Finally Chapter 7 concludes the discussion with opportunity for future development and enhancement.

CHAPTER 2. RELATED WORK

In this chapter we summarize the work previously done in the area of automated response system. The latter part of this chapter focuses on some open source intrusion detection system and response tool available to mitigate system and network attacks.

2.1. Background

Taxonomy of intrusion response system with a review of current trends is provided by Stakhanova et al. [4], which provides a general overview of the existing work on intrusion response systems. The intrusion response systems can be classified as active and passive. Passive response systems do not attempt to minimize the damage already caused by an intrusion or prevent further attacks. The main purposes of those passive systems are to notify the authorities or to provide attack information. As opposed to passive intrusion response systems the active systems aim at minimizing the damage caused by the attack and attempts to locate the attacker.

The classification according to the level of automation has been presented in early days by several authors [1][5][6]. But [4] has come up with a new classification based on the degree of automation of the intrusion response system and can be divided into notification systems, manual systems and automatic systems. Notification systems provide information about the intrusion which can be used by the system administrator to respond to the intrusion. Most existing IDS provide notification response mechanism. Manual systems provide a higher degree of automation than the notification systems. They allow the system administrators to launch an action from a predefined set of responses based on reported attack information. Automatic systems allow immediate response to the intrusion through decision making process. Current research on intrusion response systems focus on automatic response systems.

Lee et al. [7] introduce a cost-sensitive model for intrusion detection and response. They examine the major cost factors associated with the IDS which includes development cost, operational cost, damage cost due to successful intrusions and the cost of manual and automated responses to intrusions. Operational cost covers the cost of processing and analyzing the data for detecting the intrusion. Damage cost is the measure of the cost incurred

because of an intrusion. The response cost includes operational cost incurred while deploying the response actions. They define a cost model to formulate the total expected cost of an IDS and they present cost-sensitive machine learning techniques that can produce detection models that are optimized for user-defined cost metrics. These three cost factors mentioned act as metrics for selecting a response.

Toth et al. [1] provide a network-based response model and an algorithm to evaluate the impact of a response action on entities of a network. Networks are complex structures that include many elements which are heavily related and dependent on each other. The elements of a network system include services offered by a host, system users, security control information, network topology and firewall rule. They represent the dependencies between these elements using a dependency graph. There can be different types of dependencies between these elements. So when an intrusion takes place, the penalty costs of different response actions to the system are evaluated. The penalty cost refers to the impact of a response action to the elements of the system. The response with the lowest negative effect is selected. This network based response model helps to select a globally optimal response.

Balepin et al. [2] provide a structured approach in automatic responses by coupling the automated response with the specification based and host based intrusion detection. They describe a system map based action cost model that gives the basis for deciding on a response strategy. The resources of a system are arranged in two ways as resource type hierarchy and system map. The resource type hierarchy is the process of grouping the resources by type. This is because every group has most likely the same response actions associated with an attack.

The system map is a direct graph to model the local resources and their dependencies. The nodes in the graph represent every instance of the resource types and the edges represent the dependencies between them. The authors propose the use of cost based on priority of a resource as a base metric for selecting a response. There are nodes on the system map called priority nodes that represent the most important resources on the system. They carry a non-zero value. There are nodes that represent the basic resources of the system and they have no value of their own. They get the cost based on its dependency with the priority nodes of the system. Every node has a list of basic response action and every response action has an

activation condition. They use a cost model to select a response, when several response actions with activation criteria match with the current situation. Every node is associated with some numeric cost by the system administrator and the system map is generated.

They provide methods to compute the cost of intrusion action, benefit of the response and the cost of a response action. Cost of an intrusion action is defined as the sum of nodes that were previously in safe before they got affected. The benefit of the response is defined as the sum of cost of all nodes previously in set of affected nodes that this response action restores to working state. Cost of response action is the cost of the nodes that get negatively affected by the response action. Finally they provide a response selection that maximizes the benefit at the minimum cost. They also show the process of suspending the attack and then designing an optimal strategy during the time of uncertainty.

Foo et al. [8] introduce a complex framework called Adaptive Intrusion Response using Attack Graphs (ADEPTS) for determining automated responses when an intrusion takes place. They use attack graphs to identify the response actions required to stop possible attacks on targets in a distributed system. The other objective is to allow certain parts of the system to continue to provide partial system functionality in the case of failures. ADEPTS uses a directed graph representation to model the spread of the failure though the system. It also presents algorithms for determining appropriate responses and monitoring their effectiveness and quantifies the effect of disruption through a high level survivability metrics.

This framework uses graph called intrusion Graph (I-graph). The nodes of an I-graph represent sub-goals for the intrusion and edges represent pre-condition/post conditions between the goals. They also have a graph representing the interdependencies between services. Using the information of the interdependencies among services, an I-graph is generated. Compromised Confidence Index (CGI) of a node in the I-graph is a measure of the likelihood that a node has achieved by an attacker. Using this metric the response deployment location is determined. Responses are selected to frustrate attack goals based on effectiveness of that particular attack in the past, disruptiveness to legitimate users and the confidence level which indicates that attack is actually taking place.

Jahnke at al. [3] propose a graph based approach for modeling the effects of both attacks against computer networks and response measures as a reaction against the attacks. Certain properties of the model graph are utilized to quantify different response metrics that are well known to network security officers. They generate a dependency graph for the system where the nodes represent different system resources and the edges represent their dependencies. This graph is used for choosing the best response from the list of available responses after measuring the effects of all the responses. Although this method requires careful graph construction and validation, it allows automatic assessment of the response success calculated through the change of the availability of resource nodes in the graph.

2.2. Open source software

There are many intrusion detection and response tools available in commercial market and in research community. This section highlights a few interesting and popular ones.

2.2.1. Snort

Snort is a network based IDS that scans the traffic and tries to find suspicious activities using a set of rules. A rule set is a collection of specific byte pattern that indicates a particular attack. This type of IDS is usually called signature based intrusion detection system. The signature of the attacks can be downloaded from snort web site and once it is configured it can be used by snort to detect that attack. Snort can also be configured to work as a packet sniffer and packet logger. Snort performs protocol analysis, content searching/matching, and is commonly used to actively block or passively detect a variety of attacks and probes such as buffer overflows, stealth port scans, web application attacks, SMB probes, and OS fingerprinting attempts. The main problems faced with snort are difficulty in installing and maintaining, needs significant amount of tuning required to avoid false positives.

2.2.2. Fwsnort

In recent years much effort has been put into integrating the snort with the iptables. We will be discussing about several tools that are developed based on this approach.

Fwsnort stands for firewall snort. It translates snort rules into equivalent iptables rules and generates a shell script that implements the resulting iptables commands. First it parses the

iptable rule set on the machine and determines which snort rules are applicable to specific iptable policy. Then fwsnort generates iptable rules which log information to the syslog. Fwsnort utilizes the filtering and inspection capabilities of iptables including heavy use of iptables string matching extension in order to match snort rules as closely as possible within an iptable rule set. These logged messages can be analyzed with a log analyzer such as psad. Fwsnort is able to translate approximately 60% of all the rules from the Snort-2.3.3 IDS into equivalent iptables rules.

2.2.3. Psad

Port Scan Attack Detector (PSAD) is a light-weight IDS which results from Bastille-NIDS project. Psad makes use of iptables log messages to detect, alert, and (optionally) block port scans and other suspect traffic. Firewalls like iptables offer extensive logging and filtering capabilities and can provide valuable security data that cannot be ignored. A dedicated IDS such as snort offers a large feature set and comprehensive rules of language to describe network attacks. Psad utilizes both the advantages by adapting the snort rules and matching it against the logs created by the firewalls like iptables. In fact psad can be considered as a log file analyzer.

2.2.3.1. Various attacks detected by psad

Psad can detect various types of suspicious network traffic by examining network and transport layer headers. It can detect port scans generated by Nmap, probes for various back door programs, Distributed Denial of Service (DDoS) attacks and other efforts to abuse networking protocols. Psad can detect many types of port scans like TCP connect scan, TCP FIN, XMAS and NULL scans, TCP SYN or half-open scan and UDP scan to name a few. Psad can also passively fingerprint the remote operating system from which a scan or a malicious traffic originates. This information can be reported to DShield, which is a community-based collaborative firewall log correlation system. It serves as a central depot of data. These data are provided by software from both open source and commercial worlds. These collective data is used to analyze the attack and attack trends.

Psad also uses snort rules to do a signature matching. Psad uses all the 150 rules in the snort ruleset to do signature based intrusion detection. For example naptha DDoS attack is designed to flood the target TCP stack with so many SYN packets that the system cannot serve the legitimate requests. But there is a snort rule with id number 275 that can detect a packet generated by the naptha DDoS attacker who uses IP id value as 413 and TCP sequence number 6060842. Thus using the snort rules, psad is able to detect most of the known DDoS. As a matter of fact, psad can detect and generate alerts for over 60 percent of all Snort-2.3.3 rules.

2.2.3.2. Active response with psad

One of the features sought after intrusion detection is the ability to automatically respond to an attack. Psad has the ability to respond to an attack dynamically by instantiating blocking rules against the attacker. The main method psad employs to respond is by changing the local filtering policy so that it blocks all accesses from an attacker's source IP address for a configurable amount of time. It can reset the established session or a connection with the attacker for a specific amount of time. Some of the responses include instantiation of firewall blocking rules, modification of routing tables, generation of ICMP port/host unreachable packets for UDP attacks and use of TCP resets for attacks that take place over TCP connection.

2.2.4. Combining psad and fwsnort

A combination of psad and fwsnort provide a system with intrusion detection and response capabilities. Although psad provides detection, alerting and auto-response capabilities, the effectiveness of its detection engine is fundamentally limited by the characteristics of the iptables logging format. Better attack detection is offered by fwsnort, including detection for application layer attacks using string matching capabilities.

For example WEB-PHP setup.php access attack can be detected and stopped, which is an attempt to exploit an input validation weakness in the MediaWiki software. A successful exploit of the vulnerability could lead to unauthorized remote execution of code on the targeted system upon receipt of specially constructed URI parameters within a HTTP request.

This HTTP request looks like the following *http://x.x.x.x/setup.php*. Fwsnort uses the string matching extension to detect */setup.php* string within an established TCP connection. So an alert is logged when fwsnort detects string */setup.php* over a web session. Thus fwsnort is able to inspect application layer data and log malicious activities. Now psad can analyze the log and apply its alerting and reporting machinery to the event. Then psad maps this alert to appropriate snort rule so that more information (e.g. the class type of the attack) can be found. In this case it is a *web-application-activity* class type. Such valuable information is given to the system administrator who is trying to investigate the nature of the attack and to determine what a successful exploit might have meant for the security stance of the network.

The combination of these two tools can also help us in providing a cross layer intrusion response. For example when there is an attack on an application layer, like a user trying to abuse an application, the usual response would be disabling the user account. But a network level response can also be provided. The string matching libraries of fwsnort are used to inspect the application layer data and detect the application level attack. Then psad can read the logs of fwsnort and respond to the attack either by adding an iptable rule to block a connection or perform a network level response like tearing down a TCP connection of the attacker by sending a RST packet.

2.2.5. Fwknop

The biggest implication for all signature based IDS is that they detect only known attacks. The challange that arises is that how to protect network services from undiscovered vulnerabilities. These are called *zero-day attack.* These are created when someone finds an undiscovered vulnerability in a piece of software and writes an exploit for it. He becomes the first person in the world to find this vulnerability and there is no signature based IDS for it. To address this problem this tool helps in adding an additional layer of security to arbitrary network services.

2.2.5.1. Port knocking

Port knocking is the communication of authentication data across closed ports which a service (such as SSH daemon) is protected behind a packet filter configured in a default-drop

stance. Any would-be client that wishes to make a connection to a protected service through the default-drop packet filter must first prove possession of a valid port-knock sequence. If the client produces a correct knock sequence then the packet filter is temporarily reconfigured to allow IP address that sent the sequence to connect to a protected service for a certain period of time.

2.2.5.2. Fwknop tool

FireWall KNock OPerator (fwknop) is released as an open source project using GNU Public License (GPL). It combines the encrypted port knocking with passive OS fingerprinting, making it possible to allow only Linux systems to connect to your SSH daemon. Fwknop's port knocking component is based on iptables log messages and it uses iptables as a default-drop packet filter. It implements an authorization scheme known as Single Packet Authorization (SPA) for Linux systems running iptables. This mechanism requires only a single encrypted and non-replayed packet to communicate various pieces of information including desired access through an iptables policy. An authorization server fwknopd passively monitors authorization packets via libpcap and hence there is no 'server' to which to connect in the traditional sense.

Fwknopd is the server component for the FireWall Knock Operator, and is responsible for monitoring Single Packet Authorization (SPA) packets that are generated by fwknop clients, modifying an iptables or ipfw policy to allow the desired access after decrypting a valid SPA packet, and removing access after a configurable time-out. The fwknop client has a rich set of command-line options that allow system administrators to inform the fwknop server to grant the exact access according to the iptables policy. The main application of this program is to protect services such as SSH with an additional layer of security in order to make the exploitation of vulnerabilities much more difficult. Any service protected by fwknop is inaccessible (by using iptables or ipfw to intercept packets within the kernel) before authenticating and anyone scanning for the service will not be able to detect the fact that it is listening.

Thus this is a powerful tool to protect server by default-drop packer filter, through which access is granted only to clients who are able to prove their identities to use its service.

Fwknop is mostly used to provide an additional layer of security for services that typically have long running sessions such as *OpenSSH* or *OpenVPN*.

2.3. Summary

All these models have one common problem, i.e., lack of consistency in the value evaluation of the resources and lack of procedural way to select a response. Our approach builds on existing work and have provided a consistent way to evaluate the value of all resources across all system with a procedural response selection model.

CHAPTER 3. INTRUSION DETECTION & RESPONSE FRAMEWORK

This chapter describes the overall design of the intrusion detection and response framework that is developed in this project. We present the multi-source intrusion detection model and the communication method between the intrusion detection module and the intrusion response engine.

This chapter also highlights Optimized Link State Routing (OLSR) protocol [22], an IP routing protocol on a wireless ad-hoc network environment and the different packet format used by OLSR. We also illustrate working of an Internet worm that uses a specific vulnerability in OLSR routing protocol and propagates itself across the wireless ad-hoc network. We have used this worm to evaluate the intrusion detection and response framework. The other details on the implementation of the worm are given on chapter 5.

3.1. Overview

The framework developed has two main parts, namely, the multi-source intrusion detection module and the intrusion response engine. Fig. 1 shows the different modules within the intrusion detection and response framework. The intrusion detection module consists of multiple independent intrusion detection sources. These intrusion detection sources are responsible for the detection of intrusion or any malicious activity on the network or on the host and send alerts to the intrusion response engine which is responsible for deploying response so as to curtail the effects of intrusion.

The communication between different intrusion detection sources and the intrusion response engine is based on client server architecture. These intrusion detection sources can be anything from a signature based IDS to any custom designed intrusion detection module developed by a system administrator to monitor any specific aspect of the system.

The intrusion response engine on receiving an alert from these intrusion detection sources would process the damage effect of the intrusion and then would come up with a list of possible responses and deploy the best response that will mitigate the effects of the intrusion.

The response engine does a cost sensitive response selection. More detailed description of the intrusion response engine will be shown on chapter 4.

.

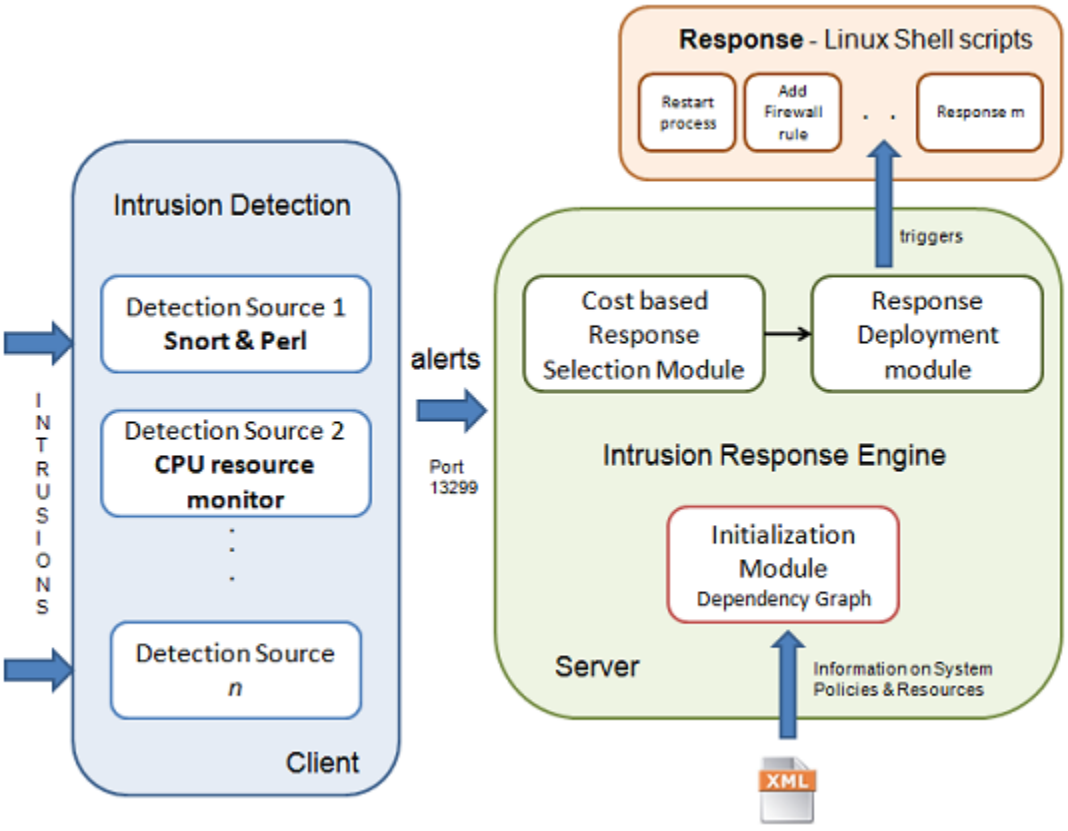

Figure 1. Multi-source intrusion detection & response framework

3.2. Multi-source intrusion detection module

The intrusion detection module is a collection of many individual intrusion detection sources. Intrusion detection basically refers to a variety of techniques for detecting attacks in the form of malicious and unauthorized activity. There are three classifications of intrusion detection approaches [16]. They are: Misuse-based, Anomaly-based and Specification based. Misuse-based technique relies on pre-specified attack signature, and execution techniques

matching these signatures are flagged as abnormal. In case of Anomaly based approach, we use machine learning algorithm to discover normal patterns and any deviation from the normal patterns will be detected as malicious. The last approach is called Specification based technique, which operates in similar fashion as Anomaly based method. It detects deviations from specified legitimate system behavior. However it requires user guidance in developing a model of valid program behavior in the form of specification.

For the implementation we have used the following types of intrusion detection sources.

- ***Snort & perl****:* Snort [17] is an open source network intrusion detection system (NIDS) which is a type of intrusion detection system that detects malicious activity by monitoring the network traffic. This is a signature based intrusion detection system. This type of IDS tries to identify the intrusion by matching the attack pattern to the list of all possible known attack patterns. When there is a pattern matching the signature, an alert is generated and logged. Then we have a custom written Perl code that reads these log files and sends a custom intrusion message to the intrusion response engine.
- ***CPU resource monitor****:* This type of intrusion detection source is useful when the signature of the attack is unknown but the affected entity in our system is known. Entity denotes a resource or a service in a system. For example in our implementation we have developed a custom intrusion detection source that will keep looking for a CPU utilization of a specific resource in our system. When the CPU utilization of that specific resource shows abnormal behavior, a possible attack might have happened. Then the intrusion detection source sends out a specific intrusion alert to the response engine. Many custom intrusion detection sources can be created by the system administrator. This type of IDS can be made to monitor specific aspects of the system and generate alert when there is an abnormality.

For the purpose of testing the intrusion detection and response framework we have used a OLSR worm attack [15] on the Optimized Link State Routing (OLSR) protocol as an intrusion on a wireless ad hoc network. Worms usually exploit security vulnerabilities and use them to gain control of the system. This OLSR worm exploits a specific vulnerability in OLSR routing protocol. OLSR worm [15] was developed by Jacob Russell Lynch. The OLSR worm sends out some specially crafted OLSR packet that causes a buffer overflow on the target machines. And thus the worm takes control of the target machine.

To know more on the working of the worm and the propagation methods on the network it is important to know the OLSR protocol and OLSR packet format.

3.3. Optimized Link State Routing (OLSR)

OLSR [22] is an IP routing protocol which is optimized for mobile ad-hoc network (MANET) but can also be used on other wireless ad-hoc networks. OLSR is best suited for network where the nodes keep moving around or join or leave frequently. OLSR is a proactive protocol, meaning the routes are available when they are needed unlike the reactive protocol where the routes are computed when needed. OLSR is also effectively used in networks that are very dense and large. This is mainly because of the optimization mechanism OLSR has, which reduces the overhead when flooding messages are sent through the network.

OLSR is an improved and optimized version of the link state routing (LSR) protocol. Optimization of the flooding mechanism is achieved by selecting a subset of nodes called multipoint relays (MPRs) to do the broadcasting in the network. A MPR is a node in the network. Each node will select a set of MPRs from its 1-hop neighbors. A set of MPRs for a node N will be able to take a message from a node N and can send it on to all the 2 hop neighbors of N. If a node wants to send a broadcast message, all its 1-hop nodes will receive the message but however only the MPRs will continue to broadcast the message. This greatly reduces the amount of congestion and bandwidth being used to send a message to everyone in the network.

3.3.1. OLSR packet format

OLSR communicates using a unified packet format for all the data related to the protocol. There are few fields in the packet header that require some explanation to understand how the OLSR worm exploits the protocol. For any OLSR implementation the following types of messages have to be supported. They are HELLO messages, Topology Control messages, (TC) Multiple Interface Declaration messages (MID). HELLO messages are the type of messages exploited by the OLSR worm to propagate. But going further, It is also necessary to understand the basics of OLSR packet format.

The OLSR packet format is shown on Fig. 2. The first two fields in OLSR packet are packet length and packet sequence number. The packet sequence number is incremented every time an interface transmits a new packet. These two fields make up the packet header. In the OLSR implementation, the packet header is included after the TCP and IP headers. This is because our implementation runs on the application layer. Immediately following the packet header is the message type which describes the type of message included next. The Vtime is used to let the receiving node know how long the information within the message must be considered to be valid. Message size is measured in bytes and it includes the message header. The originator address field is filled with the main address of the node that sends the message and it is not changed if the packet is retransmitted unlike the source address in the IP header.

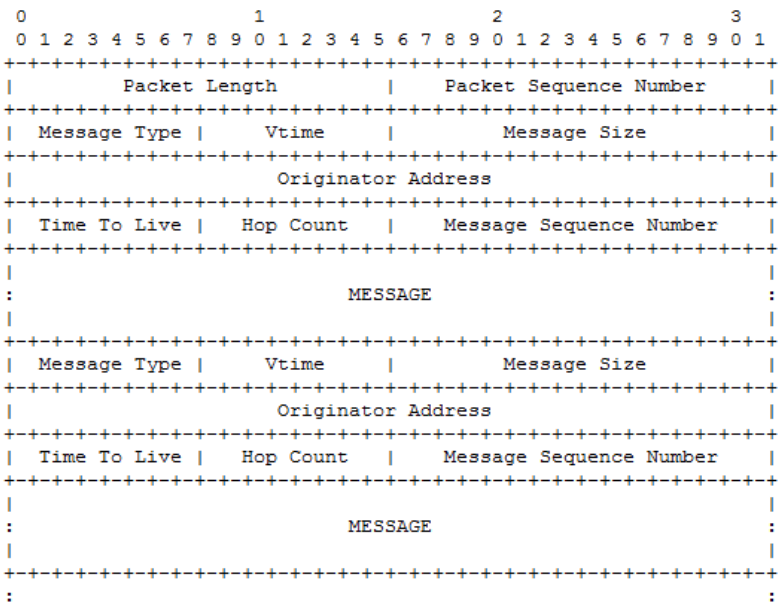

Figure 2. OLSR packet format [22]

OLSR transmits HELLO messages when a connection is setup between two computers, which establishes the symmetric link, such links are added to the routing table. A computer worm could use the HELLO message to invade a computer running OLSR. If the OSLR was

running on a wireless network it can be easy for the attacker to use the routing table to find its targets.

3.3.2. HELLO messages of OLSR

HELLO messages are the most important part of the OLSR. These HELLO messages are sent out by nodes to discover its links. Each node must create a link set, which is the set of nodes, with which it communicates. Each link in the set has a status associated with it. The status can be symmetric or asymmetric. If there is a two way communication between two neighboring nodes then the link is symmetric. If a node can only receive from another node but cannot transmit to that node then the link is asymmetric. A HELLO message is sent out periodically by every node and should never be forwarded beyond 1-hop neighbors as it is only used to discover 1-hop neighbors.

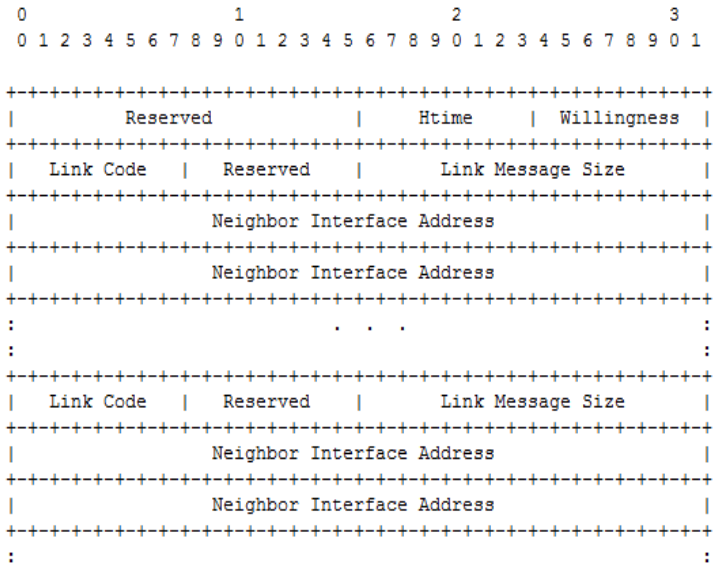

Figure 3. OLSR HELLO Message [22]

Fig. 3 shows the OLSR HELLO message format. The reserved fields are always set to thirteen zeroes. HTime is used to indicate the time interval before the node will transmit its next HELLO message. The willingness field indicates how willing the node is to forward the message. Link code has information of the interface to show if the link is symmetric or

asymmetric. The link message size field is the number of bytes starting from the previous link code field upto the beginning of the next link code field. The size is useful to include different sets of interfaces. It can also indicate the fact that there are no more following links. The neighbor interface address is the address of a neighbor node.

The next section describes in detail the basic activities of a computer worm. It also highlights the worm design used for testing purpose and the different stages involved during a worm attack.

3.4. Basic activities of an OLSR worm

Any computer worm is involved in three basic activities. They are indentifying target, propagating and attack activation [18].

Indentifying target is the first step for a computer worm. The worm becomes useless if it cannot find its target and propagate. There are many ways in which a worm finds its targets but it is not in the scope of this thesis. This OLSR worm uses the routing table to find its targets. This is because the OLSR protocol adds all other nodes with symmetric links to the routing table. So this worm can simply read the routing table and find all the potential targets within its transmission range.

The ***propagation*** is the way in which the worm makes a copy of itself from the attacker's machine to the target machine. In this case the OLSR worm uses a second channel propagation mechanism. This type of propagation method uses a separate communication channel apart from the one used to infect the target. TFTP is used to download a copy of the worm binary from the infected machine to the target machine, which requires a second communication channels. This is not the best way to propagate, but the OLSR worm was designed for the research purpose and there is scope for improvement of the propagation method.

The ***attack activation*** of the worm is the phase where the copied worm binary is activated. It can happen in different ways. One of the best ways to activate the worm is to perform a self-activation. This means that it does not need any human intervention. The OLSR worm uses this type of activation method. The worm from the infected machine after initiating a copy of the worm binary to the target machine sends out a set of shell commands, asking the target

machines to execute the newly copied binary. Thus the target machine has the worm binary activated.

3.5. Worm propagation

This section details the four main steps involved in the propagation of a worm from an infected host to a nearby target on a wireless network. It is assumed that every host on this network runs the OLSR protocol. As we have seen in the previous section, OLSR requires all hosts to periodically send HELLO mesages to its entire neighbor. All the participating hosts send HELLO messages on an open UDP port, 698 on a regular basis. Fig. 4 gives a pictorial representation of the worm propagation.

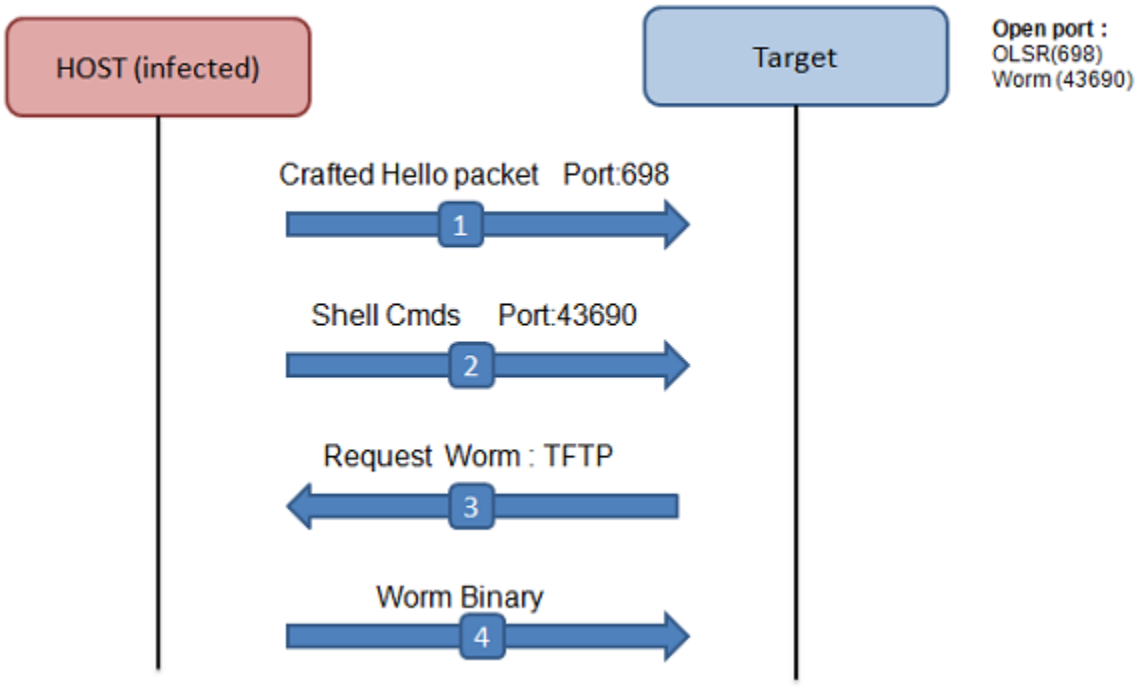

Figure 4. OLSR worm propagation

- ***Sending a crafted hello message****:* This is the first step that the worm does after finding a suitable target. All the hosts on a wireless ad hoc network running OLSR protocol send out

HELLO messages to all its neighbors to find all its one-hop and two-hop neighbors from their responses. This is because it is a mobile ad hoc network and all the nodes are mobile and any new node can enter the network and any node can leave the network at anytime. Thus both the host and the target will keep exchanging HELLO messages on a periodic basis. Now the worm sends out a crafted HELLO message to the target. The implementation details of the crafted HELLO message are given on Chapter 5. This crafted HELLO message will cause a buffer over flow on the target and will force the target to open the port 43690. This port is later used by the infected machine to establish a second communication link with the target machine.

- ***Shell commands****:* Once the port has been opened for communication via the crafted HELLO message, the infected machine sends a set of shell commands on port 43690 to the target machine to be executed on the shell prompt.
- ***Worm binary request****:* When the shell commands are executed the target machine is made to request for the worm binary from the infected machine. Then the infected machine sends the worm binary to the target through TFTP, which sets up a second communication link between the infected and the target machines.
- ***Worm binary sent and executed****:* The worm binary is sent from the infected machine to the target machine. Now along with worm binary, several shell commands are sent to the target machine. These shell commands execute the copied worm binary and now the target machine becomes a newly infected machine. By ‘infected’ we mean that the worm terminates the OLSR daemon running on the infected machine. Then this newly infected machine continues to infect all its neighbors. Thus the worm finds its targets, copies itself to the target machines using TFTP and activates the worm binary copied using the shell commands sent on port 43690.

Most worms are capable of infecting all machines in a network in a very short span of time. A properly designed worm can infect all vulnerable computers in the Internet in an hour or less[19]. Humans are not capable of reacting quickly enough to the fast-paced worm. This is the reason we need an automated intrusion detection and response system. Chapter 4 details the automated intrusion response engine developed to stop such worm attacks.

CHAPTER 4. INTRUSION RESPONSE ENGINE

This chapter provides the design details of the intrusion response engine. The subsystems of the response engine are listed and the functionality of each subsystem is explained. It also gives a detailed description on the dependency graph that is used to divide the whole system into smaller units of services and resources as well as the relationship between them. Then the value propagation method is introduced and illustrated with an example. It uses the dependency graph to propagate the total value of the system to every resource and service present in the system. Section 4.3 and Section 4.4 outline the cost sensitive response selection module and the response deployment module, respectively.

4.1. Overview

The response engine is responsible for receiving the intrusion alerts from the intrusion detection sources. Then it performs a cost sensitive analysis of the intrusion and deploys the suitable response to stop the intrusion. Fig. 1 shows that the intrusion response engine has three modules. They are the initialization module, the cost based response selection module and the response deployment module.

Initialization module is responsible for providing the response engine with all the information about the system security policies in terms of confidentiality, integrity and availability. It also has a value propagation function that does the propagation of the total value of the system to all the resources and services in the system. The initialization module reads all information regarding available resources, services and system security policies of the system from an XML file.

Cost based response selection module uses all the information provided by the initialization module on the system security policies and comes up with the best response for every intrusion alert it receives from the intrusion detection module. The core cost based response selection methodology for selecting the best response was developed by Christopher Roy Strasburg [13]. We have used this cost based selection response methodology in our intrusion response engine.

Response deployment module deploys the selected response. It manages the responses that are available for a particular system and triggers a response when it is selected by the cost based response selection module. The responses are currently implemented as Linux shell scripts. But in the future all these responses can be extended to suit other architectures like Windows OS. More implementation details are provided on Chapter 5.

Now we provide a detailed description of the three modules of the response engine

4.2. Initialization module

This module initializes the response engine with the security goals given to all resources and the services of the system. It also initializes the list of possible intrusions that can affect the system and the list of possible responses that can be used to protect the resources and services of the system. This information is hand coded as an XML file, which contains all the information about the dependency graph. We first describe the different entities of a system.

4.2.1. System entities - A layered approach

A system is a generic name given to a collection of hosts communicating with one another using the Internet. The network can be statically configured using some networking devices like router and switches or it can be dynamic as in the case of wireless ad-hoc networks. Wireless ad-hoc network represents a self-configuring network of mobile devices connected through wireless links.

Such a system contains many hosts running different applications, which are supported by resources of the system. Let *S* denote the set of all the services of the system. Each service of set *S* can be an instance of application service, component service or a system/support service. So applications of a system can be broadly categorized as one of the following services:

· Application Service (AS)

· Component Service (CS)

· System/Support Service (SS)

These applications of the system are supported by a set of resources of the system. Let R denote the set of all resources of the system. Each resource of the set R can be one of the following

· Virtual Resources (VR)

· Physical Resources (PR)

Thus any system can be divided into a multi-layered structure, each of which is explained in detail in the following sections. For the sake of convenience we denote all the services and resources of a system as entities *e*.

- ***Application Services*** - These are the services the system administrator and the business analyst really care about. These services have intuitive value in them. Some examples of the application service are web service that runs on a host or DNS hosting service that runs on a server (host) or a VoIP service.
- ***Component services*** - Component services are those that are visible to the user but not directly used by the user. These services are typically the sub-functions of a bigger application service. For example a VoIP application service is composed of different component services like a VoIP record and replay service, message sending and message receiving service etc.
- ***System/Support Services*** - System/support services are the services that support the upper layer component services or application services. They are not visible to the user. For example Secure Socket Layer (*SSL*) service is used to support upper layer application service such as a web server that uses *https* connections with its clients. In case of wireless ad-hoc networks the routing protocols like OLSR which we have discussed in the previous chapter is a system service. A network subsystem, file management systems and the process subsystem are classified as system/support services.
- ***Resources*** - These are the system entities that support all the above defined system services. They are classified as Virtual Resources (VR) and the Physical Resources (PR). In other words these resources are the building blocks of any service which need resources for its normal functioning. The virtual resources include files, sockets, device drivers etc. They are the objects or the software used as an interface with the hardware of a host. The physical

resources are the hardware such as hard disk, network interface card, motherboard etc. In the case of wireless ad-hoc network environment the virtual resources include the wireless device driver that interfaces with the hardware - wireless network interface card which forms the physical resource.

Now we categorize the system into different layers each containing multiple entities. In the following sections we will introduce the concept of dependency graph that defines the relationship between entities. This dependency graph is also used to propagate the value of the system to all the entities *e* of a system.

4.2.2. Dependency graph

Most IDS modules identify and detect an intrusion at the service level or at the resource level. In order to deploy an appropriate response for an intrusion we have to estimate the cost of intrusion. Furthermore the system administrators have to determine the cost of such response that will be deployed to stop the intrusion. We also need to take into account the impact of those deployed responses on the system as a whole. This whole process of selecting a response for an intrusion depends on the values assigned to each and every entity of the system.

Usually we depend on the system administrator's intuition in assigning specific value for every entity present in a system. This assignment may not always be accurate. Also, in case of a large system it becomes hard for the system administrator to estimate and assign values of each and every resource and service on the system.

To solve this problem and to make the cost assignment process smooth and accurate, we propose the dependency graph structure and the value propagation function. A dependency graph is one that will be used to show the functional dependencies among all the services and resources of the system. A value propagation function is one which when given the value of the level entities, will propagate that value to the lowest level entities using the dependency graph.

The value of the top level entities can be obtained from the system administrator or the business analyst as these are the entities that are visible to them to estimate a value. By top level entities we refer to all the application services of the system. For example, a business

analyst or a system administrator can come up with the value of an application level service such as a web service from the loss or the cost incurred by the organization when the web server stops functioning. Now this value of the web service can be propagated to all the services and the resources that support this web service using the value propagation function to be described in Section 4.2.3.

Dependency graph is defined as a pair *(V,E)* where *V* is set of vertices and *E* is the set of edges. Any vertex $v \in V$ in the dependency graph is a tuple of the form *(C,I,A)*. We will use *v*[C] to denote confidentiality which is the first element, *v*[*I*] to denote integrity which is the second element and *v*[*A*] to denote the availability which is the last element in the tuple for the vertex *v*.

We also use $V_C = \{ v[C] \mid v \in V \}$, $V_I = \{ v[I] \mid v \in V \}$, $V_A = \{ v[A] \mid v \in V \}$. And we define the edge relation $E \subseteq Z \times Z$ where $Z = V_C \cup V_I \cup V_A$. To simplify the notation further we denote X = { C , I , A}.

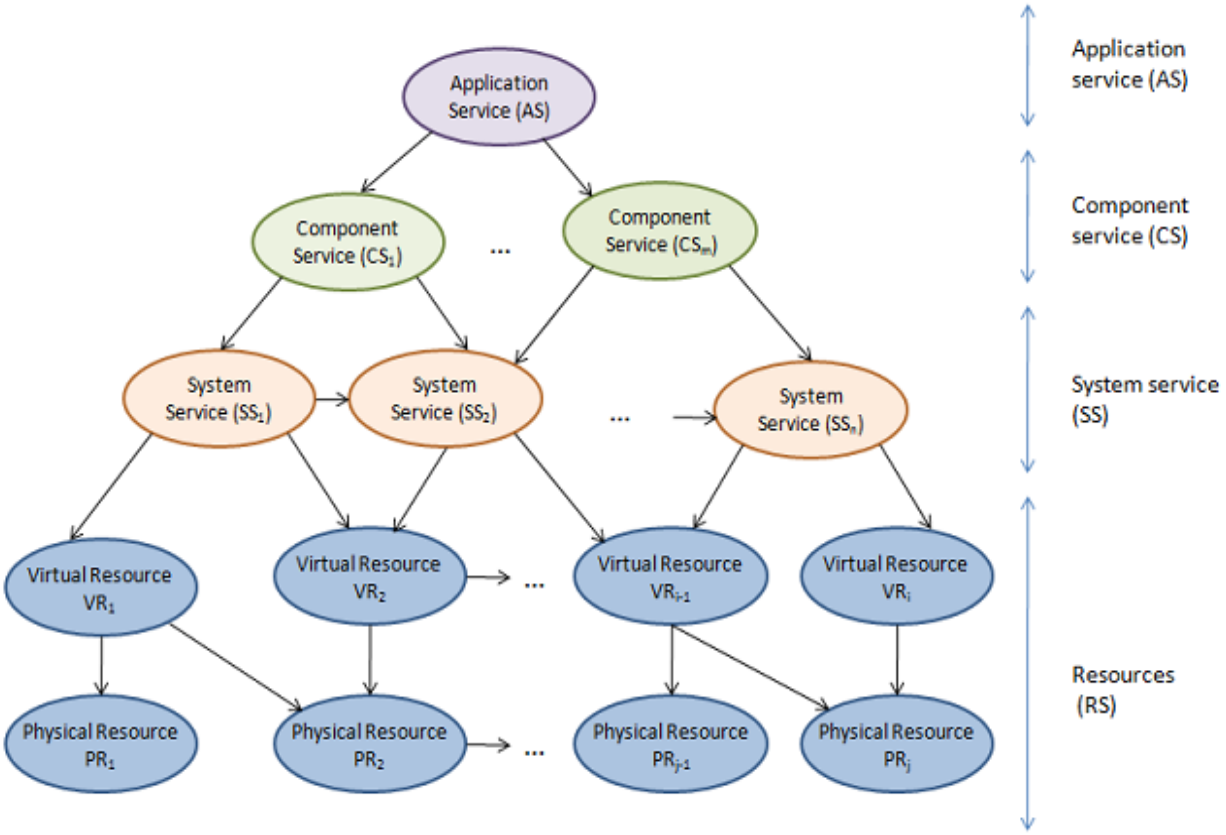

Figure 5. Dependency graph showing different entities of a system

Fig. 5 shows a dependency graph representing the dependency relations between entities of a system. An entity v_i is dependent and has a *dependency relation* on another entity v_j if and only if there exists an edge in E from $v_i[x]$ to $v_j[y]$, where $x, y \in X$. In Fig. 5 there is an application service (AS) that has dependency relations with the lower level component service $CS_j, j \in \{1,....m\}$. Similarly all the component services CS_j have dependency relations with its lower level service, namely the system/support services (SS). All these services depend of the resources (RS) and have direct or indirect dependency relation with virtual or physical resources of the system.

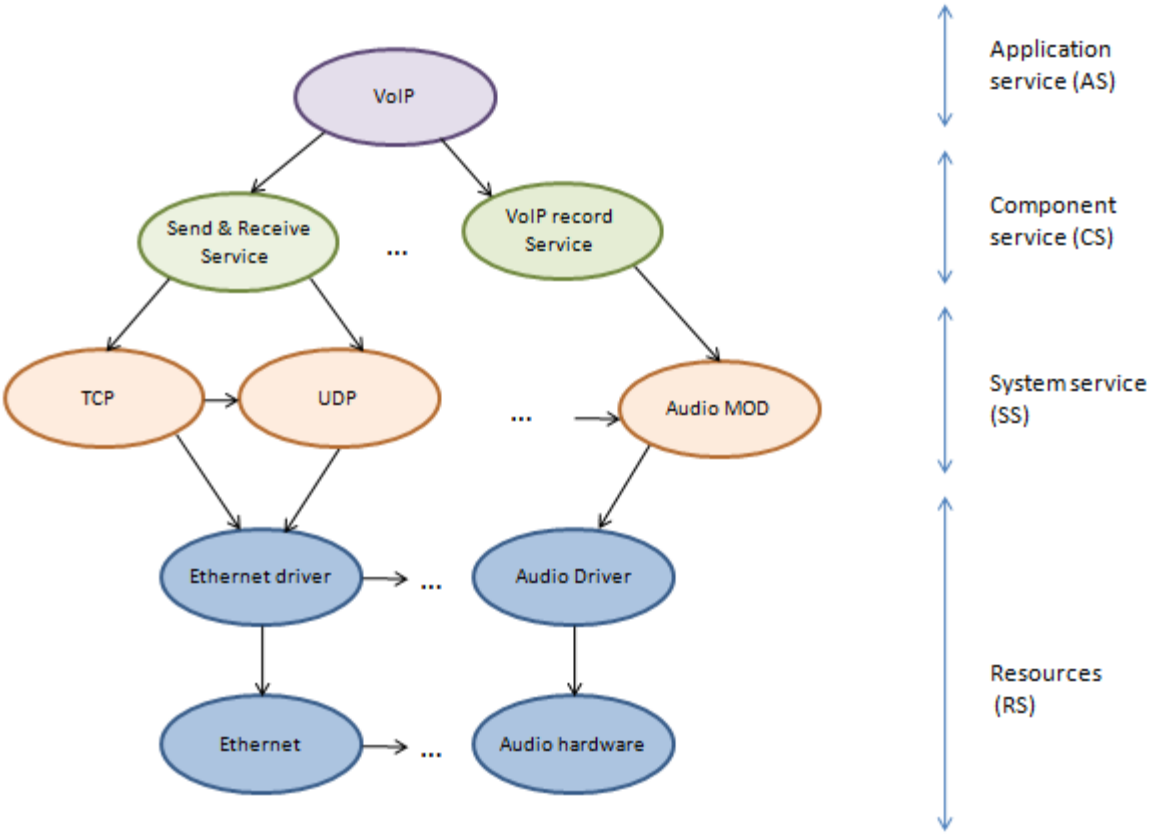

Figure 6. Dependency graph - VoIP example

Fig. 6 illustrates a dependency graph constructed using VoIP service as an example. The VoIP service forms the application service of this dependency graph. This application service depends on several component services like VoIP send and receive service and VoIP record service. These component services are in turn dependent on several system services. The VoIP send and receive service needs system services like TCP and UDP protocol for its normal functioning. Similarly, the VoIP record service needs system service like audio module. These system services need resources to support them. The TCP and UDP protocols use the virtual resource ethernet driver to communicate with the ethernet network interface card and audio module needs virtual resource audio driver to communicate with the audio hardware. The ethernet network interface card and the audio hardware form the physical resources of this system.

4.2.3. Value propagation method using dependency graph

When we define the dependency relation among the services and resources of the system we do not specify the degree to which they depend on each other. Also we do not define the dependencies in terms of system security goals like confidentiality, integrity and availability. We know that different systems with different security goals have different dependencies between services and resources.

For example the dependency relation in terms of security goals between a web server using *https* connection with a SSL service can be different from the situation that does not use *https* connection. So we need some variables to denote the *inter-dependency* relations in terms of security goals among entities. To denote these *inter-dependencies*, weights are assigned to each dependency edge. For example the dependency weight between $v_i[x]$ *and* $v_j[y]$ where $x, y \in X$ is denoted as $W_{vi[x]->vj[y]}$.

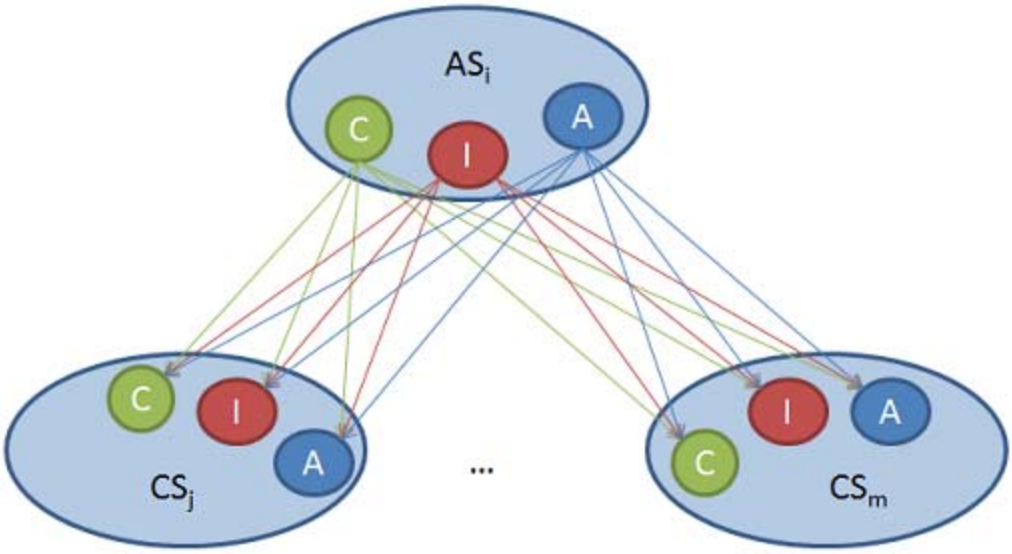

Figure 7. Example of inter-dependency weight assignment

Fig. 7 has an application service AS_i which depends on lower level component services CS_j , $j \in \{1,....m\}$. Here the availability of the application service AS_i depends on all the three security goals of the lower level component services. This means that $AS_i[A]$ has 3n edges coming out of the vertex. Each of those edges point to any one of the $CS_j[x]$, $x \in X$. Each of those 3n edges get an equal weight with a sum equal to 1.

Therefore, the weight for each edge is calculated as $WASi[A]->CSj[x] = 1/(3m)$ where $j \in \{1,....m\}$. Similarly other weights can be used. It is the responsibility of the system administrator to assign the cost of the interdependency weights. This is because they are the experts who know the system and its security goals.

We have the dependency graph created with all those interdependencies between all the entities defined. We have also assigned values for all the application services in our system by the business managers and business analysts. In order to propagate the value of the application services to all the entities of the dependency graph we use the value propagation function.

Before we get into the details of the propagation function we need to define two more important definitions, namely the dependent value and the intrinsic value.

Intrinsic value of an entity is defined as the value of the inner functionality of that entity in the system. *Dependent value* of an entity is defined as the value of an entity that is propagated down to its lower level entities on a dependency graph.

Let C_v denote the value of a vertex, $C_v(I)$ denote the intrinsic value of the vertex v and $C_v(D)$ denote the dependent value of the vertex v then,

$$C_v = C_v(I) + C_v(D)$$

This means that we divide the value of any entity into two parts namely the intrinsic value and the dependent value. This definition is reasonable because when the lower level entity fails it affects the upper level entity partially only.

For example in the case of a web service, when a lower level network service goes down or is attacked the availability of the upper level service is downgraded to half its capacity. That is the intrinsic value of the upper level service is not affected and only the dependent value on the lower level network service is lost.

To illustrate the calculation, we can assign percentage P to derive the intrinsic value from the total value of an entity. So the intrinsic value and the dependent value can be calculated as follows

$$C_v(D) = C_v * P$$

$$C_v(I) = C_v * (1 - P)$$

We want all the values of the application services to be propagated down to the lower level entities in the dependency graph so the percentage P = 1. When P=1 in the equations above the dependent value gets all the value of the entity and the intrinsic value of the entity becomes zero. Similarly all the lowest level entities on a dependency graph will have no more dependency. So they are assigned percentage P = 0, which means that dependent value of those entities are zero and the intrinsic value of those entities will be equal to the total value of the entity. For all intermediate entities on the dependency graph an appropriate value of P (where $0 \leq P \leq 1$) is assigned by the system administrator.

Now we introduce a procedural method used to propagate the system value. It has multiple steps involved and we illustrate each step in detail.

Finding the system value: As a first step, the system administrator works with the business analyst to get the system value. This may be expressed either in a static quantity i.e. dollars or in a quantity per unit time i.e. dollars per minute. This value is inferred from the direct revenue, profit and loss of the system. The metric used may differ from one system to another. Let us denote the total value of the system as C_s.

Dividing the system value among application services: As a second step, the application level services of the system are identified. Then, the overall system value is divided between the application services. For example an organization may have a web server and a mail server. Both these services may be equally important to the company. So the system administrator can divide the system value equally between these two services. We denote the web server as AS_w and the mail server as AS_m

$$C_w = 0.5 \text{ x } C_s \qquad \text{(Value of the web server)}$$

$$C_m = 0.5 \text{ x } C_s \qquad \text{(Value of the mail server)}$$

Dividing the value of application services between security goals of C, I, A: As a third step the value of each application service is divided into C,I,A values. This is done by the system administrator. In this example the integrity of the web service and the availability of the web service take the total value of AS_w into two equal halves. In the case of the mail server let us assume that all the three security goals take one third of the value of AS_m.

Web server (AS_w)

$$C_{w[C]} = 0.00 \text{ x } C_w$$

$$C_{w[I]} = 0.50 \text{ x } C_w$$

$$C_{w[A]} = 0.50 \text{ x } C_w$$

Mail server (AS_m)

$$C_{m[C]} = 0.33 \text{ x } C_m$$

$$C_{m[I]} = 0.33 \text{ x } C_m$$

$$C_{m[A]} = 0.33 \text{ x } C_m$$

Propagating the values to lower level entities on the dependency graph: Once the cost of the application service is determined we use the dependency graph to propagate the value to the lower level entities using a recursive function. The values of entities in a layer are determined by the entities of the same layer or the lower level entities. The value is calculated using the following formula.

$$\sum_{u \in V} (\sum_{y \in X} ((C_{u[y]} \times P_u) \times w_{u[y] \rightarrow v[x]})),\ x \in X.$$

In this formula *u* denotes the dependent upper level entity and *v* denotes the lower level entity. The cost propagation function first propagates the value of the application service to all the component service entities. Then the value of the component service is divided into two parts namely the intrinsic value and the dependent value. Then the dependent value of the component service is propagated to all the system/support entities. This propagation goes on till the lowest level entity is reached on a dependency graph.

For any system, we assert the following property holds.

$$\sum_{v \in V, v \notin AS} C_v = \sum_{u \in AS} C_u$$

This says that total summation of values of all entities without adding the application level services (represented by *v)* is equal to the sum of the application services (represented by *u)*. This is true because the total value of the system is divided between the application services. We know for all application services the percentage P = 1. This means that all the appplication services have just the dependent value and no intrinsic value. This dependent value of all the application services are propagated down to lower level entities using the value propagation method. So the above equation holds true.

4.3. Cost based response selection module

This module is responsible for the selection of the best response using the dependency graph and the other system security policies. In this section we introduce methods like damage assessment, cost evaluation in selecting a suitable response.

4.3.1. Damage assessment

Given a dependency graph for a system and the value of the application service we have propagated that value down to all the lower level entities on the graph. We now need a method to measure the intrusion cost on this system.

Before we determine the cost of the intrusion we classify the intrusion into different types. For different types of attacks the system administrator uses different metrics to estimate the intrusion cost. Using a penetration track of the intrusion on the dependency graph the system administrator will know the list of affected entities. Then the damage cost of the intrusion is the sum of all the values of damaged entities on the dependency graph.

Different intrusions will have varying impacts on entities of the dependency graph. These different impacts depend on the percentage of functionality loss on the security goals of each entity. Thus for every intrusion the percentage of impact on the security goals like confidentiality, integrity and availability can be measured. If we do not have sufficient information about any intrusion then we choose default values for damage assessment. Let us consider three different types of intrusion detections and their ways to evaluate the damage cost.

- ***Known attack using signature based detection system*:** As discussed in Chapter 3, an attack can be detected by matching its signature with a set of all available intrusion patterns. Snort is an intrusion detection system that works on this principle. By knowing the attack we can have some knowledge of the attack and the list of affected resources and services. Then we map those services and resources on to the dependency graph. Then we will have a list of entities that are damaged by that attack. Let us represent that list as v_i where $i \in \{1,....m\}$. Then the system administrator estimates the damage of each entity and comes up with the percentage of functionality loss for each of the security goals. Let us use $r_i[C]$, $r_i[I]$, and $r_i[A]$ to denote the loss of confidentiality integrity and availability for an entity r_i. Then the damage cost (DC) of the intrusion is calculated using this formula

$$DC = \sum_{i \in \{1,\ldots,m\}} \sum_{x \in X} (C^I_{v_i[x]} \times r_i[x])$$

- ***Unknown attack but affected entities are known*:** There are some cases where the attack is not known because the signature of the attack is unknown, however the system administrator still has knowledge of the affected services and resources. With a list of affected resources and services, we can map it to the dependency graph and come up with the list of entities affected on the dependency graph. Then we use the formula given above to find the damage cost of the intrusion. One example is a CPU resource monitor which is explained in Chapter 3.
- ***Unknown attack:*** This is when we have a new type of attack. The installed intrusion detection system is not able to detect the attack using any signature based technique. Also when all the resource monitoring detection systems fail to detect the attack we end up finding the top level application services affected. Then the damage cost in terms of security goals of those upper level entities can be easily assessed by the system administrator. For example, if a mail server is affected, the system administrator will estimate the cost as the loss to the company as a whole and will estimate the monetary loss because of the attack. This is then given some default damage cost and sent to response evaluation module.

4.3.2. Response cost evaluation

After the damage cost for the intrusion is calculated we have to define some cost evaluation for the response. The cost based model should give priority in selecting a response that has less operational cost and the minimum negative system impact. We define three important terminologies for response cost.

Operational Cost (OC) is defined as the cost incurred for deploying the response. Deploying a response requires some effort from the system administrator or the technical support person. For example, the work may include generating a report about the attack scenarios and the response deployed by the system administrator. This also includes the labor cost incurred by the organization in assigning a system administrator to analyze the situation.

Response System Impact (RSI) is the impact that the deployed response brings to the system. With the help of the dependency graph, the resources and the services affected by a response can be identified. Let S denote the set of all entities affected by the response. Let T denote the set of all entities affected by the real attack or the intrusion. Then the response system impact

is defined as the values of all entities affected by the deployed response but not damaged by the intrusion. The RSI is represented by the formula given below, where ‘d’ is the percentage by which an entity is affected by a response. It takes value of range [0,1].

$$RSI = \sum_{v_i \in S \& v_i \notin T} \sum_{x \in X} (C^I_{v_i[x]} \times d_{v_i[x]})$$

Response Success Factor (RS) is the measure of how effective the corresponding response will stop the intrusion from happening. The entities that are affected by the response are compared with the help of a dependency graph. Usually the response is said to have a full success rate if it can completely stop the intrusion from happening.The RS is represented by the formula given below, where ‘r’ is the percentage by which the entity is protected by a response. It takes value of range [0,1]..

$$RS = \sum_{v_i \in S \& v_i \in T} \sum_{x \in X} (C^I_{v_i[x]} \times r_{v_i[x]})$$

4.3.3. Response selection

So far, we have defined the ways to evaluate the damage cost of the intrusion, response cost and response success factor of a response. In this section we will provide a metric called expected value using which we can select a suitable response. The detailed way to evaluate expected value and its sub components are defined in [13].

Expected Value (EV) is defined as a measure of value gained by deploying a response when an intrusion occurs. Its value ranges between [-1,1]. This is calculated as follows

$$EV = RB - (RSI + OC)$$

If the expected value of a response is positive then that response is worth deploying. If the expected value is negative then it indicates that the response is not worth deploying and it will result in doing more harm than benefit to the system. RB stands for response benefit which is the amount of potential damage caused by an intrusion that will be stopped by a specific response. It requires the intrusion damage cost (DC) and response success factor as an input and tells us how well a response action will cover the damage cost brought about by an

intrusion. It takes a value between 0 and 1 with zero meaning the response will not cover any intrusion damage and anything greater than zero meaning the response covers some part of the intrusion.

RSI is the response system impact and *OC* is the operational cost of the response which we have discussed in the previous sections. These form the second part of the formula. The summation will be either 0 or 1 with zero indicating no cost incurred to deploy the response and 1 indicating the cost equal to the entire system value.

4.4. Response deployment module

This module is responsible for deploying the response. If the expected value (EV) is greater than zero, a response is worth deploying. This module chooses a response that has the biggest positive EV value from the list of responses. Then Linux shell scripts are triggered to deploy the corresponding responses. Some examples of the Linux shell script responses are adding a rule in firewall, killing an infected process and restarting the process, delaying a process, stopping particular type of traffic like TFTP traffic or blocking a port used by the attacker. Chapter 5 gives more implementation details on the responses deployed using Linux shell scripts.

CHAPTER 5. WORM IMPLEMENTATION & EXPERIMENTAL EVALUATION

This chapter provides the implementation details about the OLSR worm, the intrusion detection module and the intrusion response engine. It focuses on the implementation details of the dependency graph using XML. Section 5.4 of this chapter gives a detailed description of the test bed environment.

5.1. OLSR worm implementation

The OLSR worm uses the routing table for target machine discovery. We know that the OLSR protocol running on a host updates the routing table with details about connection to its neighbors on a wireless ad hoc network. It is easy for a worm to get the details of potential targets from the routing table instead of doing an exhaustive scanning for targets.

To transmit the worm from one computer to another the worm needs to exploit some vulnerability in the target machine. This vulnerability helps to setup a connection from the infected machine to the target machine, then transfer the worm to the infected machines and executes the worm in the target machines. For experimental purpose, we have added an exploit in OLSR protocol implementation. The *strcpy()* of the packet processing code of the OLSR has been modified to not check the length of the transmitted packet.

Now a specially crafted HELLO message is sent from the host machine to the target machine. This HELLO message has everything according to the specification upto the neighbor interface address in the HELLO message explained on Section 3.3.2. Then instead of including the neighbor interface address, the packet includes a special code that will cause a buffer overflow on the target machine using the *strcpy()* we added to the OLSR implementation. It copies all the data from the packet into the character buffer of the stack, which is not large enough to hold all the bytes of the transmitted packet. The stack also has a return pointer, which returns the control of the program after execution is done on the stack. This packet copy overwrites the return pointer of the stack, and makes the return address point to a new return address defined by the attacker. This causes the packet processing code to return to the new address specified by the attack machine.

A NOOP sled technique is used to perform this buffer overflow attack. A NOOP stands for ‘no operation’ command. Usually it solves the problem of finding the exact start address of the stack. When a processor executes a NOOP, it jumps to the next instruction. So when the program returns to the address where the NOOP sled resides. The execution slides through each NOOP till it reaches the shell code which is the code sent by the attacker inside the HELLO message.

These shell codes are just assembly codes translated into hexadecimal. Usually it depends on the architecture of the target machine. The main purpose of the shell code is to open up command shell in the target machine. The shell codes may vary from Windows machine to the Linux machine. The shell code in this attack is used to set up a connection that accepts unauthenticated connections on port 43690.

So once the connection is set up in the target machine by the shell code, the worm has to replicate itself from the host machine to the target machine. Using the socket connection on port 43690, shell commands are sent from the attacker machine to the target machine, which forces the worm binary to be copied onto the target machine via TFTP. This is how the worm gets replicated to the target machine. Now that the target machine has the worm binary, the attacker simply sends the execute command to the target machine. Then this sequence is repeated till all the machines in the neighborhood are affected.

5.2. Intrusion detection module implementation

This section will elaborate on the two intrusion detection sources that are implemented namely, the Snort & Perl source and the other being the CPU resource monitor tool.

Snort & Perl detection source: Snort is an IDS that will keep monitoring the network traffic and will raise an alarm when it detects abnormal activity. So in order to feed the network traffic to snort we use tcpdump utility [20]. Tcpdump is a common packet sniffer that runs under the command line. The tcpdump tool is configured to capture all the packets on a wireless interface and directs the packets to a FIFO pipe.

tcpdump -i ath0 -s 2048 -w /tmp/kismet_dump

This command line directs the tcpdump utility to capture all the packets that go via wireless interface *ath0* and dumps it into a FIFO pipe called *kismet_dump.* Now snort is configured to

read the packets dumped by the tcpdump utility from the FIFO pipe and then scans for known attacks based on their patterns. Snort can also be configured to scan for user-defined patterns using custom rules. In this implementation, there is no need for any custom rules because by default snort has rules to detect the NOOP sled attack which is caused by the worm. When an attack takes place snort logs those alerts in a log file. Snort is configured on a fast logging mode using the following command.

snort -c /etc/snort/snort.conf -r /tmp/kismet_dump -A fast

This command directs the snort to read from the FIFO pipe called *kismet_dump* and uses the configuration setting defined on *snort.conf* to detect possible attacks. Now we have a Perl module that keeps looking for specific newly logged alerts from snort alert log files. When the Perl module finds an alert logged that is related to our system, it constructs a custom alert message and sends it to the intrusion response engine. The syntax of the custom message is described in the later section of this chapter.

CPU resource monitor source: This is a custom developed tool that is used to monitor the CPU utilization of system entities. This tool is configured to monitor the OLSR process. The tool checks and makes sure that the OLSR process does not exceed the CPU utilization over a certain threshold. When CPU utilization of OLSR process exceeds that threshold a custom alert is sent to the intrusion response engine. This is because when a worm attack happens on a target machine, it sends CPU utilization of OLSR process running on that system from a normal usage to an abnormal usage. This is a possible indication that the worm has infected a particular machine. These types of custom intrusion detection sources can be developed by the system administrator and can be added as the intrusion detection framework.

Using these two intrusion detection sources, the worm attack is identified and alerts are sent to the intrusion response system. The custom messages sent by an intrusion detection source have four arguments INAME, IID, IPOSS and IPROB. Intrusion name (INAME) is the name of the intrusion. Unique identification value (IID) is the unique identification number given for every intrusion. The number of possible intrusion (IPOSS) is the set of all possible intrusions. The probability that a specific intrusion occurs is denoted as IPROB.

So when all alerts with the same IID have been received by the response engine and added to the attack profile, the attack is sent to the cost based selection module to evaluate the best response in that context.

5.3. Intrusion response engine

The intrusion response engine is a server program that keeps listening on a specific tcp port (13299) and waits for intrusion alerts from intrusion detection sources. Before the response engine starts it has to be fed with all information concerning the system and its security goals. It also needs information about the list of possible intrusions and the corresponding responses, which is provided by the initialization module.

5.3.1. Initialization module

Fig. 8 shows the dependency graph of three hosts ad hoc wireless network environment. More details of the test bed are given in Section 5.4. This dependency graph is encoded as an XML file, which is currently hand-coded and it can be automatically generated in our future enchancement.

This dependency graph in Fig. 8 has a few application services running on the system. One of which is DNS application service. All of these application services depend on a component service called OLSR. This OLSR component service depends on three system services that run on three different hosts. These system services are the OLSR daemons. They are represented as olsr_ daemon1, olsr_daemon2 and olsr_daemon3 on the dependency graph. Every daemon running on a host depends on a virtual resource like a device driver. These are represented by three device driver entities, one for each host in the dependency graph. At each host the device driver assists in the communication with the physical resource like wireless interface card. So there are three wireless interface card entities represented in the dependency graph one for each host.

.

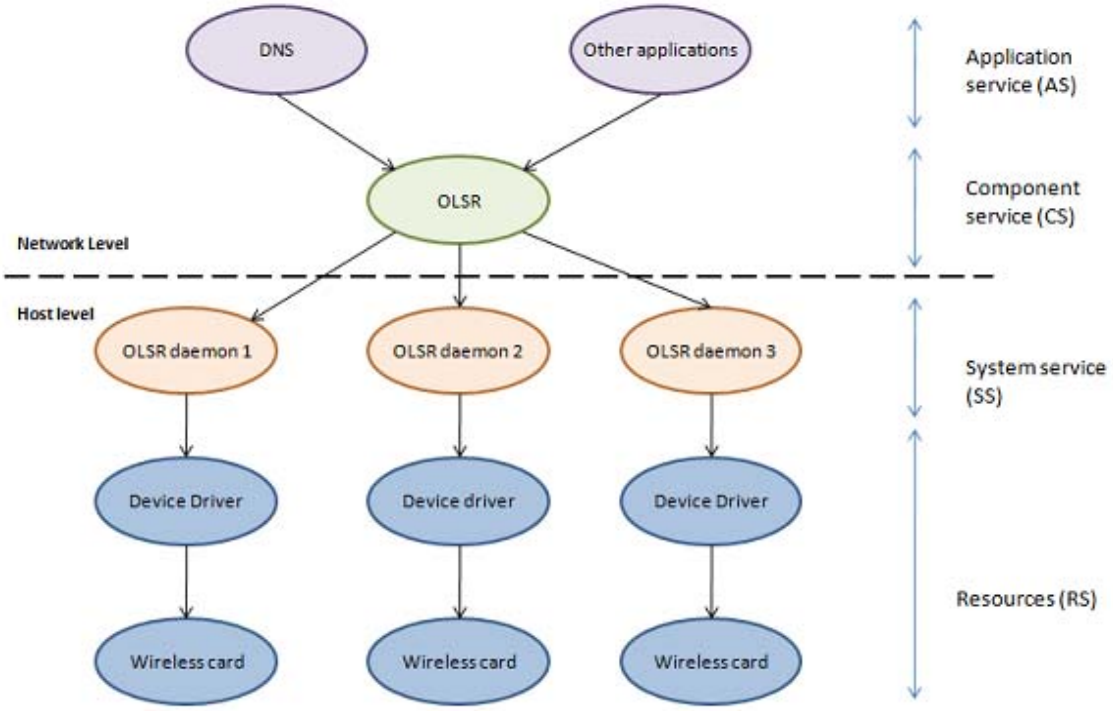

Figure 8. Dependency graph representing the test bed.

In the initialization phase the information about the system and its security goals has to be fed to the intrusion response engine. This information is encoded in an XML (eXtensible Markup Language) formatted file, which is given as an input to the response engine during its initialization. This XML file contains information about the dependency graph of the system, which has different sections that contain information about the system security policies. The sections are

· Available entities section

· Dependencies section

· Possible responses section

· Possible intrusions section

The *available entities section* provides a list of all available entities in the system. These entities are classified into five categories by the dependency graph. The application entities have the *C, I, A* values in the XML file. The security goals in terms of *C, I, A* values for all other entities are calculated using the value propagation function. Below is an example to represent an application service called DNS with its security goals defined as.

```
<AS1 NAME="DNS" C="0.4" I="0.3" A="0.3" Percentage="0.50" P="1.0" />
```

The *dependencies section* of the XML file gives information on the interdependency between entities in a system. For example, if the application service AS1 e.g. “DNS” is dependent on a component service e.g. “OLSR”, there is an entry in the section of the XML file as shown below. The value WCc= “0.5” denotes that the confidentiality of DNS depends on the confidentiality of “OLSR” by 50%. Similarly the value WIa = “0.1” means that integrity of DNS depends of availability of OLSR by 10%. These values are used by the value propagation function to propagate the value of the application level to the lower level entities.

```
<AS1 NAME="DNS">
    <DEPENDENCIES>
        <RESOURCE NAME="OLSR" wCc="0.5" wIa="0.1" />
    </DEPENDENCIES>
</AS1>
```

The *possible response section* of the XML file has information about the name and Operational Cost (OC) of the response. It also has a list of entities protected by the response and a list of entities damaged by the response. Each entity in the protected and damaged section has the corresponding C, I, A values. These values correspond to the percentage of impact caused by the intrusion. In the example below the response called blockAttackerIP protects one entity called OLSR and damages one entity called DNS. The values C=”0.2” in the protected resources section conveys that blockAttackerIP response protects the confidentiality of an entity called OLSR by 20%. The values of C, I, A in the damaged entities

conveys the percentage of damage. For example I=”0.3” says the response blockAttackerIP damages the integrity of entity called DNS by 30%

```
<R1 NAME="blockAttackerIP" OC="0.6" PAYLOAD="blockAttackerIP">
    <PROTECTEDRESOURCE>
        <RESOURCE NAME="OLSR"   C="0.2" I="0.0" A="0.6"/>
    </PROTECTEDRESOURCE>
    <DAMAGEDRESOURCE>
      <RESOURCE NAME="DNS" C="0.0" I="0.3" A="0.4"/>
    </DAMAGEDRESOURCE>
</R1>
```

The *possible intrusion section* of the XML file is similar to the possible response section, however it does not have the list of protected entities as it only has the list of damaged entities. It has name and Operational Cost (OC) of the intrusion. The C, I, A values corresponding to each of the damaged entities convey the percentage of damage this intrusion can bring to the security goal of that resource. For example C=“0.3” means that the intrusion wormattackInNeighbourhood damages the confidentiality of the entity called OLSR by 50%.

```
<I1 NAME="wormAttackInNeighborhood" OC="0.5">
    <DAMAGEDRESOURCE>
        <RESOURCE NAME="OLSR" C="0.3" I="0.5" A="0.66"/>
    </DAMAGEDRESOURCE>
</I1>
```

To read the information from an XML an open source library called TinyXML [21] is used, which is a simple C++ XML parser that can be easily integrated into other programs. It is a Document Object Model (DOM) based XML parser. The TinyXML parses the XML document and builds from that a DOM that can be read, modified and saved. Then the information from this XML file is read and is turned into data structures. This completes the initialization phase.

5.3.2. Response selection and deployment module

The response selection phase and response deployment phase come after the intrusion is detected. The response engine opens a TCP port and listens to input socket connection request from clients. When the clients send intrusion alerts, the intrusion response engine constructs

an attack profile structure and it is passed on to the response selection module. In the response selection module, all the cost computation is done and one best response is selected. This selected response is then sent to the response deployment module. Responses are currently implemented as shell scripts that are triggered from the C framework. The deployment module gets the information like traffic type, source IP address and destination IP address from the snort alert logs and passes them as arguments to the Linux shell scripts.

5.4. Test-bed setup

The test-bed setup consists of three machines running Red Hat Enterprise Linux 4.0 using kernel 2.6.9. Each machine has one wireless network interface card installed. All the three machines must be running the modified implementation of OLSR protocol. We use the Naval Research Laboratory's [22] implementation of OLSR and modify it so that the worm can propagate using the exploit we added in the regular implementation, which is called *nrlolsrd.*

In Section 5.1 we have described the *strcpy()* vulnerability that has been added to the OLSR implementation. Along with that we should also disable any rule in the firewall that stops UDP communication so that the UDP packets can be sent from one system to another. A new feature is added to many Linux distributions called *ExecShield*. This is aimed at reducing the risk of worm or other automated remote attacks on Linux systems. This is done by making the data memory of the stack as non-executable and program memory of the stack as non-writable. This feature has to be disabled as the OLSR worm attack implementation needs to execute a shell code that is on the stack.

The main focus of this implementation is to get a system level attack and check how our response system reacts to a system level attack. For this reason the *ExecShield* has been disabled and a buffer overflow attack using a tweaked version of the OLSR protocol has been put in place. Also in order to prevent the worm from spreading to the wild, it is first made to check for the presence of a path */go/yes*. The worm would then start infecting the target machines. This is mainly added to safe-guard other systems outside the testbed from being affected.

All the three systems have snort installed. There are multiple predefined rules built into snort to detect any kind of intrusions. A custom rule can also be added to snort to detect

specific intrusion. In this case we do not need any custom rule because there is already a rule in snort to check for long strings of NOOPs. In the worm implementation about nine-hundred consecutive NOOPs are sent across from the attack machine to the target machine.

Along with snort all the three machines use the tcpdump utility to capture packets on the wireless interface and dump them into a FIFO pipe. Then the snort is configured so that it can read from the FIFO pipe. When the worm sends the modified HELLO message from the attacker system, the NOOP rule in the snort installed on the target machine gets triggered and logs an alert indicating a possible attack. These alerts are logged in the log directory of snort in */var/log/snort/alert* file. Snort is configured to run in a fast logging mode so that we have control over the amount of data dumped onto the log file. This produces output that looks like the one shown below

```
03/25-22:07:32.047217  [**] [1:648:7] SHELLCODE x86 NOOP [**]
[Classification: Executable code was detected] [Priority: 1] {UDP}
192.168.1.225:34381 -> 192.168.1.226:698
```

Finally, we have a Perl module that keeps looking for all types of alerts in the log file. We use some regular expressions matching of Perl to parse each alert logged on *var/log/snort/alert* file and separate the important fields. The information is needed by the response engine to deploy a suitable response. For example the response module may need the IP address of the attacker to add a firewall rule asking it to block some specific traffic from the attacker IP address. So information like the name of the intrusion, time-stamp, priority of the intrusion, source IP address, source port, destination IP address, destination port and the protocol type are read from the snort log files and are sent to the response engine along with the alert.

CHAPTER 6. EXPERIMENTAL RESULTS

This chapter describes the attack scenario that is used to test the intrusion detection and response framework. Then the attack scenario is revisited with the intrusion response engines activated on all hosts. This chapter also provides details on the mitigation technique used by the response engine to stop the OLSR worm attack. Section 6.3 illustrates the performance of the intrusion response engine. We conclude the chapter by providing some information on possible enhancement to our generic response model.

6.1. Attack scenario without automated response

This section illustrates the propagation of the worm in the testbed consisting of three hosts. This will be useful to understand how our response engine responds to stop the propagation of the worm and keeps the OLSR protocol running on all the three hosts.

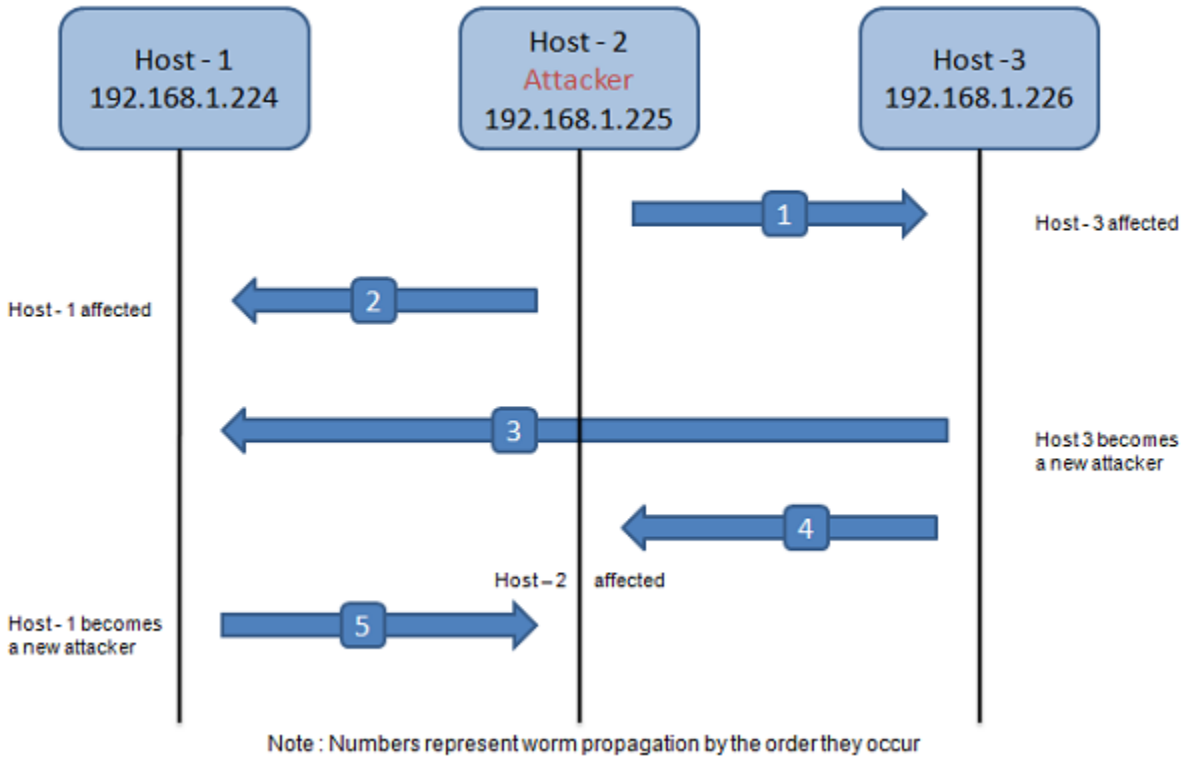

Figure 9. OLSR worm attack scenario without automated response

Fig. 9 shows the attack scenario of the worm using three machines. All these machines are one hop away from each other. Host-2 is the place where the worm attack is launched. The numbers on the arrows in Fig. 9 represent the steps by which the worm propagation occurs. The steps involved during the worm attack are explained in detail as follows.

- Step 1: Worm propagation occurs from host-2 to host-3. As a result, the OLSR daemon running on host-3 is terminated.
- Step 2: Worm propagation happens from host-2 to host-1. As a result, the OLSR daemon running on host-1 is also terminated. By end of step 2 all the neighbors of host-2 are infected.
- Step 3: Host-3 is infected and becomes an attacker. Worm propagation occurs from host-3 to all its neighbors. The neighbors of host-3 are host-1 and host-2. As a first step, worm propagation occurs from host-3 to host-1.
- Step 4: Worm propagation occurs from host-3 to host-2. As a result of this worm propagation the OLSR daemon running on host-2 is terminated.
- Step 5: Host-1 is infected and is becomes an attacker. As a result, worm propagation occurs from host-1 to host-2.

So by the end of the five step processes all OLSR daemons running on the three machines are terminated. This represents a huge loss to the organization as all the applications services will stop, including the DNS service, which depends on these OLSR daemons running on these three machines.

6.2. Attack scenario with automated response

Now we have our response engines installed on all the three hosts on the testbed. Fig. 10 shows the attack scenario with deployment of the response engine.

- Step1: Worm propagation happens from host-2 to host-3. This triggers two types of alerts generated by the IDS snort. The first alert is generated on host-3 and second alert is generated on host-1. The snort alert on host-3 informs the response engine on host-3 that there is a worm trying to infect the machine. As a result of this alert, the response engine on host-3 selects an optimal response that stops TFTP traffic and restarts OLSR daemon. When the OLSR worm infects any machine, the OLSR daemon running on that machine will be ter-

minated. This deployed response restarts the terminated OLSR daemon to keep it alive.

- Step 2: When worm propagation happens between host-2 and host-3, snort on host-1 triggers an alert indicating a worm attack has happened in the neighborhood of host-1. Along with that alert snort sends information (e.g. attackers IP address) to the response engine running on host-1. The response engine on host-1 selects the optimal response which blocks the attackers IP address.

Step 2 in Fig. 10 conveys the fact that when the worm tries to propagate from host-2 to host-1 the worm propagation fails. This is because the response engine on host-1 has already deployed the response that brings up the firewall blocking all traffic from the attackers IP address. Hence the OLSR daemon running on host-1 is not affected.

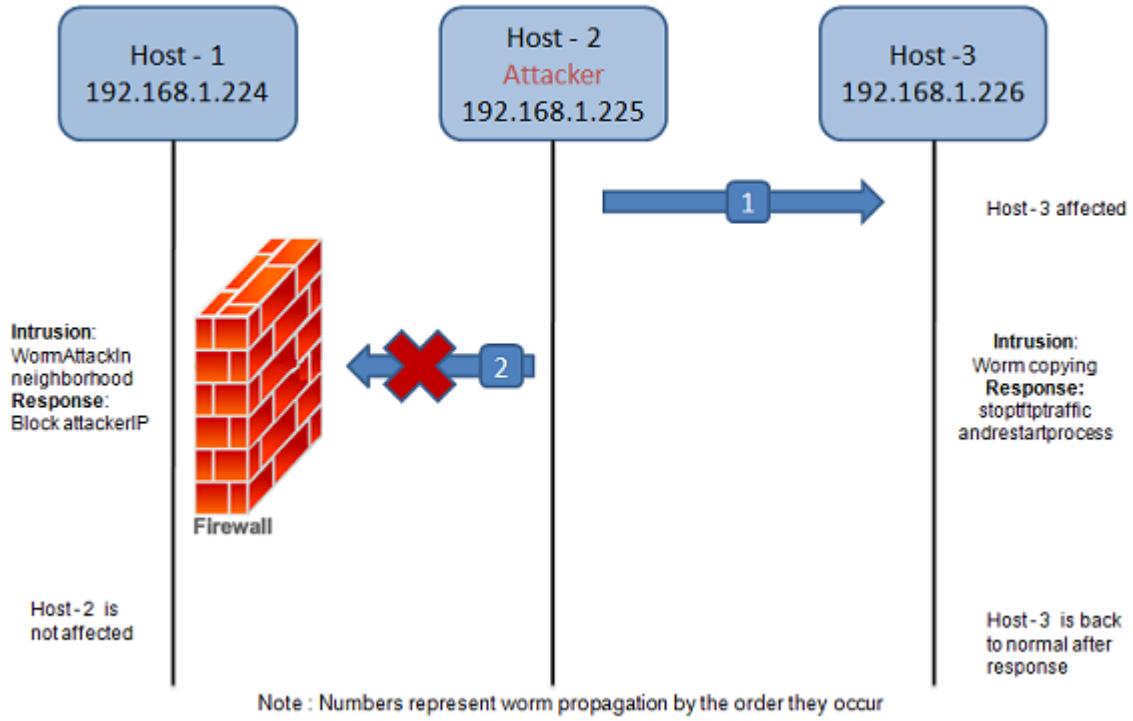

Figure 10. OLSR worm attack scenario with automated response.

The response engines installed on all the three systems respond so that even after the attack takes place, we have all the OLSR daemons running on all the systems which in turn keep all the application services alive. This saves the organization from a huge loss.

6.3. Performance metrics.

To evaluate the performance we have defined several metrics as follows.

Detection time is defined as the time need by the intrusion detection module to detect an attack.

Detection latency is defined as the latency between the detection of the attack by the intrusion detection module and the reception of the alert on the attack by the intrusion response engine.

Response selection latency is defined as the time taken for the response engine to select the best response on receipt of an intrusion alert. It is defined as the time difference between the reception of the intrusion alert by the response engine and the selection of the best response by the response engine.

Response deployment latency is the time taken to deploy the selected response. It is defined as the time difference between the selection of the response and the suceessful deployment of the response by the response engine.

6.3.1. Benchmarking the response selection time

Table 1 shows the results by benchmarking the response selection system based on the wireless ad hoc network. The entities, number of intrusion and number of system responses are varied and the different response time is benchmarked.

For example the 5000x in Table 1 conveys the fact that there are 5000 entities in the system, 5000 possible intrusions and 5000 possible responses available for the system. Thus these values are varied logarithmically from 20 to 10,000 and the four metrics defined above are calculated. The results are the average value of 10 experiments.

Performance metrics	20 'X'	100 'X'	500 'X'	1000 'X'	5000 'X'	10000 'X'
Intrusion detection time	0.008062	0.008389	0.025593	0.042691	0.040308	0.037484
Detection latency	0.109049	0.169358	0.16589	0.220554	0.180417	0.129542
Response Selection Latency	0.000589	0.012911	0.125345	0.157507	0.226155	0.428286
Response deployment latency	0.029504	0.036336	0.060171	0.047508	0.05341	0.096251

Table 1. Performance metrics on a host

The results are plotted on a graph and is represented below on Fig. 11

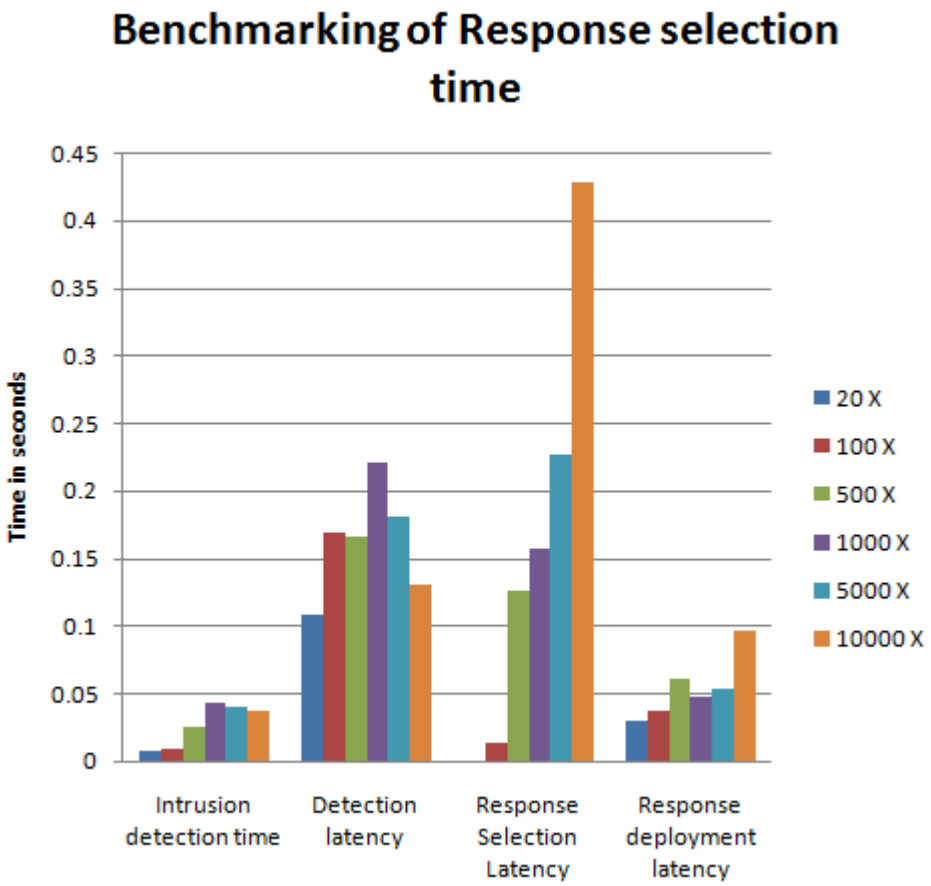

Figure 11. Benchmarking of response selection time

Fig. 11 conveys that there is no significant change in the response selection time even when we increase the number of entities, responses and intrusions to 10,000. The response selection is well below 0.5 seconds. This demonstrates that our response engine is highly scalable.

6.3.2. Cumulative response time

In order to show the efficiency of the total response time, we plot the graph with x axis having the cumulative value of the metrics we have defined in Section 6.3. For example cumulative response latency will be defined as the time difference between the start of the worm and the selection of the response. This graph helps us to understand the total time taken by our response engine to select a response and deploy it from the start of the worm.

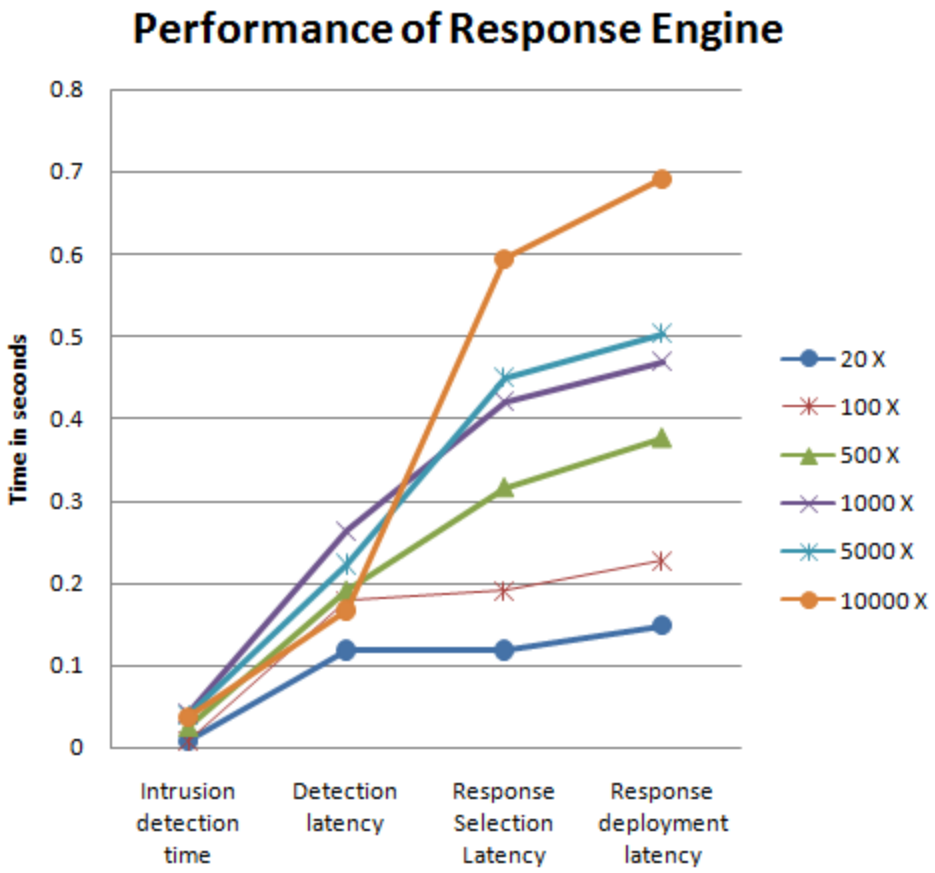

Figure 12. Cumulative response time

Fig. 12 shows that the total time taken to deploy a response is well within one second range even when there were 10,000 of entities, response and intrusion available. Attack time is defined as the time taken for the OLSR worm to infect a system and turn it into an attacker. The average attack time of an OLSR worm is 10 seconds. If we respond within that attack time we will be able to stop the worm from propagating to its neighbors. The results show that we respond on an average less than one second which stops the target machines from being infected.

6.4. Discussion

6.4.1. Benchmarking using different types of attacks

The benchmarking of the response engine is done only by using a worm attack on OLSR protocol. The number and types of attacks on this wireless environment can be varied and the response engine can be benchmarked. In order to detect different types of attacks we may have to add different types of IDS including anomaly based IDS. In order to have the ability to detect application layer attacks we can use the combination detection capabilities of fwsnort and psad discussed in Section 2.2.4. The response engine can use fwknop tool discussed in Section 2.2.5 to respond.

6.4.2. Dependency graph enhancement

The dependency graph used in this approach has a room for improvement. Some of the possible enhancements are described in the following sections.

6.4.2.1. Automatic generation of the dependency graph

Currently the dependency graph used in our response engine is manually created and is handcoded as an XML file. The generation of dependency graph can be automated. Given an application level service of a system there should be some methods which automatically list its dependent entities. It can also provide information about the interdependency between entities in a system, which can be done by using programs like *strace* and *ptrace* to track all the system calls made by application services. Then these information can be outputted as an XML file with the format specified on Section 5.3.1.

6.4.2.2. Visualization of dependency graph

The dependency graph generated can be visualized. This helps the system administrator to easily understand the status of the system. It can also help the system administrator to manually validate the interdependencies between entities in terms of system security goals, and can fine tune the interdependencies in accordance with the company policies. During the time of attack, the affected entities of the dependency graph can be highlighted. This would also help the administrator to manually respond when the automated response system fails.

6.4.2.3. Alternative value evaluation method

In Section 4.2.3 we divide the value of an entity into the intrinsic value and the dependent value. The dependency value is propagated down to other entities of the dependency graph. An alternative method of value propagation is based on the idea that the fuctionality of one entity is entirely dependent on other entities. This type of propagation can capture the situation where one service is solely dependent on other services. This will change the way in which we do the cost computation of intrusion damage and response impact.

6.4.3. Cross layer response

A cross layer based response can be evaluated using tools like fwsnort and psad. Most of the application level attacks can be detected using fwsnort using its string matching libraries. Then psad can be used to analyze the logs of fwsnort and deploy a network layer level response like adding an iptable rule to block a connection or tearing down a TCP connection of the attacker by sending an RST packet.

6.4.4. Communication between the response engines

The response engines are deployed on every host and currently there is no communication between them. Communication can be enabled between them so that the knowledge learnt about an attack on one response engine can be shared with the other response engine present on this distributed network. This communication ensures that the knowledge learnt is shared among them.

CHAPTER 7. CONCLUSION AND FUTURE WORK

In this chapter we summarize our contributions and present opportunities for future development and enhancements

7.1. Conclusion

In this project we have presented a generic framework for intrusion detection and response. We have implemented the dependency graph using XML technology. We have introduced a value propagation method which provides a way to propagate total system value to every entity of a system. We have also used the dependency graph for damage cost evaluation, response cost evaluation and evaluation of response effectiveness. Finally, all the metrics have been incorporated into the cost based automatic response engine to select the best response in the event of an intrusion.

The performance results have shown that the generic response framework is scalable and has a very fast response time. But the field of automated response is still in an infant stage and significant effort is needed to address challenges in this field of research.

7.2. Future work

This section focuses on some avenues for future enhancement and imporvement.

- Automatic generation of the dependency graph for the system.
- Developing metrics to measure the system security policies like availability, integrity and confidentiality. Developing standards to evaluate the system resources in terms of security policies.
- Providing a mechanism to transfer the knowledge of one response engine to another. Enabling communication between one response engine and another so that the knowledge thus gained by the response engine is shared across all hosts.
- Finally developing standards to measure the success of a selected response on different environments. This will help in comparing results of response selected between one environment and another.

BIBLIOGRAPHY

[1] T. Toth and C. Kruegel. “Evaluating the impact of automated intrusion response mechanisms”, in *the 18th Computer Security Applications Conference (ACSAC02)*, Las Vegas,NV, 2002, p. 301C310.

[2] I. Balepin, S. Maltsev, J. Rowe, and K. Levit. “Using specification-based intrusion detection for automated response,” in *the 6th International Symposium on Recent Advances in Intrusion Detection (RAID) 2003*, 2003.

[3] M. Jahnke, C. Thul, and P. Martini. “Graph based metrics for intrusion response measures in computer networks,” in *32nd IEEE Conference on Local Computer Networks (LCN)*, Dublin, Ireland, October 2007.

[4] N. Stakhanova, S. Basu, and J. Wong. “A taxonomy of intrusion response systems,” *International Journal of Information and Computer Security*, vol. 1, pp. 169–184, 2007.

[5] D. Ragsdale, C. Carver, J. Humphries, and U. Pooch. Adaptation techniques for intrusion detection and intrusion response system. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics at Nashville,Tennessee*, pages 2344–2349, 2000.

[6] C. Carver, J. M. Hill, and J. R. Surdu. A methodology for using intelligent agents to provide automated intrusion response. In *Proceedings of the IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop,West Point, NY, June 6-7, 2000*, pages 110–116, 2000.

[7] W. Lee,W. Fan, M. Miller, S. J. Stolfo, and E. Zadok. Toward cost-sensitive modeling for intrusion detection and response. J. Comput. Secur., 10(1-2):5-22, 2002.

[8] B. Foo , Y.-S.Wu, Y.-C. Mao, S. Bagchi, and E. H. Spaord. ADEPTS: Adaptive intrusion response using attack graphs in an e-commerce environment. In Proceedings of DSN, pages 508-517, 2005.

[9] Fwsnort, “Firewall snort,” [Website], Availalble: HTTP://www.cipherdyne.org/fwsnort/

[10] Fwknop, “Firewall Knock Operator,” [Website], Availalble: HTTP://www.cipherdyne.org/fwknop/

[11] Psad, “Port scan attack detector,” [Website], Availalble: HTTP://www.cipherdyne.org/psad/

[12] OLSR, “Optimized Link State Routing Protocol (OLSR),” [Online document], 2003 Oct., [cited 2006 Apr. 3], Available HTTP: http://www.ietf.org/rfc/rfc3626.txt

[13] C. Strasburg, “A framework for cost-sensitive automated selection of intrusion response,” *Master Thesis*, 2009.

[14] N. Stakhanova “A framework for adaptive, cost-sensitive intrusion detection and response system” *Phd thesis*, 2007.

[15] Jacob Russell Lynch **“**Intrusion detection systems in wireless ad-hoc networks: detecting worm attacks” *Master thesis*, 2006.

[16] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou. Specification-based anomaly detection: a new approach for detecting network intrusions. In Proceedings of the 9thACM conference on Computer and communications security, 2002.

[17] Snort, “Snort – the de facto standard for intrusion detection/prevention,” [Website], 2006 Mar 23, [cited 2006 Apr 3], Available HTTP: http://www.snort.org

[18] N. Weaver, V. Paxson, S. Staniford, R. Cunningham, “A taxonomy of computer worms,” in: *Proceedings of the 2003 ACM Workshop on Rapid Malcode*, 2003, pp. 11-18.

[19] S. Staniford, V. Paxson, N. Weaver, “How to own the Internet in Your Spare Time,” *Proceedings of the 11th USENIX Security Symposium*, 2002.

[20] “Tcp dump” [Online document]. Available:http://www.tcpdump.org/

[21] “Tinyxml” [Online document]. Availalble:http://www.grinninglizard.com/tinyxml/

[22] “Naval research laboratory” [Online document]. Available:http://cs.itd.nrl.navy.mil/work/olsr/index.php

[23] “OSSEC” [Online document]. Available:http://www.ossec.net/

ACKNOWLEDGEMENTS

I would like to express my gratitude to all my committee members for all their invaluable support, instruction and inspiration. First of all, I would like to thank my major professor, Dr. Johnny S. Wong for all his guidance and consistent support. He was a great source of inspiration and motivation throughout my work. I feel extremely lucky to get an opportunity to work with him. I would also like to thank Dr. Samik Basu for his guidance and inspiring discussions during our research meetings and Dr. Shashi K. Gadia for his willingness to serve on my program of study committee.

It was my pleasure working with my research group and I enjoyed having interesting discussions, meetings and e-mail exchanges with all of them. I specially thank Dr. Xia Wang for all her inputs and insightful feedback, Christopher Strasburg for his constant support and inputs towards my implementation and Tanmoy Sarkar for collaborating with me. Without their support and encouragement this work would not have been possible.

I am also grateful to all the people whom I met during my stay at Iowa state university. Special mention has to be made about the constant encouragement given to me by my roommates Valliappan, Bharath and Kartic. Finally I would like to thank my parents and my brother for their immense love and constant support throughout my life.