University of Louisville
ThinkIR: The University of Louisville's Institutional Repository

Electronic Theses and Dissertations

12-2004

Anomaly based intrusion detection for network monitoring using a dynamic honeypot.

Jeff Hieb
University of Louisville

Follow this and additional works at: https://ir.library.louisville.edu/etd

Recommended Citation

Hieb, Jeff, "Anomaly based intrusion detection for network monitoring using a dynamic honeypot." (2004). *Electronic Theses and Dissertations.* Paper 616.
https://doi.org/10.18297/etd/616

This Master's Thesis is brought to you for free and open access by ThinkIR: The University of Louisville's Institutional Repository. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of ThinkIR: The University of Louisville's Institutional Repository. This title appears here courtesy of the author, who has retained all other copyrights. For more information, please contact thinkir@louisville.edu.

ANOMALY BASED INTRUSION DETECTION FOR NETWORK MONITORING USING A DYNAMIC HONEYPOT

By

Jeff Hieb
B.S. Furman University, 1992.
B.A. Furman University, 1992.

A Thesis
Submitted to the Faculty of the
Graduate School of the University of Louisville
In Partial Fulfillment of the requirements
For the degree of

Master of Science

Department of Computer Engineering and Computer Science
University of Louisville
Louisville, Kentucky

December 2004

ANOMALY BASED INTRUSION DETECTION FOR NETWORK MONITORING USING A DYNAMIC HONEYPOT

By

Jeff Hieb
B.S. Furman University, 1992.

A Thesis Approved on

November 10, 2004

By the following Thesis Committee:

____
Dr. James H. Graham, Thesis Director

____
Dr. Melvin J. Maron

____
Dr. Gail W. DePuy

ACKNOWLEDGEMENTS

I would like to express my deep appreciation to my advisor, Dr. James Graham, for his assistance and support prior to and during the pursuit of this research. His technical guidance, encouragement, and perspective were essential to the completion of this thesis. I have also enjoyed our many conversations, and the lessons that came with them.

My deepest gratitude is also extended to the other members of my thesis committee: Dr. Melvin Maron and Dr. Gail DePuy. Their comments and suggestions made this a better thesis, and I appreciate their willingness to share their valuable time and allow me to benefit from their expertise.

I would also like to express my sincerest appreciation to Dr. Hayden Porter, Dr. James Edwards, and Dr. David Shaner. My undergraduate years at Furman University were fulfilling and rewarding in a large part because of these men; and also because of Duane Twardokus, who has been a great source of encouragement and inspiration for many years.

Finally I would like to thank my wife, Jennifer, who kept me going when no one else could. She has encouraged me in all endeavors and been there when ever I needed a friend; completion of this thesis would not have been possible without her.

ABSTRACT

ANOMALY BASED INTRUSION DETECTION FOR NETWORK MONITORING USING A DYNAMIC HONEYPOT

Jeff Hieb

November 10, 2004

This thesis proposes a network based intrusion detection approach using anomaly detection and achieving low configuration and maintenance costs. A honeypots is an emerging security tool that has several beneficial characteristics, one of which is that all traffic to it is anomalous. A dynamic honeypot reduces the configuration and maintenance costs of honeypot deployment. An anomaly based intrusion detection system with low configuration and maintenance costs can be constructed by simply observing the egress and ingress to a dynamic honeypot.

This thesis explores the design and implementation of a dynamic honeypot using a variety of publicly available tools. The main contributions of the design consist of a database containing network relevant information and a dynamic honeypot engine that generates honeypot configurations from the relevant network information. The thesis also explores a simple intrusion detection system built around the dynamic honeypot. These systems were experimentally implemented and preliminary testing identified anomalous traffic, though in some cases it was not necessarily intrusive. In one instance the dynamic honeypot based intrusion detection system identified an intrusion, which was not detected by conventional means.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii
ABSTRACT iv
LIST OF FIGURES viii

CHAPTER I INTRODUCTION 1

1.1 Background 1

1.2 Organization Of Thesis. 6

CHAPTER II LITERATURE REVIEW 7

2.1 Intrusion Detection Systems. 7

2.2 Honeypots 13

CHAPTER III THEORY AND DESIGN 21

3.1 Honeypots And Intrusion Detection. 21

3.2 Dynamic Honeypots 24

3.2.1 Passive Network Analysis 24

3.2.2 Virtual Honeypot Deployment 26

3.3 A Dynamic Honeypot Design 28

3.4 Intrusion Detection Using A Dynamic Honeypot 30

CHAPTER IV IMPLEMENTATION 35

4.1 Dynamic Honeypot Implementation 35

4.1.1 Gathering Network Information 36

4.1.2 Generating Honeypot Definitions. 43

4.1.3 Deploying The Honeypots 47

4.2 Anomaly Based Intrusion Detection 52

4.2.1 Reporting The Honeypot Traffic 53

4.2.2 An Additional Alarm Mechanism Based On Honeypot Traffic 55

CHAPTER V TESTING AND RESULTS 57

5.1 Testing Network Analysis 57

5.2 Testing Honeypot Configuration 63

5.3 Testing Virtual Honeypot Deployment 64

5.4 Testing The Intrusion Detection Abilities 68

5.4.1 Controlled Intrusions 70

5.4.2 Real World intrusions 74

CHAPTER VI CONCLUSIONS AND FUTURE DIRECTIONS 77

6.1 Conclusions 77

6.2 Directions For Future Research 79

REFERENCES 82

Appendix A 87

Appendix B 99

Appendix C 101

Appendix D 103

Appendix E 106

Appendix F 107

Appendix G 109

Appendix H.. 111

CURRICULUM VITAE.. 135

LIST OF FIGURES

Figure 1.1. Computer Security related losses for 2004 [62]. 1

Figure 1.2. Percentage of companies using various security technologies [62]. 3

Figure 2.1. The IDES Model [28] 11

Figure 2.2. Honeypot deployment on a DMZ [1] 15

Figure 3.1. Sample p0f output... 26

Figure 3.2. Sample Honeyd configuration. 27

Figure 3.3. Model for a dynamic honeypot design. 29

Figure 3.4. Design for dynamic honeypot intrusion detection.. 33

Figure 4.1. Host and port table definitions. 36

Figure 4.2. Configuration of the development and test network. 39

Figure 4.3. The use of interfaces in the dynamic honeypot. 40

Figure 4.4. Contents of the host, ports, and flock tables as a result of passive network analysis 42

Figure 4.5. Honeypots, honeyports, honeyhosts, and scripts table definitions. 44

Figure 4.6. Definition of default tcp, udp, and icmp actions in Honeyd. 48

Figure 4.7. Adding additional open ports and actions to Honeyd. 50

Figure 4.8. Complete Honeyd style honeypot definition. 50

Figure 4.9. Conditional binding of different honeypots to one IP address in Honeyd. ... 51

Figure 4.10. Complete honeypot definition. 51

Figure 5.1. Contents of the host, ports, and flock table following some passive network analysis 58

Figure 5.2. Additions to the flock, host, and ports tables after external connections to servers. 59

Figure 5.3. Contents of host and ports tables after the internal nmap scan. 60

Figure 5.4. Contents of the host and ports tables after external Nmap scan. 62

Figure 5.5. Contents of the honeypots, honeyhosts, and honeyports tables as a result of configuration. 65

Figure 5.6. Connection to the web server from an internal host. 66

Figure 5.7. Connection to the honeypot web server from an internal host. 67

Figure 5.8. Connection to the honeypot web server from external host. 68

Figure 5.9. Connection to various honeypot services from external host. 69

Figure 5.10. ACID console for the honeypots prior to any activity. 69

Figure 5.11. ACID listing of alerts after initial test traffic. 71

Figure 5.12. ACID view of an individual packet that caused an alert. 72

Figure 5.13. ACID view of an exploit packet. 73

Figure 5.14. ACID console after some external traffic is allowed to reach the honeypots. 75

Figure 5.15. Listing of the unique alerts generated while exposing the honeypot to the Internet. 76

CHAPTER I

INTRODUCTION

1.1 Background

Given the growing dependence of the American economy on information technology and the proliferation of networks, computers, and connectivity; securing computer systems is more difficult and more important. Fortunately, industry, the government, and individuals have begun to practice better computer security. In the *2004 Computer Crime and Security Survey* conducted by the Computer Security Institute and the FBI [62], the reported total loss in dollars was less than that reported in 2003. Unfortunately, the amount was still over $140,000,000, with over $55,000,000 alone attributed to viruses.

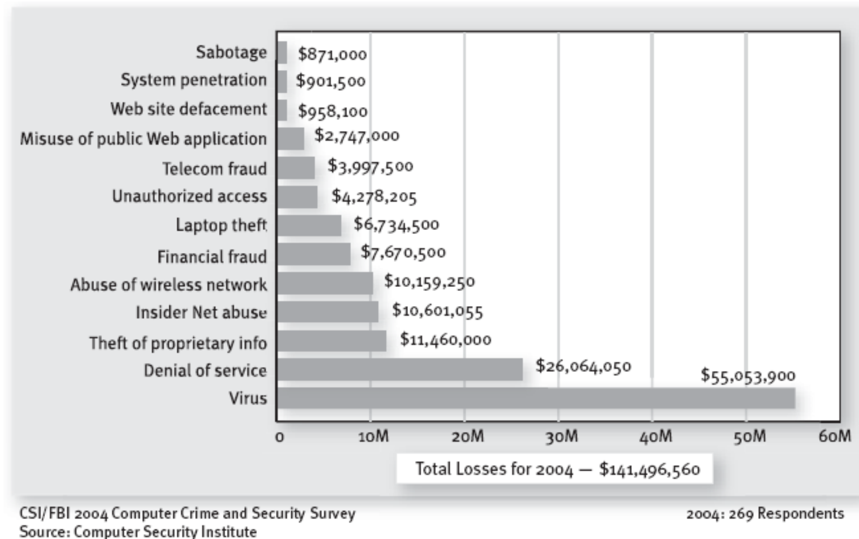

Figure 1.1. Computer Security related losses for 2004 [62].

Another area of concern is critical infrastructure and SCADA systems, and it is difficult to associate a dollar figure with compromises to these systems. Critical infrastructures and SCADA systems are ubiquitous; they involve everything from water and power to financial and logistic systems. In a recent article in *Information Security* titled "Mission: Critical", Stephen Barlas and other discuss cyber security and critical infrastructure [61]. A successful attack on one of these systems could be catastrophic. Imagine the consequences of a large scale and persistent outage of telecommunications networks, such an event could create panic and disorder, cripple the government's response capabilities, as well as do serious financial damage to a wide variety of companies. Similar scenarios are true for other critical infrastructures.

"According to SBC communications, the number of telecom vulnerabilities doubles each year," and in the financial sector "more than half of IT and security professionals . . . say they're unprepared for a cyber attack" [61]. While most agree that we are better prepared today than a few years ago, cyber security for critical infrastructures is an ongoing and never ending task that requires the continued development of newer and better security technologies.

The objective of computer security is to ensure the confidentiality, integrity, and availability of data or resources and good security is best achieved through the combined use of various security technologies. Examples of such technologies include firewalls, encryption, access control lists, and intrusion detection systems. As part of the CSI/FBI survey, information about the types of security technologies used was collected and is shown in figure 1.2. Firewalls and anti-virus software are the predominant security technologies in use today, being the only two security technologies used by almost every

respondent to the survey, however the use of intrusion detection is increasing. And as companies are required to be more accountable with respect to the security of sensitive data in their position, good intrusion detection will soon become a necessity.

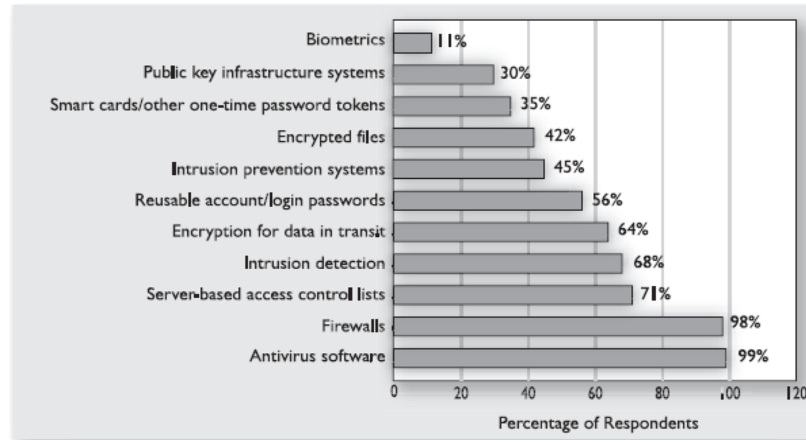

Figure 1.2. Percentage of companies using various security technologies [62].

Early work by Denning on intrusion detection systems identified two separate but equally valid approaches in detecting intrusions: anomaly detection and misuse detection. Misuse detection identifies an intrusion using a set of "rules" developed by analyzing known attacks. Anomaly detection identifies an intrusion based on a deviation from normal activity. Today's systems continue to use either anomaly detection or misuse detection, or some combination of both.

There are a variety of both commercial and public domain intrusion detection systems, Snort being one of the most well known in the public domain. Snort is also the basis for the commercial intrusion detection system Sourcefire. Snort, as do many other intrusion detection systems, uses misuse detection. It depends on a set of rules that define different types of known intrusion signatures. When the conditions of a rule are met, Snort generates an alert indicating that it has detected an intrusion.

Maintaining and updating these rules and responding to alerts are ongoing and time-consuming tasks, and if the rules become out of date, then the intrusion detection system becomes increasingly less effective. In addition to maintaining the rules, someone must respond to the alerts. Sometimes signatures may also match valid activity, meaning that responding to alerts first requires determining whether the alert is the result of an intrusion or unexpected, but valid, system activity. All of these require highly trained personnel to carry out.

Another problem faced by current intrusion detection technologies is bandwidth. As bandwidth continues to increase it becomes more and more difficult to capture and analyze the volume of information in an acceptable period of time (micro-seconds). When the bandwidth limits of an intrusion detection system are exceeded, it can fail to detect an intrusion. Current intrusion detection systems like Snort are effective; however, it is commonly held that anomaly detection will ultimately prove more valuable and robust because it has the potential to identify previously unknown intrusions or attacks.

Honeypots are a new security technology that, while not a replacement for traditional intrusion detection systems, address some of the weaknesses of intrusion detection systems. Because their only purpose is to be attacked, all traffic to the honeypot can be considered an intrusion or an anomaly of some sort. For this reason there is no need to separate normal traffic from anomalous; this makes any data collected from a honeypot of high value. Neither are they vulnerable to the bandwidth issue that more traditional IDSs face.

Honeypots do face several important challenges: 1) honeypots are totally unaware of attacks not directed at them, 2) they must avoid being fingerprinted because if an

attacker can easily identify honeypots their usefulness will be severely limited, and 3) like so many security technologies, they require configuring and maintaining by a knowledgeable person.

Lance Spitzner has recently put forth a new honeypot concept called a dynamic honeypot. A dynamic honeypot is a plug and play solution that configures itself to suit the network environment in which it finds itself. This makes the honeypot much simpler to use and maintain and improves the likelihood that a network intrusion will include a visit to one of the honeypots. Dynamic honeypots might also be more difficult to fingerprint, because they are properly configured and "unique."

Honeypots, because of their very nature, excel at detection. What makes them most attractive in the area of detection is the fact that they implement anomaly detection, and appear to do so very effectively. An intrusion detection system that uses a dynamic honeypot could potentially provide anomaly based intrusion detection. Such a system could be deployed on a production network and require very little maintenance and configuration. Both anomaly detection and low configuration and maintenance overhead are desirable characteristics for intrusion detection.

Dynamic honeypots have yet to receive a lot of research attention, having only been proposed in September 2003. This thesis will describe the design and implementation of an experimental dynamic honeypot and a simple intrusion detection system based upon this honeypot. The dynamic honeypot was able to achieve autonomous configuration and deployment of honeypots in a variety of simulated network environments. The intrusion detection system reported various anomalies and in

during one test detected an exploit attempt that was not detected by a conventional intrusion detection system.

1.2 Organization Of Thesis.

Chapter two presents a detailed literature review of intrusion detection concepts, principles and approaches. Chapter three discusses the design of the dynamic honeypot and an intrusion detection system based upon the honeypot. Chapter four describes the implementation of the dynamic honeypot and the intrusion detection system. In Chapter five, testing of the dynamic honeypot and the intrusion detection system are described in detail. Chapter six presents some conclusions and possible directions for future research.

CHAPTER II

LITERATURE REVIEW

This chapter gives an overview of relevant previous work by other researchers. It includes sections on intrusion detection systems and honeypots.

2.1 Intrusion Detection Systems.

Since the development of time-sharing systems in the 1960's the need for computer security has been recognized and studied [10, 30,36,31,32] and has lead to the development of a variety of security systems and approaches. Initial systems developed during the 1960's and 1970's focused on prevention by attempting to deny access to unauthorized resources [10,36]. For example: a user identification and password that would prevent unauthorized individuals from logging onto the system. For systems with many users, an access control matrix would prevent valid users from accessing files (or system resources) to which they had not been granted authorization. Beyond these types of measures, security officers were charged with assessing the security of the system, based in part on lengthy logs.

In 1980 Anderson [36] showed that a variety of threats could be addressed by analyzing audit trails. He began by identifying the following types of intruders or penetrators: *external penetrators*, *internal penetrators*, and *misfeasors*. *External penetrators* were those not authorized to use the computer at all. *Internal penetrators* were those who were authorized to use the computer but not authorized to use the

specific data or resource being accessed. This included two sub-categories, *masqueraders*, individuals who used someone else's user id and password, and *clandestine users*, individuals who evaded auditing and access control measures. *Misfeasors* were individuals who were authorized to use the computer and the specific resource, but who misused their privilege.

One example given by Anderson was detecting an *external penetrator* based on failed login attempts. Anderson went on to outline numerous security related audit trails and their relation to various threats. He also realized that large amounts of audit data, while potentially very useful for assessing and monitoring the security of a system, would overwhelm a security officer. At the same time storage was becoming cheaper, allowing audit logs to be moved online [45]. So Anderson began to explore automating audit trail analysis, and described a surveillance system that would collect and processes audit files and produce a daily report [28,36].

In 1987 Denning presented an abstract model of an Intrusion Detection Expert System called IDES [24]. IDES was a model of a real-time intrusion-detection expert system, meaning that it would process audit data as it was generated, and immediately inform the security officer of an intrusion. It was independent of any particular system, application, or vulnerability, and served as a general framework for an intrusion detection expert system. Denning intended IDES to be implemented on a separate, high performance, system allowing IDES the ability to process audit records in real time without interfering with the performance of the target system.

Denning's IDES model was based on the idea that exploiting (or attacking) a system involves abnormal use of the system and therefore an intrusion could be detected

from abnormal patterns of system use. To do this the IDES generated and updated profiles that represent normal system use. Audit records were then matched against the profiles using rule-based pattern matching. An anomaly was generated when the audit data fail to conform to the profile. This gave IDES the ability to detect a wide variety of threats, attacks, or intrusions, independent from any knowledge about the specific vulnerabilities the target system might have.

In 1989 Teresa Lunt [28] elaborated on the IDES model and begin development of an actual system. Lunt pointed out that Anderson's approach of detecting an external *penetrator* by auditing failed login attempts could be though of as looking for specific characteristic's of an intrusion in audit records, where as Denning's IDES approach looked for audit records that did not fit normal system/user behavior. The IDES architecture designed by Lunt included both of these approached in a loosely coupled system, as seen in figure 2.1. The statistical intrusion detection monitors subjects via audit records, identifying audit records that fail to fit with in the normal profile for that subject. The rule-based intrusion detection system examines audit data for known intrusion scenarios such as failed login attempts. These two approaches continue to define intrusion detection system approaches today and are typically known as *misuse detection* and *anomaly detection* [46,45].

Misuse detection is usually implemented with a rule-based system developed from knowledge about the characteristics of previous intrusions. These characteristics or descriptions are often referred to as signatures. Specific signatures are then matched against the data, in essence looking for evidence of an intrusion or attack, as in

Anderson's example of failed login attempts. Another example of this approach, presented by Ilgun [44], uses state transition signatures.

Anomaly detection searches for unusual events, often using statistical metrics to define "unusual." Anomaly detection begins with one or more "models" of user or system behavior that are built up over time. These models describe the normal behavior of the system or user. Deviations from the normal indicate anomalies that are then assumed to be an intrusion or attack. Different modeling approaches have included statistical methods [28,14], rule based systems [26], neural networks [27] and other soft-computing techniques [13].

An important set of concepts related to intrusion detection is *false positive* and *false negative*. *False positives* are events that the system detects as an intrusion but are in fact acceptable system events. *False negatives* are intrusions that the system fails to recognize. It is important that both of these are kept to a minimum; too many *false positives* leads to alarms being ignored, too many *false negatives* and the IDS isn't doing anybody any good.

Both misuse detection and anomaly detection have their advantages and disadvantages. One of the biggest advantages of anomaly detection is its potential to detect novel or previously unknown attacks. Another appealing advantage is that once it is installed and set up anomaly detection requires little additional administrative maintenance. However, anomaly detection tends to have a very high *false positive* rate, often requires extensive training, and can be computationally expensive [45,46]. Misuse detection, one the other hand, is very efficient at detecting attacks without generating lots of *false positives*, and can be more easily used by system managers with less security

expertise. Unfortunately misuse detection cannot detect unknown attacks, meaning it has a potentially high *false negative* rate. Additionally, because new attacks are constantly being release, the signature database must constantly be updated, requiring ongoing administrative maintenance. Failure to keep signature databases up to date severely limits the capabilities of misuse detection [45,46].

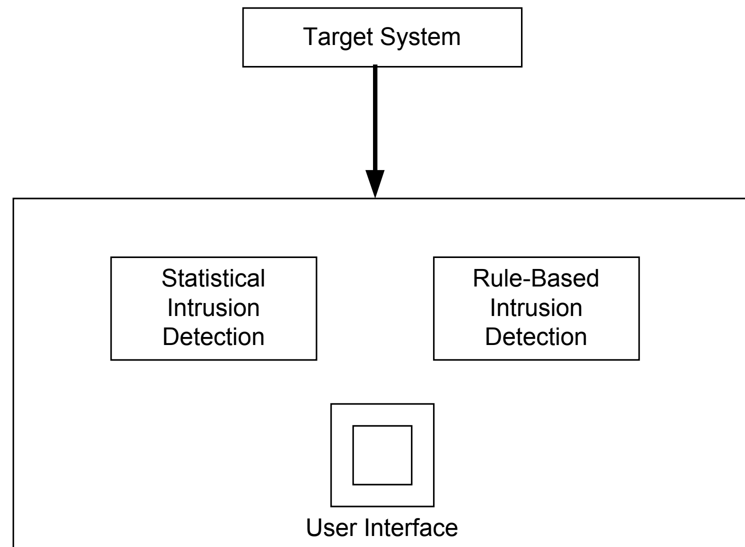

Figure 2.1. The IDES Model [28]

Early systems were all host-based systems (HIDS), meaning that they operated on only one computer, usually referred to as the target. This reflects the fact that most computers at that time were large mainframes with terminal connections. As PCs reached their glory in the late 80’s and 90’s, the paradigm of the single large mainframe gave way to networks with a variety of desktops and servers comprising LANs and WANs. The proliferation of the Internet and networking technology reinforced this trend.

In response to this changing environment, came the idea of a NIDS [10,45], or network intrusion detection system. Instead of examining events on a specific host, these

systems would try to identify intrusions by monitoring network traffic. Basically NIDS use the same approach as HIDS, misuse and anomaly detection, but the source of information on which they operate, called a sensor, is different. Essentially NIDS examine the communications between computers on a network, as opposed to system level audit trails. As would be expected *false positives* and *false negatives* continued to be a primary concern [45].

The main advantage of NIDS is that a few well-placed sensors are capable of monitoring a large network. In addition, their deployment usually has little effect on the rest of the network, as far as performance is concerned. However, they do not have access to system level audit logs as do HIDS; and more importantly, for busy networks they may fail to process all packets. Another problem they face is the fact that they cannot analyze encrypted packets. While HIDS do not have these weaknesses, in large networks, with many hosts, a HIDS on every host can be extremely hard to manage [46].

Combining NIDS and HIDS was investigated through the development of distributed systems and hybrid systems [10,20,30]. Such approaches centralize the analysis component, with host-based sensors and network based sensors feeding into a single analysis engine. While maximizing the advantages of HIDS and NIDS, it creates a single point of failure, which itself might become the target of an attack. Attempts to address this weakness have included investigations into mobile security agents [25] and artificial immune system models [12]. These approaches de-centralize control and operation of distributed intrusion detection systems, but their reliance on some type of anomaly detection approach means they are still plagued by high false positive rate.

Largely for performance reasons commercial IDS today are primarily based on misuse detection techniques [45,46]. Snort [55], one of the most popular IDS used today, is an excellent example. Snort is a network intrusion detection system that matches packets collected from the network against a set of rules or signatures. These signatures are developed from known attacks, and must constantly be updated. One way to improve the effectiveness of IDS like Snort is to include some type of anomaly detection [45].

2.2 Honeypots

Recently, honeypots have been receiving attention from security professionals looking for new tools to help them in the fight against the exponentially growing number of threats. The basic idea of a honeypot is to observe a system that you allow to be attacked, figure 2.2. Lance Spitzner defines a honeypot as follows: "A honeypot is a security resource whose value lies in being probed, attacked, or compromised" [1]. Since honeypots have no production value, no resource or person should be communicating with them, and therefore any activity arriving at a honeypot is likely to be a probe, scan, or attack. Their value comes from their potential ability to capture scans, probes, attacks, and other malicious activity.

There are three types of honeypots: low interaction, medium interaction, and high interaction [1]. In order to collect information a honeypot must interact with the attacker, and the level of interaction refers to the degree of interaction the honeypot has with a potential attacker. A low interaction honeypot provides minimal service, like an open port. A medium interaction honeypot simulates basic interactions like asking for a login and password, but providing no actual service to log into. High interaction honeypots

offer a fully functioning service or operating system, which can potentially be compromised.

There are two important categories of honeypots, production honeypots and research honeypots [1]. Production honeypots are honeypots that are used to protect an organization or an organization's operational network. Research honeypots, on the other hand, are used for research and intelligence gathering, and are not part of any commercial security mechanism. Production honeypots tend to be low-interaction and research honeypots tend to be high-interaction. Finally honeynets [53] are elaborate networks of multiple high-interaction honeypots and sophisticated monitoring software, and they are used almost exclusively for research.

The first documentation relating to honeypots occurred in the early 1990's in the works of Clifford Stoll, and Bill Cheswick. [33,34]. Neither describe their systems or techniques as "honeypots" but the central ideas of honeypots can be clearly seen in their work. These publications, while not technical, contributed significantly to a growing interest in honeypots and the development of numerous solutions. The first available honeypot solution, Deception Toolkit or DK, was released in 1997 [1]. DK is a collection of PERL scripts and C code that emulate various Unix vulnerabilities and then log the behavior and actions of an attack or attacker. CyberCop Sting, in 1998, was the first commercial honeypot, and introduced for the first time, virtual systems bound to a single host.

The idea of virtual honeypots made honeypot technology more affordable and available to a wider audience and encouraged development of better implementations. One of these is Honeyd, a robust open source honeypot solution developed by Niels

Provos [6,56]. Honeyd has the ability to emulate different operating systems at the IP layer and run scripts attached to specific ports using *stdin* and *stdout*. Instead of being limited to one IP address, Honeyd has the ability to bind to multiple IP address, making it able to create virtual honeypots that emulate multiple systems with different operating systems and applications and map them to unused IP addresses. It is a very powerful honeypot solution [50] version 0.8 was released in 2003, and included significant improvements.

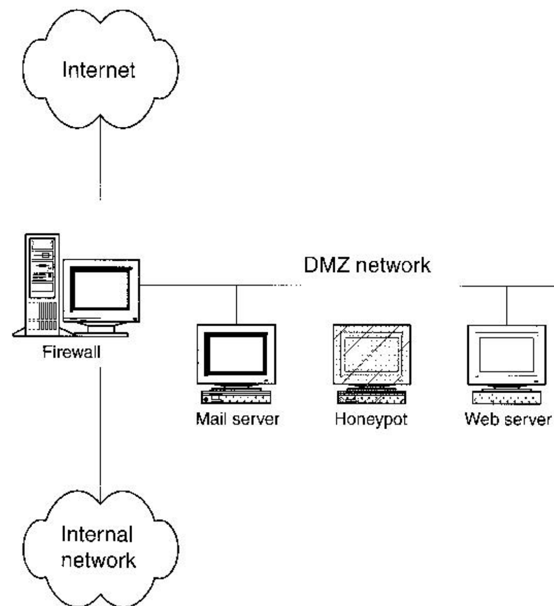

Figure 2.2. Honeypot deployment on a DMZ [1]

Initially, honeynets were just a network containing one or more honeypots [53], the next generation of honeynets, known as Gen II honeypots by the Honeynet project, were adapted from captured rootkits [52]. These honeypots are kernel modules, and provide superior clandestine monitoring capabilities for high-interaction honeypots.

Sebek is the most well known Gen II honeypot, and is a Linux kernel module. Sebek hooks all sys_read calls and covertly transports them to a remote monitor. This allows the host based monitoring of a compromised system without the attackers awareness.

With the success of early honeypots like Bof and Specter, more investigation into their potential uses in security has been done. Their potential use in prevention lies in distracting hackers or wasting their valuable time and resources attacking honeypots instead of production systems. The Labrea Tar pit honeypot is designed to prevent attacks by maintaining an, artificially slow, open connection to non-existent services [1]. Honeypots have also been explored as a means of detecting and preventing a DDoS attack and capturing forensic data [26,41].

Honeypots have even greater potential when it comes to detection. Levine et al [16] deployed a honeynet at Georgia tech that they used to detect exploited systems across their enterprise network. They successfully identified a system within the network that they suspected had been infected by a worm. In addition to identifying the infected system, they were able to provide the IT department with enough data to develop a signature for the previously unknown exploit [16]. They also identified an account whose password had been compromised by analyzing traffic to a backdoor installed on one of the honeypots from a system in the production part of the network.

Honeypots have also been shown to be effective against Internet worms. Laurent Oudot [7] demonstrated how MSBlast could be detected and captured using Honeyd and some simple scripts. He also showed how worm propagation can be slowed using Honeyd to attract the worms attention and then respond very slowly to its requests. Using scripts, Oudot demonstrated how a honeypot could even launch a counter attack

against a worm outbreak, either by isolating services or network segments, or by abusing the same vulnerability the worm used and then trying to kill the worm process.

Detection and identification of new threats is one of the most important areas in security. In July of 2002 the Honeynet project caught a previously unknown dtscpd exploit "in the wild" in one of their honeypots [1]. Identifying the new exploit and understanding it were carried out by researchers, but the fact that a honeypot was able to capture a new exploit points strongly towards their potential use in helping IDS identify new attacks. Zang et al [23] drew similar conclusions in their description of honeypots as a supplemental active defense system for network security. However, current investigations into integrating honeypots with IDS have primarily used honeypots to extend a detected intrusion's session [41,43] by rerouting an attacker (once identified) to a honeypot.

While not yet in wide use, according to NIST [46], honeypots are now considered to be part of the intrusion detection product line. According to NIST, as part of an IDS, honeypots serve as decoy systems that divert an attacker away from critical resources, collect information about attackers, and encourage the attacker to stay on the system long enough for administrators to respond.

Another advantage of honeypots is their ability to provide correlated high value data [1,15,39]. All of the data captured in a honeypot is relevant security data, as opposed to many other security related logs (such as firewall logs) where the majority of the data may pertain to normal network operations. In addition, honeypots can provide easier and more extensive monitoring of attacker's actions, and can potentially detect insider threats [5].

There are three significant disadvantages of using production honeypots. The most highly debated of these pertains to the unclear legal implication of using such devices [46,51]. Some argue that honeypots violate entrapment laws; while other argue they violate forth amendment rights. These concerns can be lessened through the use of banners, however the issues remain unclear and as yet untested in court. There is also the more important legal concern that your honeypot maybe compromised, and then used to attack systems other than your own.

Another potential disadvantage of production honeypots is that if they are detected they lose much of their value, and may even be used against you. This was not a problem early on, but as honeypots have grown in use, tools and techniques for detecting and/or fingerprinting honeypots have become available and openly discussed [3,37]. While making the use of honeypots more difficult, the fact that attackers are spending the effort to develop such tools and techniques is an indication that they consider honeypots a legitimate security measure.

A final disadvantage of honeypots is the fact that a high level of expertise is needed to configure and maintain these systems [3,46]. Initial configuration includes: determining the number and location of honeypots, what means will be used to attract traffic to the honeypot(s), determining the level of interaction, choosing an operating system (actual of virtual), deciding what services should be available or emulated, and how data is to be captured. On going maintenance includes keeping the honeypots secure and making sure that the honeypots continue to adequately mirror the production environment. Improper configuration can also lead to detection, making these issues doubly important.

Lance Spitzner has recently addressed some of the disadvantages associated with honeypots in his proposed concept of a dynamic honeypot [2]. A dynamic honeypot is a plug and play solution that automatically determines how many honeypots to deploy, how to deploy them and what they should look like. A dynamic honeypot could address, at least in part, both the problem of a lack of resources for configuration, and the problem of detection (which in some, potentially many, cases is directly related to configuration). By learning and monitoring networks, dynamic honeypots could reduce the amount of configuration and maintenance needed, and potentially decrease the chance that attackers would easily detect a honeypot.

Such a dynamic honeypot would learn about the network, and then deploy honeypots to appropriately blend in with the rest of the network. In addition it could also continue to monitor the network and update the honeypots based on changes it sees in the network. For example, if a network has all Windows systems, only Windows honeypots will be deployed. Later, if a Linux machine is added, miraculously Linux honeypots are deployed. The goal according to Lance Spitzner is "an appliance, a solution you simply plug into your network, it deploys the proper number and configuration of honeypots, and adapts to any changes in your networks." [2]

Some initial attempts have been made to implement a dynamic honeypot [49,57]. Both of these attempts have used Honeyd [[6] as the honeypot engine, and p0f [8,9] for passive network analysis. One problem these approaches have discovered is that passive network analysis (via p0f) is not 100% accurate. Both attempts met with success, but lacked a specific context in which the dynamic honeypot was to be used.

Not all responses to honeypot technology have been positive, Rong and Yang [40] argue that the potential misuses of honeypot technology by black hats present a threat to consumer trust in e-commerce (among other things) and therefore the use of honeypots should proceed with caution.

Initially honeypots were an esoteric security phenomenon, and largely deployed for intelligence gathering. Lance Spitzner and the Honeynet group have used them to gather a great deal of information about black hats [1]. Their effectiveness at intelligence gathering has lead to a desire to incorporate them into production systems [32,38]. While the use of honeypots in production security systems is still in its infancy, honeypots are already beginning to show up in commercial security software [19].

A recent paper by Kuwate, Sarj, and Masri, [63] explores the design and development of a dynamic honeypot. The design is based on Lance Spitzner and the Honeynet Organization's proposed concept of a dynamic honeypot [2]. They sight minimizing configuration and supervision as an important motivation for the development of a dynamic honeypot. While they do indicate that the dynamic honeypot can be used for intrusion detection, that functionality was not explored in the paper.

CHAPTER III

THEORY AND DESIGN

3.1 Honeypots And Intrusion Detection.

There are several aspects of honeypots that make them attractive for intrusion detection. Most notable is the fact that they innately implement anomaly detection, which has continued to be both desirable and illusive. Given that most systems generate lengthy logs everyday, the high value/high correlation aspects of the data generated by honeypots is also attractive. There are several ways honeypots can be used as part of production intrusion detection systems.

One approach is to use honeypots as a resource to which to divert malicious activity [43,46]. This is useful both in consuming attacker's resources and in potentially gaining further knowledge about the attacker [59] and possibly the attack. While this approach can be effective, it could be considered something like a jail to which offenders are sent after they are caught (and perhaps interrogated). When used this way it is up to some other systems to identify the malicious traffic, or intrusions. The drawback of this approach is that it does not take advantage of the native ability of honeypots to do anomaly detection.

Another approach is to use honeypots for intelligence gathering, deploying honeypots, then extracting from the honeypot data signatures to be used by the production intrusion detection system [59]. This approach does make use of some of the anomaly detection ability of honeypots, in a more "research" oriented approach.

However, no automated means exists of converting the raw honeypot data into usable signatures. Therefore, use of such an approach requires a great deal of man-hours from highly skilled personel. Since there is no necessary correlation between the honeypot and the production system, it is possible (or likely, depending on how the honeypot was deployed) that this approach would produce some or many irrelevant signatures.

In order for a honeypots to genuinely carryout intrusion detection, they must be placed appropriately within a production network (rather than being isolated from it). Once deployed, any interaction with the honeypot can be considered anomalous, and therefore intrusive. This approach uses the intrinsic anomaly detection ability of honeypots to implement intrusion detection. This approach might also be called behavior based, since it is based on the behavior of systems on the network as they interact with the honeypot.

The advantage of such an approach is the honeypot's ability to detect previously unknown attacks, and minimize false negatives. However, it is possible for an intrusion to never interact with a honeypot (good deployment should minimize this possibility). This is one of the drawbacks to honeypots and why identification of honeypots makes them virtually useless. Therefore the objective of a honeypot based intrusion detection system is not to replace existing IDS, but instead to compliment current IDS, like snort, by providing independent, anomaly based, intrusion detection. In addition a honeypot based intrusion detection system can provide additional data that can be used for incidence response. Precisely what that data will consist of depends on the level of interaction of the honeypot.

Recall that low interaction honeypots can generally only capture transaction level data, the source and destination of an attack, but that high-level of interaction honeypots can capture more detailed information ranging from the application layer session data (which might include commands and exploits) to the contents of files uploaded to a victim machine. Therefore, a behavior based or anomaly based IDS using a low interaction honeypot will at most identify the network address and port from which an intrusion occurred or is occurring. But a higher interaction honeypot will potential capture more detailed information.

As far as simple detection is concerned, a low-interaction honeypot is just as good as a high-interaction honeypot. A very simple port monitor listening on unused (undesignated) IP addresses can detect a connection attempt, indicating an intrusion of some sort. However, little else could be learned from such a honeypot based IDS. Since the objective is to provide detection and a resource for incidence response more information would be helpful, making a high-level interaction honeypots a more attractive choice.

All other things being equal, a high-level interaction honeypot is obviously the best choice for use with a behavior based IDS. However there are other elements that must be taken into account. All honeypots have some risk associated with them, but high-interaction honeypots have significantly more than low-interaction honeypots, and an intrusion detection system should ideally not increase the amount of risk associated with the network it is trying to protect. High interaction honeypots that are used in bait and switch techniques present little additional risk because they are isolated from the

production part of the network. A high interaction honeypot within the production part of the network could pose a serious security risk.

Another important issue to consider is configuration and maintenance. Honeypots require a great deal of effort to configure and maintain, with high interaction honeypots being the most difficult to configure and maintain. Usually a high interaction honeypot is an actual system (or virtual machine) to which an attacker is given full access. However, even low interaction honeypots take significant effort to deploy and maintain properly. If the overhead of configuring and maintaining the honeypots is too great then it will not be practical to deploy them as part of an intrusion detection system. This issue can be addressed by using a dynamic honeypot.

3.2 Dynamic Honeypots

Recall Lance Spitzner's description of a dynamic honeypot [2] as a plug and play solution, capable of determining how many honeypots to deploy, what they should look like, and deploying them. Furthermore, once deployed, the dynamic honeypot continues to monitor the network for changes, maintaining the honeypot deployments and keeping them up to date relative to changes in the network. There are a number of significant challenges to developing such a system; fortunately there already are existing tools that address some of these problems.

3.2.1 Passive Network Analysis.

The most significant challenge facing the dynamic honeypot is how it is learns about the network in which it has been placed. With out such knowledge a dynamic honeypot is not possible. There are several approaches that could be used here such as

actively probing the network, using something like Nmap, to determine what systems are up and what they are running. There are some disadvantages to this approach: you might miss something that is fire walled, you introduce more traffic onto the network, probing may cause a system to shutdown unexpectedly, and finally the result of the probe is a static picture of the network, meaning it will have to be preformed on a regular basis to keep the knowledge base up to date.

A better approach, advocated by Spitzner, would be to use passive OS fingerprinting. Passive OS fingerprinting is similar to probing. It maps and identifying systems on a network, but instead of sending out packets and examining the response, passive fingerprinting examines captured packets and compares them to a database of signatures. This approach is much less intrusive and can be carried out continuously to provide a real-time mapping of the network.

Passive network analysis operates by examining packets from actual or legitimate sessions instead of generating packets that are apart of its own session. The values of certain fields, from the TCP header for example, are then compared to known values for specific operating systems. These known values are called fingerprints. Based on the fingerprint database, the operating system type for the host that generated the packet can be established.

P0F is a free open source a passive OS fingerprinting tool written by Michal Zalewski, and can be used to carry out passive network analysis for the dynamic honeypot. It uses numerous different metrics for fingerprint identification that are supposed to give it a high degree of accuracy. There are actually three modes of operation: incoming connection fingerprinting, outgoing connection fingerprinting, and

outgoing connection refused fingerprinting, corresponding to the SYN, SYN/ACK, and RST parts of the TCP handshake. P0f operates on single packets, and generated the following output, shown in figure 3.1, when run on a test LAN.

```
<Sun Mar 28 22:47:19 2004> 192.168.1.101:6000 - Linux recent 2.4 (2)
  -> 192.168.1.100:44003 (distance 0, link: ethernet/modem)
<Sun Mar 28 22:51:37 2004> 192.168.1.23:5000 - Windows 2000 Professional
  -> 192.168.1.100:57615 (distance 0, link: ethernet/modem)
```

Figure 3.1. Sample p0f output.

In this example, the host 192.168.1.101 has been fingerprinted as “Linux recent 2.4 (2)”, and the host 192.168.1.23 has been fingerprinted as “Windows 2000.”

3.2.2 Virtual Honeypot Deployment

Another challenge for the dynamic honeypot is that to be truly effective, the dynamic honeypot will be deploying multiple honeypots. While individual machines could be used for each honeypot and deployed throughout the network, this is obviously impractical on many levels and could hardly be construed as plug and play. Instead, a much more desirable solution would be for a single appliance to deploy multiple virtual honeypots on a network’s unused IP addresses. The open source solution Honeyd [6] has exactly this ability.

Honeyd is a very powerful, very configurable honeypot solution. Configuration is done through a configuration file, specified in the command line. Honeyd allows for the definition of various “personalities” which are then bound to an IP address. The OS simulation for each personality is based on the NMAP finger print file, giving Honeyd the ability to emulate over 500 different operating systems. For each personality port actions are defined for TCP, UDP, and ICMP.

The response options for ports are open, block, reset. Block does not respond to a connection attempt, reset closes a connection attempt, and open creates a connection. When a port is open incoming data is allowed, but not captured by Honeyd. Honeyd sends out no application data and the connection remains open till closed by the source. In addition to these options, an executable can be attached to a specific protocol and port. This creates an open port with incoming data forwarded to the executable and output from the executable forwarded to the source through using I/O redirection and standard I/O. In the example listed in figure 3.2, test.sh is a script that just echoes back to the sender whatever it receives.

```
#Sample Configuration
create windows
set personality windows "Windows NT SP4 – SP5"
set windows default tcp action block
set windows  default udp action reset
set windows  default icmp action block
set windows tcp port 137 action open
add windows tcp port 1006 "scripts/test.sh"
bind 192.168.1.110 windows
```

Figure 3.2. Sample Honeyd configuration.

You can specify to Honeyd on the command line an IP range on which to operate. If none is specified it will attempt to respond to any packet it sees. Individual personalities are bound to specific IP addresses, as seen in the above example: *bind 192.168.1.110 windows*. There must be a default personality defined as well, which is used for connection attempts to IP addresses for which no specific personality has been bound.

Unless traffic is specifically routed to Honeyd, arpd must be used to attract traffic to the honeypot. Arpd responds to unused IP addresses (the range of which is given in the command line) with the system's MAC address, which means any traffic not destined for an actual host will end up at the system running arpd and therefore be seen by

Honeyd. For example arpd 192.168.1.0/24 will respond to the arp protocol "who-has" for any address between 192.168.1.1-192.168.1.255 for which no other system responds.

Honeyd also has some dynamic capabilities that can be accessed through the configuration file. This gives Honeyd the ability to create different virtual honeypots based on the source's IP address, operating system, or time of day. These dynamic honeypots are then bound to an IP address, and respond with different templates based on the condition given in the configuration.

3.3 A Dynamic Honeypot Design

The initial dynamic honeypot design takes into account the functionality of p0f and Honeyd. Recall that the goal of the dynamic honeypot it to determine how many honeypots to deploy, where to deploy them and what they should look like to blend in with the surrounding environment. Figure 3.3 shows the basic elements needed for a dynamic honeypot.

The passive network analysis module carries out passive network analysis, specifically OS fingerprinting. It will sniff packets directly off the wire and place the results into the dynamic honeypot database, the collection of information about the network that represents what the dynamic honeypot has learned about the network. It will use p0f to passively fingerprint systems based on packets sniffed from the network. This must include a set of IP addresses and the operating systems associated with them, as well as a list of open ports associated with each host. This information will be stored in the dynamic honeypot database.

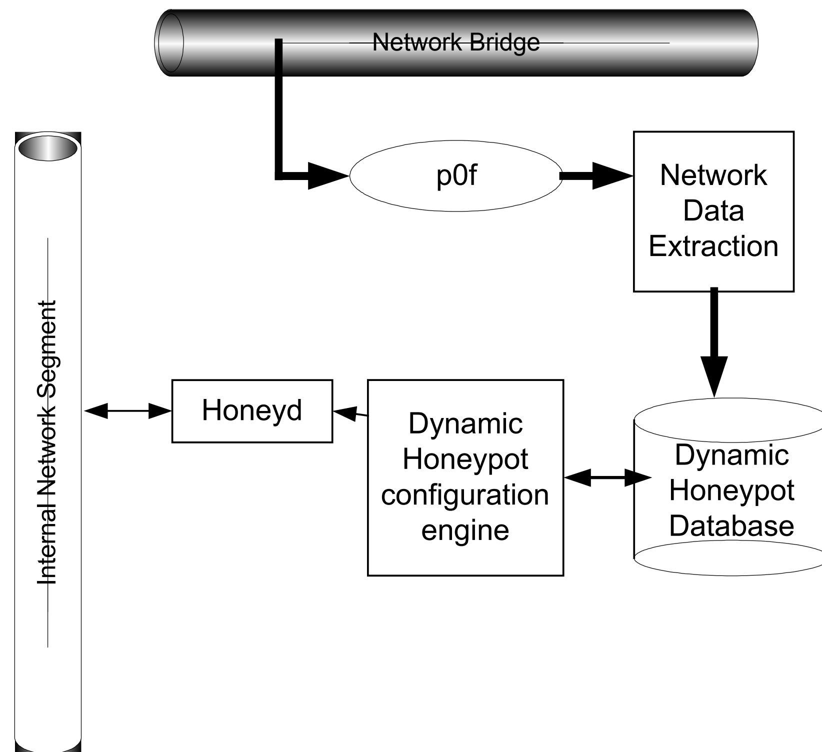

Figure 3.3. Model for a dynamic honeypot design.

The honeypot deployment module will deploy the virtual honeypots, based on some given configuration. It will be able to simulate a variety of different OS and deploy honeypots on multiple IP addresses from a single network connection. This module will consist almost entirely of Honeyd, with the virtual honeypot definitions taking the form of Honeyd style configuration templates and bindings

The configuration module will use the data from the network mapping storage to create the virtual honeypot definitions. A virtual honeypot definition will consist of an IP address, an OS, and a set of open ports. Configuration will be achieved by grouping the existing hosts together based on their OS type and the distance between their IP addresses. Each group of hosts will then have an associated honeypot that has the same (or similar) OS as the group members. The open ports on that honeypot will include all

open ports of each host in the group. Finally the IP address of the honeypot must be similar to those of the group, but cannot be an IP address already in use. In order to maintain consistency with the current network, the honeypot definitions will be updated at regular intervals.

3.4 Intrusion Detection Using A Dynamic Honeypot

Having designed a dynamic honeypot, we now return to using it to carry out intrusion detection. Once deployed, the network interface on which the dynamic honeypot is listening will be sniffed. Ideally any and all traffic seen will constitute an intrusion. In reality it is likely that even if there are no intrusions (insider threat and internal compromised host count as an intrusion), there may be traffic to the honeypot. Such benign traffic would still reflect some error in the network somewhere, and reporting it should help administrators identify the source of the problem. It is therefore reasonable to capture all the traffic using some kind of packet sniffer, and for all traffic to merit some kind of alarm, if not a red alert. In addition it might be useful to look for known intrusion signatures since these may have eluded the defenses elsewhere.

We have yet to resolve the issue of what level of interaction the honeypot is to be. Honeyd is generally a low to medium interaction honeypot, capable of port monitoring or passing payload data to a script or executable. However Honeyd can also use the “proxy” action to pass a connection to another system entirely. By using the proxy action for a port it is possible for Honeyd to achieve a high-level of interaction, by proxying to an actual system. While that system could be one of various high-interaction honeypots (isolated from the rest of the network), it could also be an actual host in the production

network. The issue that must first be addressed is how this might impact the security of the existing network.

Using the dynamic template ability it is possible to have the honeypots react differently depending on whether the connection host is apart of the internal or external network. This makes it possible to use a higher level of interaction for internal hosts, or a lower level of interaction for external hosts. In this way the dynamic honeypot can use hosts on the internal network as high-interaction proxies when connecting systems are located with in the internal network, and use just simple open ports or scripts to emulate services when the connection host is apart of the external network.

This does not add unreasonable additional risk even though the honeypot is redirecting traffic to a production system. Since the source is coming from the internal network, whichever host the honeypot proxies is also directly reachable from the source. Therefore the honeypot does not introduce additional risk by using proxies for internal sources only. Forwarding external hosts to internal host would represent a serious increase in the risk for the internal network, and must be avoided.

This approach may decrease the amount of time it takes for an internally compromised machine to find additional hosts to compromise, but in so doing it should also significantly decrease the period of time before such an event is noticed. In addition, the honeypot will capture the entire session, and isolate it from any production host. This should aid in incidence response and potentially provide good forensic evidence that can be easily preserved without disturbing any production hosts. In addition efforts to secure and maintain the production systems at the same time secures and maintains the high-interaction honeypots.

Using low interaction honeypots for external hosts also makes good sense. They are the most secure kind of honeypot, and therefore the least likely to be compromised. If the source of an intrusion is from the external network, then the primary goal of the honeypot based IDS is to identify the intrusion, which a low-interaction honeypot can do adequately. Scripts or executables could be used to emulate various services, and the dynamic honeypot will facilitate them being associated with the appropriate ports. However, they are not absolutely necessary and do pose certain risks. Very simple emulators lack realism, and are probably easily fingerprinted, even by automated attacks. More complex emulators, while much more realistic, have greater potential to be compromised, and introduce unwanted additional risk. Using just the port listening capabilities of Honeyd will capture any and all payloads pushed to the honeypot. While this type of connection lacks realism, it hardly gives itself away sense it does not respond at all.

It is also makes sense to treat the internal and external networks differently from the perspective of intrusion detection. Connection from an external host is a good indication of a mis-configured firewall or border security measure. Connection from an internal host is likely indication of an insider threat or a compromised host on the internal network.

To use the dynamic honeypot to carry out intrusion detection once the dynamic honeypot has been deployed, the interface on which the honeypot is listening must be sniffed. This traffic is the high value data that will be used for intrusion detection. Since all traffic to a honeypot is anomalous the need to identify anomalous events vs. normal traffic is eliminated. Therefore a simple reporting mechanism that presents the raw

traffic data would be sufficient for intrusion detection. However, additional processing of the traffic data is also possible.

Additional processing could take the form of applying a set of rules to the honeypot data. These rules would define an intrusion in terms of honeypot traffic. An alarm or alert that an intrusion has occurred would be generated when the conditions of one of the rules was met. For example: if the majority of the traffic is coming from a single host, or if an individual host is interacting with multiple honeypots on the same port. A potential set of rules was developed, and is listed in appendix D. A layout of the dynamic honeypot intrusion detection system is listed in figure 3.4.

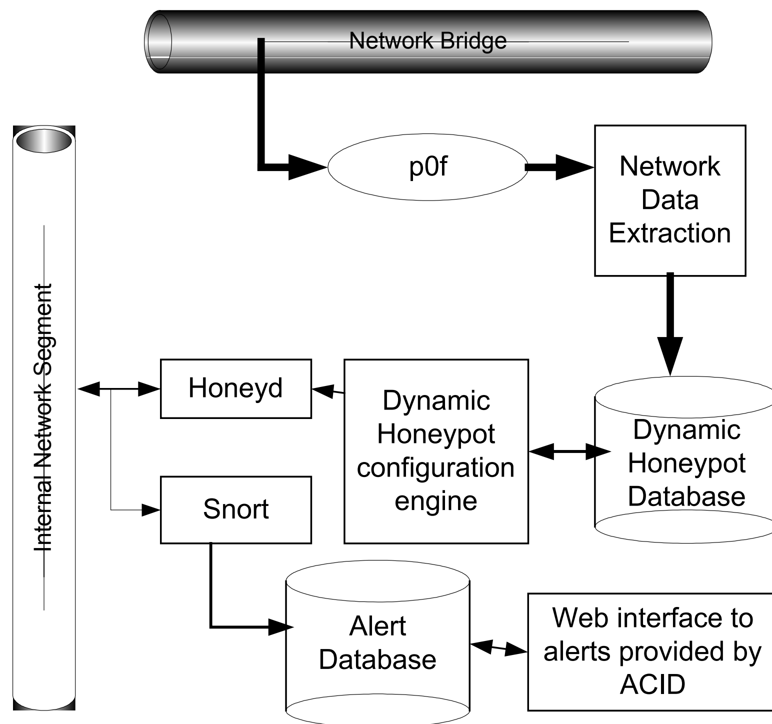

Figure 3.4. Design for dynamic honeypot intrusion detection.

The network interface on which the dynamic honeypot listens is the internal interface, the same interface to which production systems are physically connected. The

sniffer captures all packets bound for the honeypots and logs them. Snort is excellent for packet capture and has become the sniffer of choice for many honeypot deployments. Once packets are captured they are passed to the log/alert/alarm mechanisms. All packets will be logged for future reference, incidence response, and forensics. The alert and alarm mechanisms generate the output of the intrusion detection system. Since it has been established that all traffic to the honeypots is anomalous and potentially indicative of an intrusion, one of these mechanisms will output all traffic to the honeypots. Another mechanism will perform some additional analysis of the traffic by apply some rules to the honeypot traffic.

CHAPTER IV

IMPLEMENTATION

An intrusion detection system based on a dynamic honeypot was implemented for a small LAN. This first required the implementation of a dynamic honeypot. Once deployed, the dynamic honeypot was used to carryout anomaly based intrusion detection by monitoring all traffic coming into the dynamic honeypot[s]. The dynamic honeypot was developed using C++ on Red Hat's Fedora core 2 Linux distribution, kernel 2.6. The LAN consisted of a Belkin router, a Linksys router, three Linux systems, a Windows 2000 system, and a Windows XP system connected to the Internet through a high-speed cable modem.

4.1 Dynamic Honeypot Implementation

There are three key elements that must be addressed by the implementation: gathering information about the network, generating honeypot definitions from gathered information, and finally deploying the honeypots. This dynamic honeypot implementation was an integration of some existing solutions with the implementation of a solution that generates honeypot definition from network information. Existing tools, specifically p0f and Honeyd, were used rather than re-implementing these partial solutions. Since there did not exist a publicly available tool for generating honeypot

definitions from network information, this part of the dynamic honeypot was implemented from scratch.

4.1.1 Gathering Network Information

Recall that it was determined that p0f would be used to carryout passive network analysis. Since passive network analysis will be an ongoing process, storing the results in some type of database will facilitate sharing that information with other parts of the dynamic honeypot. MySQL was chosen as the database, and several tables were created to store the various information. One table, called *host*, holds the IP address, operating system fingerprint, the number times this host has been fingerprinted as this operating system, and the time the last fingerprint was made. Another table, called *ports*, has fields IP address and port number that contain all the open ports for each host, associated by IP address. The configuration of these tables is given in figure 4.1; the field service and app of the ports table are not used in this implementation.

```
mysql> desc host;
+--------+------------------+------+-----+---------+-------+
| Field  | Type             | Null | Key | Default | Extra |
+--------+------------------+------+-----+---------+-------+
| ipaddr | int(10) unsigned |      | PRI | 0       |       |
| os     | varchar(255)     |      | PRI |         |       |
| count  | int(11)          | YES  |     | NULL    |       |
| last   | datetime         | YES  |     | NULL    |       |
+--------+------------------+------+-----+---------+-------+
4 rows in set (0.00 sec)

mysql> desc ports;
+---------+------------------+------+-----+---------+-------+
| Field   | Type             | Null | Key | Default | Extra |
+---------+------------------+------+-----+---------+-------+
| ipaddr  | int(10) unsigned |      | PRI | 0       |       |
| port    | smallint(6)      |      | PRI | 0       |       |
| service | text             | YES  |     | NULL    |       |
| app     | text             | YES  |     | NULL    |       |
+---------+------------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Figure 4.1. Host and port table definitions.

Since p0f lacks a native MySQL interface, the output was piped into a secondary program that extracted the relevant data and inserted it into the database. Both the SYN and SYN/ACK modes of p0f were used. This was necessary since a server might never initiate a connection, and would subsequently not be fingerprinted by the SYN mode of p0f. When p0f cannot fingerprint a particular packet, it labels its operating system as 'UNKNOWN' followed by the associated fingerprinting values for that packet. It is possible to suppress this feature using the –U option, which was used in this implementation.

Unfortunately p0f is not 100% accurate or deterministic. It is possible that one packet generated by a host will be fingerprinted as Windows NT, while another might be "Windows XP" (or potentially Linux 2.2, but this is much less likely). P0f also appends additional data separated by a comma or inside parenthesis, for example '(2)' for a second fingerprint for the same operating system. To simplify matters slightly once a comma, parenthesis, or bracket was encountered in the operating system string, the remainder of the string (including the symbol) was truncated. This effectively produced OS labels like Windows XP, Linux 2.1-2.4, where the main operating system type, i.e. Windows, Linux, AIX, BSD, etc. was listed first. This approach allowed a single host, or IP address, to have more than one associated operating system fingerprint.

Recall that a dynamic honeypot will need to decide what port should be open on each honeypot. Therefore the passive network analysis module needs to generate a set of open ports for each host. The output of p0f does included port numbers; however these could just as likely be ephemeral ports or open ports. It makes no sense to have open ephemeral ports, and including them in a honeypot definition could potentially give away

the honeypot. Unfortunately there is not enough information in the p0f output to distinguish between a server port and an ephemeral or client port.

For TCP packets, the SYN/ACK flag combination indicates that the source host is listening on that TCP port. Thus tcpdump can be used to find server ports, at least for TCP. This was achieved by selecting only those packets where byte 13 in the TCP header has value 18 (10010, 0x12). This can be achieved using tcpdump with the following options:

```
tcpdump -nn 'TCP[13] == 18'
```

Determining listening ports for UDP and ICMP will be more difficult, so for this project we will focus on TCP only (this is hardly unreasonable since TCP is by far the most commonly used protocol). Once the packets have been selected, then extracting the source address and port is simple, and the result can be inserted into the honeypot database.

The dynamic honeypot will want to fingerprint as many possible systems on the local network as possible. A single interface on the local network, even running in promiscuous mode, is not sufficient to capture all the traffic even on small network, and for larger networks would be even more inadequate. Some kind of interface must be used that makes sure that all network traffic can be seen by p0f. Two possibilities for improving the scope of the traffic available to p0f are using a spanning port or a bridge. A spanning port receives all the traffic flowing through a router or a switch, allowing p0f to see traffic to all systems connect to the switch or router. A bridge connects to network segments, and all traffic between the two segments must cross the bridge, allowing p0f to see traffic to all systems connected through the bridge.

The bridge was chosen for several reasons: It doesn't rely on someone to correctly identify (and configure) the spanning port on a network switch. It allows the dynamic honeypot to become a device that you physically place in front of the network on which you want it to operate. It operates at the link layer, making it a stealth device, meaning that not only can it not be seen; neither is it necessary to make any changes to routers or host [60]. Finally, it could eventually carryout intrusion prevention since it has control over what flows in and out of the local network. Figure 4.2 shows the construction of the LAN. The Belkin router and its systems are the internal or productions network on which the dynamic honeypot will be deployed. The Windows XP system is a part of the external network, and interacts with the dynamic honeypot and the internal network the same as a remote host on the Internet.

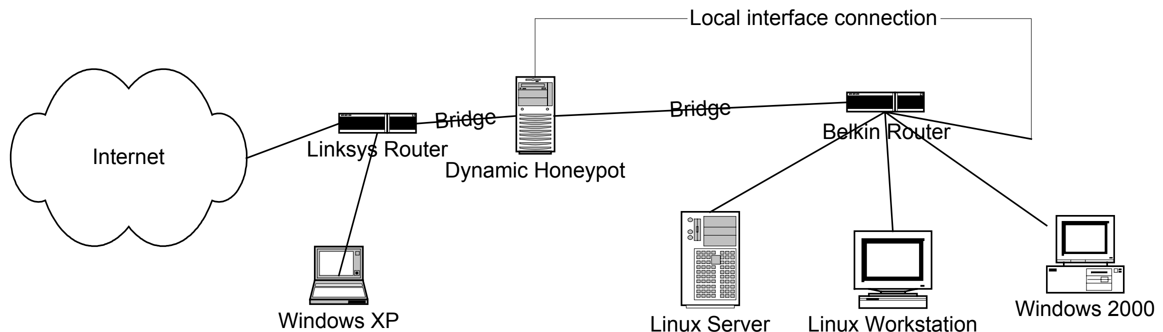

Figure 4.2. Configuration of the development and test network.

Using a bridge will require two additional interfaces. The bridge will use two interfaces, and the third interface will be connected to the internal network as if it were any other normal host. The bridge will be connected between the top-level router/switch for the internal network, and the perimeter connection from the ISP or a firewall. The

honeypots will be deployed through the interface that is connected to the internal network, as seen in figure 4.3.

Another issue that must be addressed at some point is differentiating between the local network, on which we want to deploy honeypots, and the external network. While it is clear in figure 4.3 that network interface 1 is the WAN connection and interface 2 is the LAN connection, this is not inherently obvious from the perspective of the bridge. Interface 1 and 2 could be switched and have no effect on how the bridge performs or how it views packets. The dynamic honeypot must have some means of deciding which IP address (seen on the bridge) are apart of the local network, and which are not.

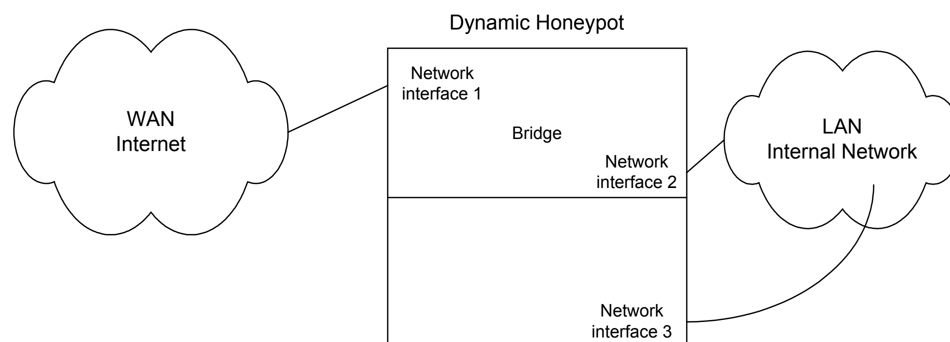

Figure 4.3. The use of interfaces in the dynamic honeypot.

This does not necessarily have to be addressed during passive network analysis; it could be addressed when the honeypots are being configured. In this case the dynamic honeypot would be trying to fingerprint every host on the internal and external network. While this might provide some useful incidence response information, at this point it is superfluous, and undesirable. Instead, it would be better during passive network analysis to filter out everything but systems on the local network. It would appear that this should be very easy to do because of the bridge.

Unfortunately there is really no simple method of determining direction of flow on the bridge. However, if the mac address of the internal switch or router were known, then traffic could be filtered at the link layer. This was achieved by filtering traffic on the bridge based on the address and mask of the local network interface (network interface 3) and extracting the source's mac address. Assuming that the bridge connects to some type of router, filtering at the link layer for packets whose link layer source is this mac address will filter out all hosts not apart of the local network. For this implementation a single router, switch, or hub will be assumed. (It is possible that there might be additional devices all directly connected to the bridge, which would simply mean filtering for all these devices' mac addresses as well.)

Since it may happen that some hosts are not fingerprinted at all, and it is important that honeypots are not assigned an IP address already in use, it would be advantageous to have a separate list of all hosts on the internal network, even if they have not been fingerprinted. This was achieved by adding another table to the database, called *flock*, which consists of two fields, an IP address, and a timestamp. Values are inserted into the table by extracting the relevant IP address from a continuous tcpdump stream that implements the appropriate link layer filter. The processes that insert the operating system data and port data can first check to make sure that the host is in the internal network using this list. In addition, honeypots can consult this table to make sure they are not binding to an IP address already in use.

Tcpdump and p0f are launched in daemon mode and their output is piped into an executable that extracts the relative data and inserts it into the database. This processes continues indefinitely, and the timestamps in the *flock* and *host* tables are used by the

dynamic honeypot to select only up to date information. This process results in the three tables *flock*, *host*, and *ports* being filled. A sample result is shown in figure 4.4. This part of the dynamic honeypot will run independently for a period of time, so that the database is adequately populated.

```
mysql> select inet_ntoa(ipaddr) as ipaddr,os,count,last from host;
+--------------+----------------------+-------+---------------------+
| ipaddr       | os                   | count | last                |
+--------------+----------------------+-------+---------------------+
| 192.168.2.16 | Linux 2.4/2.6        |   118 | 2004-09-07 12:33:09 |
| 192.168.2.22 | Windows 2000 SP2+    |    94 | 2004-09-07 11:06:24 |
| 192.168.2.62 | Linux 2.4/2.6        |    27 | 2004-09-07 11:51:25 |
| 192.168.2.40 | Linux 2.4/2.6        |    18 | 2004-09-07 12:39:43 |
+--------------+----------------------+-------+---------------------+
4 rows in set (0.03 sec)

mysql> select inet_ntoa(ipaddr) as ipaddr,port from ports;
+--------------+------+
| ipaddr       | port |
+--------------+------+
| 192.168.2.16 |   22 |
| 192.168.2.16 |  443 |
| 192.168.2.16 | 1241 |
| 192.168.2.16 | 3306 |
| 192.168.2.22 |  135 |
| 192.168.2.22 |  139 |
| 192.168.2.22 |  445 |
| 192.168.2.22 | 1025 |
| 192.168.2.40 |   22 |
| 192.168.2.40 |   80 |
| 192.168.2.40 |  111 |
| 192.168.2.40 |  443 |
| 192.168.2.62 |   22 |
| 192.168.2.62 |  111 |
| 192.168.2.62 | 3306 |
| 192.168.2.62 | 6000 |
+--------------+------+
16 rows in set (0.02 sec)

mysql> select inet_ntoa(ipaddr) as ipaddr,time from flock;
+--------------+--------------+
| ipaddr       | time         |
+--------------+--------------+
| 192.168.2.16 | 040907130248 |
| 192.168.1.98 | 040908100631 |
| 192.168.2.22 | 040907130248 |
| 192.168.2.62 | 040907120331 |
| 192.168.2.40 | 040907130248 |
| 192.168.2.1  | 040907130248 |
+--------------+--------------+
```

Figure 4.4. Contents of the host, ports, and flock tables as a result of passive network analysis

4.1.2 Generating Honeypot Definitions.

Once the host table and port tables have been populated the processes of configuring the honeypots begins. Based on the values in host and ports, the dynamic honeypot must make the following determinations:

- How many honeypots to deploy?
- What the OS personality each honeypot should have?
- What the IP address of each honeypot will be?
- What TCP ports should be open?

To make these determinations a simple rule based approach was taken that addresses each problem incrementally. A very simple set of rules was developed and is listed in appendix C. These rules were then implemented in a C++ program that interfaces with the dynamic honeypot database.

The information needed to define each honeypot will be stored in the dynamic honeypot database. Three tables, *honeypots*, *honeyhosts*, and *honeyports* will be used. The table *honeypots* will have the following fields: honeypot id, IP address and operating system. The honeypot id is a unique integer identifier for each honeypot, the IP address and operating system will be the operating system personality for the honeypot, and the IP address will be the IP address to which the honeypot is bound. The *honeyports* table will have fields: honeypot id, port, and proxy. A description of each table is given in figure 4.5.

To establish the number of honeypots to deploy, all hosts are partitioned into groups, and a honeypot is created for each group. Two characteristics will be used in determining group membership for each actual host: 1) the distance between the IP addresses of the hosts in the group and 2) the similarity between the operating systems as determined by p0f during passive network analysis. Each host is evaluated individually

to determine the group to which it will belong. If there does not exist a group to which the host belongs, a new group is created and the host is placed in that group.

Initially there are no groups, so the first host is placed in a new group. Then each successive host is analyzed to see if it is a member of an already existing group using a membership function that returns the group to which that host belongs, or –1 if it does not belong to any group. The membership function examines the IP address distance between the current host and hosts in other groups and the operating system type of the current host and the operating systems type of hosts in other groups.

```
mysql> desc honeypots;
+--------+------------------+------+-----+---------+-------+
| Field  | Type             | Null | Key | Default | Extra |
+--------+------------------+------+-----+---------+-------+
| hpid   | int(11)          |      | PRI | 0       |       |
| ipaddr | int(10) unsigned | YES  |     | NULL    |       |
| os     | varchar(255)     | YES  |     | NULL    |       |
+--------+------------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> desc honeyhosts;
+--------+------------------+------+-----+---------+-------+
| Field  | Type             | Null | Key | Default | Extra |
+--------+------------------+------+-----+---------+-------+
| hpid   | int(11)          |      | PRI | 0       |       |
| ipaddr | int(10) unsigned |      | PRI | 0       |       |
+--------+------------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> desc honeyports;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| hpid   | int(11)     |      | PRI | 0       |       |
| port   | smallint(6) |      | PRI | 0       |       |
| proxy  | varchar(40) | YES  |     | NULL    |       |
| script | varchar(80) | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
4 rows in set (0.01 sec)

mysql> desc scripts;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| port   | smallint(6)  |      | PRI | 0       |       |
| script | varchar(255) |      | PRI |         |       |
| os     | varchar(40)  | YES  |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Figure 4.5. Honeypots, honeyports, honeyhosts, and scripts table definitions.

In order for two hosts to be members of the same group the distance between their IP addresses must be less than a predetermined threshold. This distance threshold is a percentage of the average distance between the actual hosts, which is determined by dividing the total address space by the number of hosts. The result of such a function forces more densely populated networks to be more discriminating in their group selection and vice versa. The size of the address space is determined by finding the highest order bit that differs among the hosts, the address space then being 2 to that power.

This approach actually generates an address space that is possibly much smaller than the actual address space, but yields a more useful average distance. For example, given the host addresses: 192.168.1.16, 192.168.1.22, and 192.168.1.40 on a 192.168.1.0/24 subnet, the actual address space is 255 giving an average distance of 85, which is much greater than the distance between any of the hosts. However, only 6 bits are used to distinguish the hosts, yielding an address space of 64 and an average distance of approximately 21, which is closer to the observed distances. Were there to be an additional host at 192.168.1.250 this would result in an average distance of 64, making it more likely the initial hosts would be grouped together. The threshold distance used for membership is a percentage of the average distance, initially set at 75%. In theory, a higher percentage should produce fewer groups and subsequently fewer honeypots, and vice versa.

The comparison between two operating system strings is based on the general operating system class, such as Linux, or Windows. This will always be the initial substring that contains no white space. Since all the values are stored in the MySQL

database, the LIKE function can be used in conjunction with a select statement for comparison. The LIKE function is not case sensitive; in addition the '%' is appended to the string so that it will match to any other operating systems type that begins with that general class. For example LIKE 'Linux%' will match with any other operating system type that begins with Linux, e.g.: Linux 2.4, or Linux 1.9.

While there is no reason that a host cannot be a member of more that one group, it was decided that each host should only belong to one group. Therefore, if a host meets the membership requirements for more than one existing group, the distance threshold is divided by 2 and the membership function is called again. This processes is repeated recursively till the host belongs to only one or zero groups. In the case of zero, the host will be considered a member of one of the groups from the previous match. At the end of this process, each host on the network will be associated with some group, called the honeypot group.

Next, each honeypot group needs to be assigned an operating system. Honeyd uses the Nmap fingerprint database to simulate different operating system TCP/IP implementations, so the operating system needs to be selected from possible Nmap operating system fingerprints. All possible Nmap fingerprint names are placed in a table called *osfinger*. For each group, the operating system with the highest count is selected from all the hosts in that group. Then the operating system type is extracted and all possible Nmap fingerprints are selected that match the operating system type. Further matching continues by successively appending an additional character from the original operating system fingerprint string until no matches are found or the end of the string is reached, and the result of the previous comparison is used as the final match. For

example: LIKE ‘Linux 1%’, LIKE ‘Linux 1.%’, and LIKE ‘Linux 1.2%’ would be generated from the operating system string ‘Linux 1.2’. This process should result in a limited set of potential candidates from which the personality is chosen at random.

Next, an unused IP address must be selected for each group, this will be the IP address of the honeypot associated with that group. It would be desirable for the IP address to closely resemble the IP addresses of the hosts in the group. Therefore the rule for selecting an IP address begins by identifying only those bits that are not the same for each host. For groups that only have one host, the zero bits of the subnet mask from the local interface will be used. A candidate IP address is generated by flipping one or more of these bits in the IP address of one of the hosts in the group. To become the group’s IP address, the candidate IP address must not already be in use by any of the physical systems on the network, including the dynamic honeypot systems, or by any other group.

For each honeypot group, the open or listening ports (TCP only for this implementation) come from the hosts associated with that honeypot. Recall that during passive network analysis the dynamic honeypot is populating a table listing the open ports associated with each host. For each honeypot group, its open ports are all the open ports associated with any host that is a member of the group. These ports are determined by selecting all elements from the ports table whose IP address is in the current honeypot group.

4.1.3 Deploying The Honeypots

Recall that we have already decided that Honeyd will be used to deploy virtual honeypots. Therefore the dynamic honeypot will create a configuration file for Honeyd based on the results of determinations it has made previously and stored in the dynamic

honeypot database. When Honeyd is launched it will read this file and deploy the honeypots, see figure 3.2 in section 3 for a sample configuration.

Recall that the dynamic honeypot will respond differently depending on whether the source of the connection is coming from the internal or external network. To achieve this each group or honeypot will have two definitions, one to be used when the source is external, and one to be used when the source is internal. Each Honeyd honeypot definition must be given a distinct name when it is created, so the honeypot id number (hpid) will be placed between 'honeypot' and either 'external' or 'internal' depending on which is appropriate. The first line of a Honeyd configuration is "create name", where 'name' is the name of that definition. Using the naming convention above the dynamic honeypot will begin the configuration definition will lines like 'create honeypot20internal' and 'create honeypot20external'.

Each definition, external or internal, will use the same personality, which has already been determined and stored in the honeypots table. For example: set honeypot20internal personality "Windows 2000 SP2" and set honeypot20external personality "Windows 2000 SP2". They will both also use the same default port actions, blocking by default TCP and UDP ports, but leaving ICMP open. Figure 4.6 shows these elements from a sample configuration generated by the dyamic honeypot.

```
set honeypot20extern default tcp action block
set honeypot20extern default udp action block
set honeypot20extern default icmp action open
```

Figure 4.6. Definition of default tcp, udp, and icmp actions in Honeyd.

Once the default setting have been given, the actions for individual ports can be given. This is where the open ports for each honeypot group come into play.

For the external definition an open port can simply be declared as open, which will cause Honeyd to complete the connection handshake, but make no response to any incoming data. Recall from the previous discussion that Honeyd also allows scripts or executables to be attached to an open port using standard I/O. These scripts will be exposed to the external network, so some degree of caution is merited in their use. Complex scripts might contain unknown vulnerabilities flows, and so are avoided here. However, simple scripts that give at least some sense of realism are appropriate. For example, port 22 is typically SSH, and when a client opens an SHH connection, the server responds with a banner like: “SSH-1.99-OpenSSH_3.3p1.” A script to send such a banner is trivial to implement and since it processes no input, it will be very difficult if not impossible to exploit (never say never).

Simple scripts for SSH, FTP, and MySQL were written to provide the appropriate server responses. However not all servers push initial data on their own, for example, HTTP and NET BIOS. But some simple emulation scripts for HTTP do exist, which respond with the HTTP/1.1 400 Bad request. iis.sh [56], by Fabian Bieker, is one such script, and was chosen for its simplicity. NET BIOS, being proprietary, is less understood, and no scripts will be used to simulate it.

To attach the scripts to the Honeyd definition, another table, called *scripts*, was created. It has two fields port and script, where port is the port number for that script, 22 for SSH etc. Script contains the full path to the executable, and any necessary arguments. Honeyd supports the following tokens for variable expansion: ipsrc, ipdst, sport, and dport. This allows the source address and destination address to be passed as argument to the scripts, which can use them to annotate the log files to which they write. The *script*

table can be accessed when the honeypot definition is being created, and if a script is available it will be used, otherwise the keyword open will be used. A sample definition is given in figure 4.7.

```
add honeypot30extern tcp port 22 "./scripts/ssh.sh $ipsrc $sport"
add honeypot30extern tcp port 111 open
add honeypot30extern tcp port 3306 "./scripts/mysql.sh $ipsrc $sport"
add honeypot30extern tcp port 6000 open
```

Figure 4.7. Adding additional open ports and actions to Honeyd.

For the internal definition, each open port will have a proxy to some internal host. This is exactly the information stored in the *ports* table when it is used to determine the open ports for each honeypot group. Therefore when the *honeyports* table is being created, besides inserting port numbers, a proxy for each port is also inserted into the proxy field. An example of a complete definition is shown in figure 4.8.

```
create honeypot20intern
set honeypot20intern personality "Windows 2000 SP2"
set honeypot20intern default tcp action block
set honeypot20intern default udp action block
set honeypot20intern default icmp action open
add honeypot20intern tcp port 135 proxy 192.168.2.22:135
add honeypot20intern tcp port 139 proxy 192.168.2.22:139
add honeypot20intern tcp port 445 proxy 192.168.2.22:445
add honeypot20intern tcp port 1025 proxy 192.168.2.22:1025
```

Figure 4.8. Complete Honeyd style honeypot definition.

Using the dynamic feature of Honeyd, both the internal and external definition for each honeypot can be bound to the same IP address, which has already been selected. This requires the creation of a third honeypot definition, whose name must be unique. The naming convention used is 'honeypot' followed by the honeypot id, for example create dynamic honeypot20. The previous internal and external honeypot definitions are conditionally bound to this dynamic template, as shown in figure 4.9. The condition source ip = 192.168.2.0/26 is determined from the IP address and subnet mask of the local interface at the time the dynamic honeypot is launched.

```
dynamic honeypot20
add honeypot20 use honeypot20intern if source ip = 192.168.2.0/26
add honeypot20 otherwise use honeypot20extern
bind 192.168.2.23 honeypot20
```

Figure 4.9. Conditional binding of different honeypots to one IP address in Honeyd.

Each honeypot group then has three definitions associated with it: an internal definition, and external definition, and a dynamic template that determines when to use the internal and external definitions. A complete set of definitions for one honeypot is shown in figure 4.10.

```
create honeypot20intern
set honeypot20intern personality "Windows 2000 SP2"
set honeypot20intern default tcp action block
set honeypot20intern default udp action block
set honeypot20intern default icmp action open
add honeypot20intern tcp port 135 proxy 192.168.2.22:135
add honeypot20intern tcp port 139 proxy 192.168.2.22:139
add honeypot20intern tcp port 445 proxy 192.168.2.22:445
add honeypot20intern tcp port 1025 proxy 192.168.2.22:1025
create honeypot20extern
set honeypot20extern personality "Windows 2000 SP2"
set honeypot20extern default tcp action block
set honeypot20extern default udp action block
set honeypot20extern default icmp action open
add honeypot20extern tcp port 135 open
add honeypot20extern tcp port 139 open
add honeypot20extern tcp port 445 open
add honeypot20extern tcp port 1025 open
dynamic honeypot20
add honeypot20 use honeypot20intern if source ip = 192.168.2.0/26
add honeypot20 otherwise use honeypot20extern
bind 192.168.2.23 honeypot20
```

Figure 4.10. Complete honeypot definition.

Once the configuration file has been written, the dynamic honeypot can launch Honeyd to deploy the virtual honeypots. Honeyd reads the configuration file and deploys the honeypots. The dynamic honeypot must also launch aprd to enable the binding of honeypots to unused IP space on the subnet. During this process passive analysis is still going on in the background, so the process can be repeated at regular intervals to keep the dynamic honeypot up to date with changes in the network. Prior to updating the

definitions, the dynamic honeypot removes any honeypots that have been fingerprinted as actual hosts.

4.2 Anomaly Based Intrusion Detection

Recall from figure 3.2, that to implement intrusion detection a sniffer will be deployed to capture traffic coming into the dynamic honeypot. The discussion of the design in section 3.3 identified two approaches for generating intrusion alerts based on the honeypot traffic. Since all the traffic is anomalous, by definition of a honeypot, an intrusion alert can defined as any interaction with the honeypot. To implement this approach, the output of the intrusion detection system is simply a listing of the traffic to the honeypot. The second approach involved generating alerts based on some data analysis of the honeypot traffic. Applying a set of rules was proposed as one data analysis technique, and a potential set of rules was developed and is listed in appendix D. In this case intrusion alerts will be generated by a set of rules applied to the honeypot traffic.

Snort was selected to capture the traffic to the honeypots. Snort is an open source network based intrusion detection system that can also be used for packet capture. It is commonly used by the Honeynet project to capture honeypot traffic, and it was deployed to listen on the same interface as Honeyd. A MySQL database was chosen to be the repository of sniffed packets, and Snort is capable of inserting the captured traffic directly into the database. The database will store the alerts, generated by Snort, representing all the traffic to the honeypot. Snort was also configured to log every packet to a tcpdump file.

Snort is packet based, that is, it operates on single packet at a time. To each packet it captures it applies rules. To configure Snort to log everything the following rule was used:

```
Log ip any any <> any any
```

This tells Snort to log any IP based packet from any host on any port to any host on any port. The log output was configured to be a tcpdump file.

4.2.1 Reporting The Honeypot Traffic

To implement the intrusion detection approach, where an intrusion is any interaction with the honeypot, a similar alert rule can be added to Snort.

```
alert ip any any -> any any
```

Since Snort is sniffing the local interface on which Honeyd is deployed, alerts should only occur for the honeypots. However, the above rule generates alerts for ports which are not open on any of the honeypots, and it is possible that it may even alert on IP addresses not bound to any specific honeypot because of how arpd operates. A better rule would limit itself to the established honeypots and their ports.

This is achieved by having the dynamic honeypot define a variable HOME_NET to be a list of the honeypot IP addresses. This variable definition is written to a separate file that is included in the snort configuration file. Unfortunately Snort does not presently have the ability to define multiple ports as a variable (only port ranges or single ports). Any time the dynamic honeypot updates the HOME_NET variable, it sends a signal to Snort, telling it to re-read its configuration file. This keeps HOME_NET up to date within Snort. It is now possible to write a rule that captures only traffic destined for one of the honeypots:

```
alert tcp any -> $HOME_NET any
```

Using the TCP qualifier makes the rule only apply when a connection or connection attempt is made using TCP, which is the only protocol of concern for this implementation. This rule will still alert on traffic to non-honeypot ports (i.e., port scans, etc.) and on traffic that is not a part of an established connection. The flow preprocessor can be used to further address some of these issues, especially since this implementation is only concerned with TCP. Using the flow preprocessor the above rule can be limited to established connections flowing to the server.

```
Alert tcp any -> $HOME_NET any (msg:"honeypot traffic";
flow:to_server,established;)
```

This rule will cause Snort to generate an alert called "honeypot traffic", for each TCP packet that is part of an established connection to one of the honeypots. Snort was configured to place these alerts in a MySQL database. This rule implements the simplest form of honeypot based intrusion detection, where all traffic is considered anomalous. The contents of the database are viewed using ACID.

The ACID main console displays, among other things, the total number of alerts, unique alerts, source and destination addresses, source and destination ports, and a bar graph indicating the percentage of TCP, UDP and ICMP traffic. From the main console a single click can produce a listing of all the alerts, only the most recent, or any number of additional options. From within a listing, clicking on an individual alert brings up a detailed description of the package that caused the alert including both a hex and ASCII version of the payload.

4.2.2 An Additional Alarm Mechanism Based On Honeypot Traffic

An additional alarm mechanism that analyzed the honeypot traffic data was suggested during design, and some preliminary rules were developed and are listed in appendix D. There are two categories of rules, transaction rules and session rules. The transaction rules use only the source and destination IP addresses and ports as inputs. The rules are intended to provide an analysis of the current honeypot traffic, and identify suspicious traffic patterns. The session rules apply to the content of the honeypot traffic. These rules implement conventional intrusion detection by identifying known attacks, or in the case of rule #4, that conventional intrusion detection failed to generate any alarms for a specific session with one of the honeypots.

Since the session rules have much in common with conventional intrusion detection, Snort was used to implement this functionality. Snort was configured to use its default rules set along with the previously discussed log and alert rules. This applies conventional intrusion detection techniques to the honeypot traffic. Any alerts generated by the default rules will be inserted into the same database as the honeypot traffic alerts, and will subsequently be viewable through ACID. While rule number 4 is not implemented, the juxtaposition of alerts from Snort's default rules and the "honeypot traffic" alert provide essentially the same result. By looking at the source and destination of an alert and its surrounding alerts, it should be very clear when a session with one of the honeypots generates only a "honeypot traffic" alert and no alerts from Snort's default rule set.

The transaction rules are based on statistical information about traffic to the honeypot. Snort does not have the capability to generate or process this kind of

information. Implementing these rules will require a separate process that can access the database, and generate the appropriate statistics. Since these rules will use percentages that have yet to be determined, it was decided not to implement them at this time. Instead future analysis of collected honeypot traffic may help determine values and additional metrics that could be used to create and implement real-time intrusion detection using data analysis of honeypot traffic.

The dynamic honeypot intrusion detection system as implemented here consists of a dynamic honeypot that deploys virtual honeypots on a single interface, and a packet sniffer, Snort, that captures all traffic on the local interface. All traffic is logged to a tcpdump file. An intrusion alert, called "honeypot traffic," is issued for any TCP traffic that is part of an established connection and flowing to a server. In addition Snort's default rule set is used to apply conventional intrusion detection techniques to honeypot traffic. The "honeypot traffic" alert indicates a network anomaly; other alerts, generated by default rules, indicate network anomalies that match a previously known attack. The alerts and the packets that caused them are viewed using ACID.

CHAPTER V

TESTING AND RESULTS

Before testing the intrusion detection capabilities of the dynamic honeypot, the dynamic honeypot itself was tested. Three key aspects of the dynamic honeypot were tested: network analysis, honeypot configuration, and honeypot deployment. Once the dynamic honeypot was tested, its ability to carry out intrusion detection was tested. These tests included launching an exploit against a honeypot and an actual host, as well as exposing parts of the test network to the Internet.

5.1 Testing Network Analysis

The first test of the dynamic honeypot was to tests the performance of its network analysis. Once the dynamic honeypot was started, the Linux workstation and the Windows 2000 desktop (see figure 4.2) used a standard browser to connect to and surf the Internet. Allowing for sufficient time and resources to be used (because of buffers there is a certain latency between a packet being sniffed of the wire, and the subsequent insertion into the database) the honeypot database was then independently queried to examine its contents. The results are shown in figure 5.1.

Both systems (the Linux workstation and the Windows 2000 desktop) were accurately fingerprinted and inserted into the honeypot database. However, no open ports (listening ports) were established for either system. Also notice that additional systems, 192.168.2.16 (dynamic honeypot local interface) and 192.168.1.98 (belkin router), were

included in the *flock* table. These are actual hosts that were a part of the production network but for whom no operating system fingerprint was identified. Recall that this was the intended purpose of the *flock* table, and it was clearly being filled appropriately.

```
mysql> select inet_NTOA(ipaddr) as ipaddr,os,count,last from host;
+--------------+-------------------+-------+---------------------+
| ipaddr       | os                | count | last                |
+--------------+-------------------+-------+---------------------+
| 192.168.2.22 | Windows 2000 SP2+ |     3 | 2004-09-19 17:57:25 |
| 192.168.2.62 | Linux 2.4/2.6     |    34 | 2004-09-19 18:01:34 |
+--------------+-------------------+-------+---------------------+
2 rows in set (0.00 sec)

mysql> select inet_NTOA(ipaddr) as ipaddr,port from ports;
Empty set (0.00 sec)

mysql> select inet_NTOA(ipaddr) as ipaddr,time from flock;
+--------------+--------------+
| ipaddr       | time         |
+--------------+--------------+
| 192.168.2.16 | 040919175640 |
| 192.168.1.98 | 040919180119 |
| 192.168.2.62 | 040919180119 |
| 192.168.2.22 | 040919180052 |
+--------------+--------------+
4 rows in set (0.00 sec)
```

Figure 5.1. Contents of the host, ports, and flock table following some passive network analysis

The server had yet to be fingerprinted or identified, so the next test was to connect to it from the external network. This was done using the Windows XP desktop that, as seen in figure 4.2, was external to the test LAN. The server was running both HTTP and FTP services and connections to both of these services were made. Since the Linux workstation was running MySQL, a connection was also made to this service for comparison. Once these tests were complete, the database was queried again, and the results are listed in figure 5.2.

There were now open ports listed in the *ports* table, 80 (HTTP) and 21(FTP) on the server (192.168.2.40), and 3306 (MySQL) on the Linux workstation (192.168.2.62). These were in actuality open or listening ports, and were correctly associated with the appropriate host. The *host* table contained an additional entry for the server

(192.168.2.40) that was fingerprinted by the ACK mode of p0f. The server was also added to the *flock* table.

```
mysql> select inet_NTOA(ipaddr) as ipaddr,port from ports;
+--------------+------+
| ipaddr       | port |
+--------------+------+
| 192.168.2.40 |   21 |
| 192.168.2.40 |   80 |
| 192.168.2.62 | 3306 |
+--------------+------+
3 rows in set (0.00 sec)

mysql> select inet_NTOA(ipaddr) as ipaddr,os,count,last from host;
+--------------+-------------------+-------+---------------------+
| ipaddr       | os                | count | last                |
+--------------+-------------------+-------+---------------------+
| 192.168.2.22 | Windows 2000 SP2+ |     3 | 2004-09-19 17:57:25 |
| 192.168.2.62 | Linux 2.4/2.6     |    35 | 2004-09-19 19:04:21 |
| 192.168.2.40 | Linux 2.4/2.6     |     5 | 2004-09-20 08:44:56 |
+--------------+-------------------+-------+---------------------+
3 rows in set (0.00 sec)

mysql> select inet_NTOA(ipaddr) as ipaddr,time from flock;
+--------------+--------------+
| ipaddr       | time         |
+--------------+--------------+
| 192.168.2.16 | 040920084215 |
| 192.168.1.98 | 040920084215 |
| 192.168.2.62 | 040920084215 |
| 192.168.2.22 | 040919180052 |
| 192.168.2.40 | 040920084215 |
+--------------+--------------+
5 rows in set (0.00 sec)
```

Figure 5.2. Additions to the flock, host, and ports tables after external connections to servers.

The Windows XP computer and an iBook running OS X were temporarily connected to the internal LAN to further test the fingerprinting abilities of the dynamic honeypot. Both systems were correctly fingerprinted (Windows XP as "Windows XP Pro SP1" and OS X as "BSD 4.5"). The entries in the os field of the *host* table in both figure 5.1 and 5.2 show that the dynamic honeypot was truncating any additional fingerprint information, and inserting the desired operating system type and some subsequent version information into the dynamic honeypot database. P0f has far too many fingerprints to test exhaustively, so it was concluded that the dynamic honeypot

was correctly carrying out passive operating system fingerprinting and inserting the appropriate information into the honeypot database.

In addition to operating system fingerprinting, network analysis also includes determining the open or listening ports for each host. From the *ports* table in figure 5.2 it is clear that this processes was started, but that it was not complete. The intention was for additional open or listening ports to be determined over time by analysis of normal network traffic. To simulate the opening of TCP connections, the normal network activity that the dynamic honeypot would use to establish additional open ports, Nmap in full connect mode was run from the dynamic honeypots local interface:

```
Nmap -sT -e eth0 192.168.2.0/24 -F
```

Appendix E lists the results of the Nmap scan. The database was queried again and the resulting *host* and *ports* table are listed in figure 5.3.

```
mysql> select inet_NTOA(ipaddr) as ipaddr,os,count,last from host;
+--------------+-------------------+-------+---------------------+
| ipaddr       | os                | count | last                |
+--------------+-------------------+-------+---------------------+
| 192.168.2.22 | Windows 2000 SP2+ |     3 | 2004-09-19 17:57:25 |
| 192.168.2.62 | Linux 2.4/2.6     |    36 | 2004-09-20 11:06:59 |
| 192.168.2.40 | Linux 2.4/2.6     |     5 | 2004-09-20 08:44:56 |
+--------------+-------------------+-------+---------------------+
3 rows in set (0.00 sec)

mysql> select inet_NTOA(ipaddr) as ipaddr,port from ports;
+--------------+------+
| ipaddr       | port |
+--------------+------+
| 192.168.2.40 |   21 |
| 192.168.2.40 |   80 |
| 192.168.2.62 | 3306 |
+--------------+------+
3 rows in set (0.00 sec)
```

Figure 5.3. Contents of host and ports tables after the internal nmap scan.

From figure 5.3 it is obvious that the dynamic honeypot was unable to determine any open or listening ports from the TCP traffic generated by Nmap. Since Nmap was able to identify the open ports, the problem must be the dynamic honeypot. In fact the problem was that the traffic was not crossing the bridge, and therefore was not seen by

the dynamic honeypot. It was previously determined that using the local interface, in promiscuous mode, would be insufficient as the only location from which to sniff packets. To verify that using the local interface in that way would not address the problem, an additional tcpdump process, listening on the local interface, was piped into the program that extracts open ports:

```
Tcpdump –I eth0 'tcp[13] == 18' | port
```

Nmap was run again, this time from the Linux workstation. This failed to add any additional open ports to the *ports* table, proving that the local interface is insufficient for gathering network information.

To verify that using tcpdump to extract all the open ports does work, Nmap was again run, this time from the external Windows XP machine, which forced all the Nmap traffic across the bridge. The results of the Nmap scan are listed in appendix F. Once the Nmap scan was complete the dynamic honeypot database was queried again. The resulting tables are listed in figure 5.4.

This time the dynamic honeypot was able to identify the open ports for each host on the internal network. In fact the dynamic honeypot identified many of the same open ports as Nmap. There are some discrepancies, which is to be expected. 192.168.2.1 is the address of the LAN interface on the belkin router. The open ports listed by Nmap are likely the result of the belkin's firewall configuration, which was supposed to be disabled, since these same open ports show up in the other hosts as well. An attempt to actually open an ftp connection to 192.168.2.1 failed and confirmed that the belkin router was behaving somewhat unpredictably. This did not affect the functioning of the dynamic honeypot, in fact it could be argued that the dynamic honeypot's passive network analysis is more accurate than Nmap's.

```
mysql> select inet_NTOA(ipaddr) as ipaddr,os,count,last from host;
+--------------+-------------------+-------+---------------------+
| ipaddr       | os                | count | last                |
+--------------+-------------------+-------+---------------------+
| 192.168.2.22 | Windows 2000 SP2+ |     3 | 2004-09-19 17:57:25 |
| 192.168.2.62 | Linux 2.4/2.6     |    38 | 2004-09-20 15:07:04 |
| 192.168.2.16 | Linux 2.4/2.6     |   256 | 2004-09-20 10:42:03 |
| 192.168.2.40 | Linux 2.4/2.6     |     5 | 2004-09-20 08:44:56 |
+--------------+-------------------+-------+---------------------+
4 rows in set (0.00 sec)

mysql> select inet_NTOA(ipaddr) as ipaddr,port from ports;
+--------------+------+
| ipaddr       | port |
+--------------+------+
| 192.168.2.16 |   22 |
| 192.168.2.16 |  443 |
| 192.168.2.16 | 1241 |
| 192.168.2.16 | 3306 |
| 192.168.2.22 |  135 |
| 192.168.2.22 |  139 |
| 192.168.2.22 |  445 |
| 192.168.2.22 | 1025 |
| 192.168.2.40 |   21 |
| 192.168.2.40 |   22 |
| 192.168.2.40 |   80 |
| 192.168.2.40 |  111 |
| 192.168.2.40 |  443 |
| 192.168.2.62 |   22 |
| 192.168.2.62 |  111 |
| 192.168.2.62 | 3306 |
| 192.168.2.62 | 6000 |
+--------------+------+
17 rows in set (0.00 sec)
```

Figure 5.4. Contents of the host and ports tables after external Nmap scan.

Unfortunately the dynamic honeypot is only able to identify open ports that are accessed across the bridge, which make it blind to strictly internal network services. In some situations, a DMZ for example, it is desirable for the dynamic honeypot to emulate only externally accessible services. In such cases the dynamic honeypot would be focused specifically on external threats.

But if the dynamic honeypot is to concern itself with all services on all hosts that are apart of the LAN, then some other solution, in place of or along side of the bridge, must be used. Some options include using a spanning port, or resorting to using some active, rather than passive, measures. The passive bridge approach does identify all the hosts, and was able to fingerprint their operating systems; it is only the identification of

open ports where it proved inadequate.   Other viable and straightforward approaches have been suggested, so this problem was not considered an impediment to continued testing for the dynamic honeypot.

5.2 Testing Honeypot Configuration

Having established that the passive network analysis is working somewhat as expected, and that it's shortcomings can readily be addressed, the next step was testing the honeypot configuration. Testing how the dynamic honeypot configures the virtual honeypots required more than one LAN configuration. Rather than using virtual machines, or some other modification to the LAN, different LAN configurations were directly entered into the database. This made it possible to test several network configurations in a relatively short amount of time.

First the actual LAN configuration, seen in the previous section, was used (some IP addresses have changed due to a system reset). Once the database was populated, shown in figure 5.3, and configuration was complete, the dynamic honeypot had created four honeypot groups, one for each host. This is consistent with what would be expected, since the total address space used by these host is only 64. For four hosts this produces an average distance of 16 and a threshold of 12 (threshold is initially set to 75% of the average distance). Appropriate open ports and proxies were established. The tables pertaining to the honeypot definitions are listed in figure 5.5.

Once the dynamic honeypot filled these tables it wrote the Honeyd configuration file based on them, and then started Honeyd. Appendix G shows the Honeyd configuration file generated from the tables in figure 5.5. Honeyd was successfully launched, indicating there were no syntactical errors in the configuration file.

To carry out further tests different network configurations were manually entered into the database, the dynamic honeypot then configured the honeypots based on these simulated network configurations. Appendix A lists the relevant tables from the honeypot database as well as the Honeyd configuration file for these tests. The various configurations demonstrate the dynamic honeypot's ability to sensibly configure honeypots based on passive network analysis. Similar operating systems are grouped together, but only if they are close enough in IP address space. Each test resulted in a successful launch of Honeyd.

Tests 2 and 3 both use the same network configuration information, but use a different percentage to determine the membership threshold. This shows the effect of using a different percentage of the average distance for the membership threshold. The result being that the two honeypots from test 2 (20 and 30) were combined into one honeypot (10) in test 3, which used the same network information as test 2. The dynamic honeypot also made appropriate choices for IP addresses of the honeypots. For example, in test 1 the two honeypots with more than one member (20 and 30) both only have even numbered IP addresses, consistent with the fact that the hosts in each of these groups have even IP addresses.

5.3 Testing Virtual Honeypot Deployment

From inside the network, the Linux workstation was used to connect to various actual hosts and their corresponding honeypots. Since the Linux workstation is on the internal network the dynamic honeypot should proxy any honeypot connection attempts to an actual host. For example figures 5.6 shows a screen shot of the browser connected to the web server (192.168.2.40) and figure 5.7 shows a screen shot of the browser

connected to the honeypot associated with the web server (192.168.2.41). Both pages are the same, indicating the honeypot was acting as a proxy for the server, as was intended. Similar tests were carried out using SSH, FTP, and netcat, all with similar positive results.

```
mysql> select hpid,inet_NTOA(ipaddr) as ipaddr,os from honeypots;
+------+--------------+--------------------------------------------+
| hpid | ipaddr       | os                                         |
+------+--------------+--------------------------------------------+
|   10 | 192.168.2.23 | Windows 2000 SP2                           |
|   20 | 192.168.2.63 | Linux 2.4.16 - 2.4.18                      |
|   30 | 192.168.2.17 | Linux 2.4.16 - 2.4.18                      |
|   40 | 192.168.2.41 | Linux 2.4.16 - 2.4.18                      |
+------+--------------+--------------------------------------------+
4 rows in set (0.00 sec)

mysql> select hpid,inet_NTOA(ipaddr) as ipaddr from honeyhosts;
+------+--------------+
| hpid | ipaddr       |
+------+--------------+
|   10 | 192.168.2.22 |
|   20 | 192.168.2.62 |
|   30 | 192.168.2.16 |
|   40 | 192.168.2.40 |
+------+--------------+
4 rows in set (0.00 sec)

mysql> select hpid,port,proxy from honeyports;
+------+------+--------------+
| hpid | port | proxy        |
+------+------+--------------+
|   10 |  135 | 192.168.2.22 |
|   10 |  139 | 192.168.2.22 |
|   10 |  445 | 192.168.2.22 |
|   10 | 1025 | 192.168.2.22 |
|   20 |   22 | 192.168.2.62 |
|   20 |  111 | 192.168.2.62 |
|   20 | 3306 | 192.168.2.62 |
|   20 | 6000 | 192.168.2.62 |
|   30 |   22 | 192.168.2.16 |
|   30 |  443 | 192.168.2.16 |
|   30 | 1241 | 192.168.2.16 |
|   30 | 3306 | 192.168.2.16 |
|   40 |   21 | 192.168.2.40 |
|   40 |   22 | 192.168.2.40 |
|   40 |   80 | 192.168.2.40 |
|   40 |  111 | 192.168.2.40 |
|   40 |  443 | 192.168.2.40 |
+------+------+--------------+
17 rows in set (0.02 sec)
```

Figure 5.5. Contents of the honeypots, honeyhosts, and honeyports tables as a result of configuration.

From the external network, using the Windows XP machine, the same types of connections were made. Figure 5.6 is a screen shot of the browser connected to the honeypot at 192.168.2.41. This time the honeypot is using the iis.sh script and not

proxying to an internal host. As seen in figure 5.8, the script provides some degree of realism.

Using netcat other connections were made to internal honeypots, figure 5.9 shows some of these. The honeypots deployed by the dynamic honeypot preformed as expected. When an internal host connected to one of the honeypot it acted as proxy for some actual host on the internal network. When an external host connected to one of the honeypots it functioned as port monitor, and in some cases used very simple scripts to simulate aspects of a service.

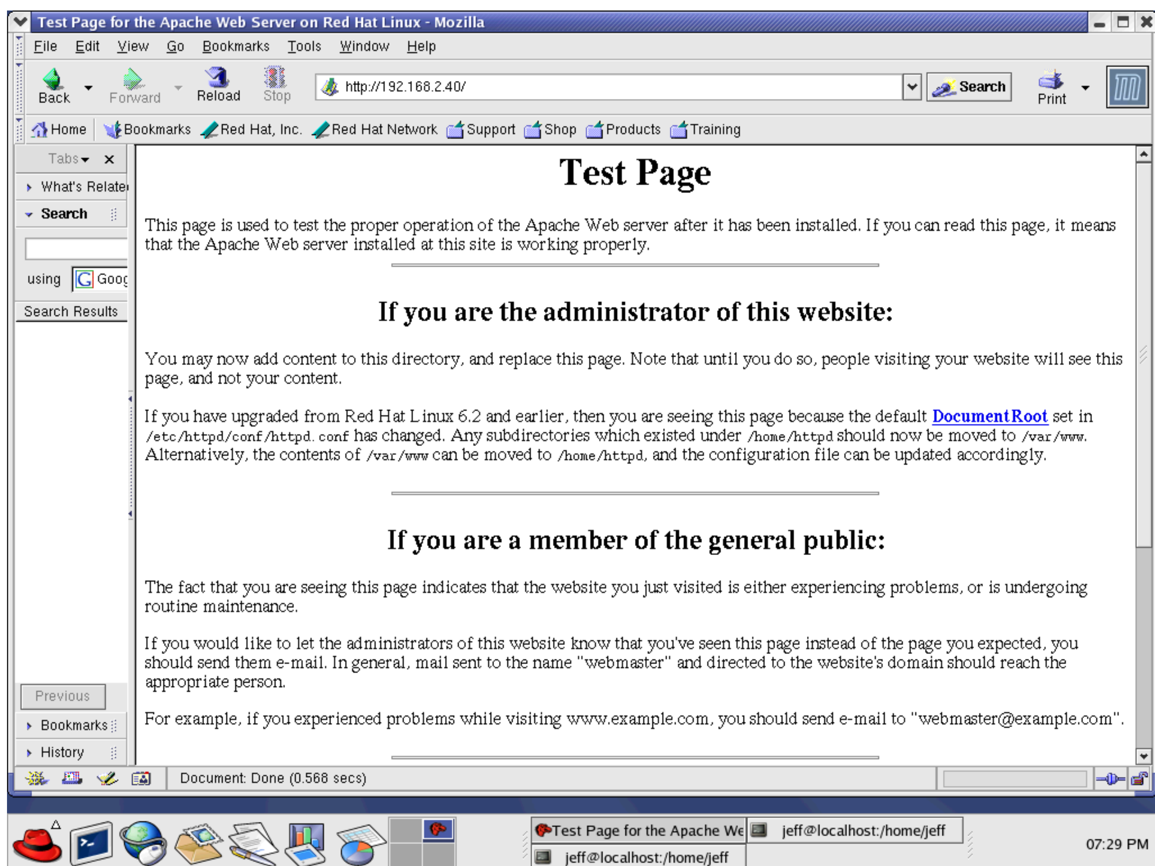

Figure 5.6. Connection to the web server from an internal host.

During the tests the dynamic honeypot accurately distinguished between internal and external hosts; however, this may have been an artifact of the network configuration

(the Windows XP system uses a wireless connection to the Linksys router). Considering the potential risk that would result from on external host being identified by the system as internal, extensive testing of the mechanism for distinguishing between internal and external hosts would be needed prior to deployment on a real network.

The dynamic honeypot was deployed on a single subnet, 192.168.2.0. While test 4 in appendix A shows that the dynamic honeypot can handle configuration of multiple subnets, it has no means of deploying virtual honeypots on another subnet. To be useful in a production environment the dynamic honeypot will need the ability to deploy honeypots on multiple subnets. Since multiple other subnets may use NAT, passive network analysis may need to address this issue as well.

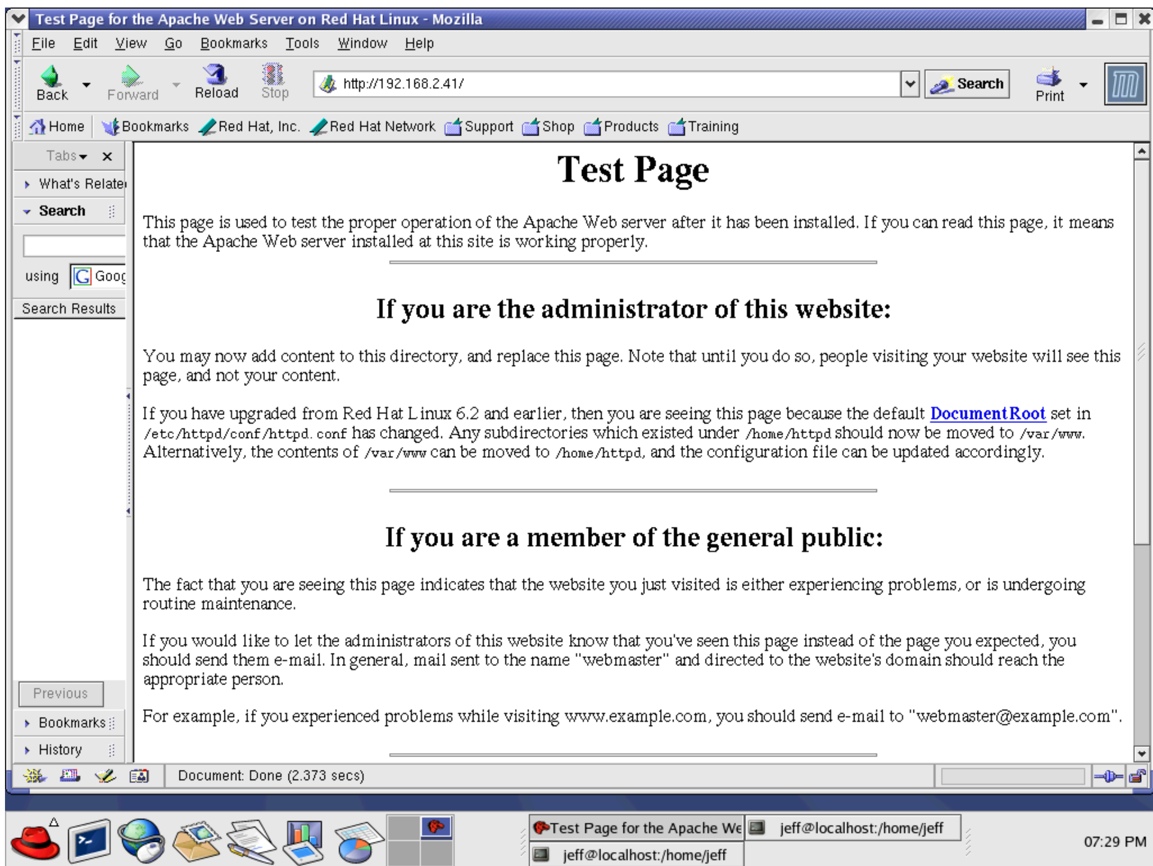

Figure 5.7. Connection to the honeypot web server from an internal host.

Another problem that was encountered was that *aprd*, used to attract traffic to the honeypot can interfere with normal network traffic. The most noticeable interference came when trying to add an additional host to the LAN. The IP address being handed out by the DHCP server on the Belkin router was an already in use as a honeypot. Since the DHCP server new nothing of the assigned IP address the new host was unable to connect to the network. To overcome this problem, the dynamic honeypot had to be stopped, allowing the system to connect, and then the dynamic honeypot was started up again.

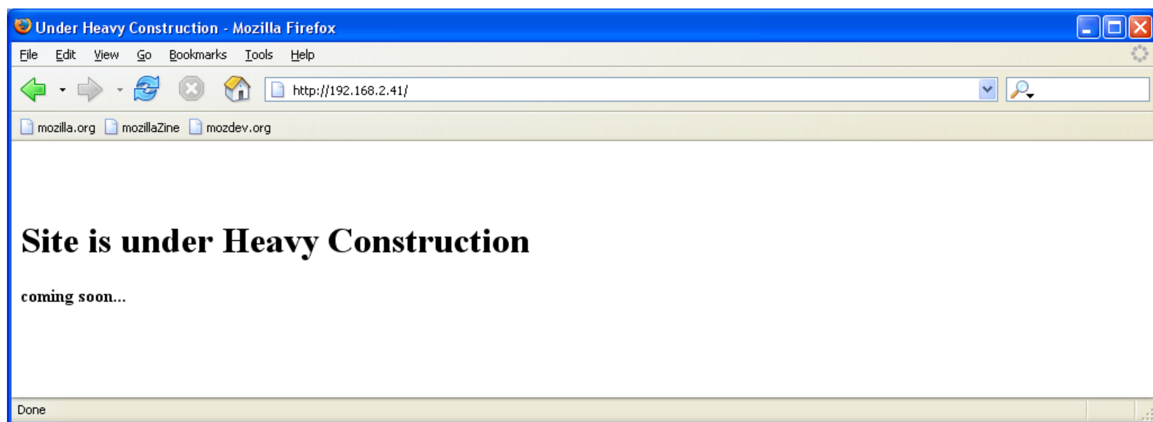

Figure 5.8. Connection to the honeypot web server from external host.

5.4 Testing The Intrusion Detection Abilities

With the dynamic honeypot performing as expected some preliminary testing of the intrusion detection capabilities of the system were carried out. Recall that Snort was deployed to monitor the honeypot interface, and configured to alert on any established TCP connection flowing to the server, as well as when any of its standard rules are triggered. ACID was configured to provide a GUI to the alerts generated by Snort, which are stored in a MySQL database. Any alerts generated from the previous testing were deleted; figure 5.10 shows the main ACID console with no alerts.

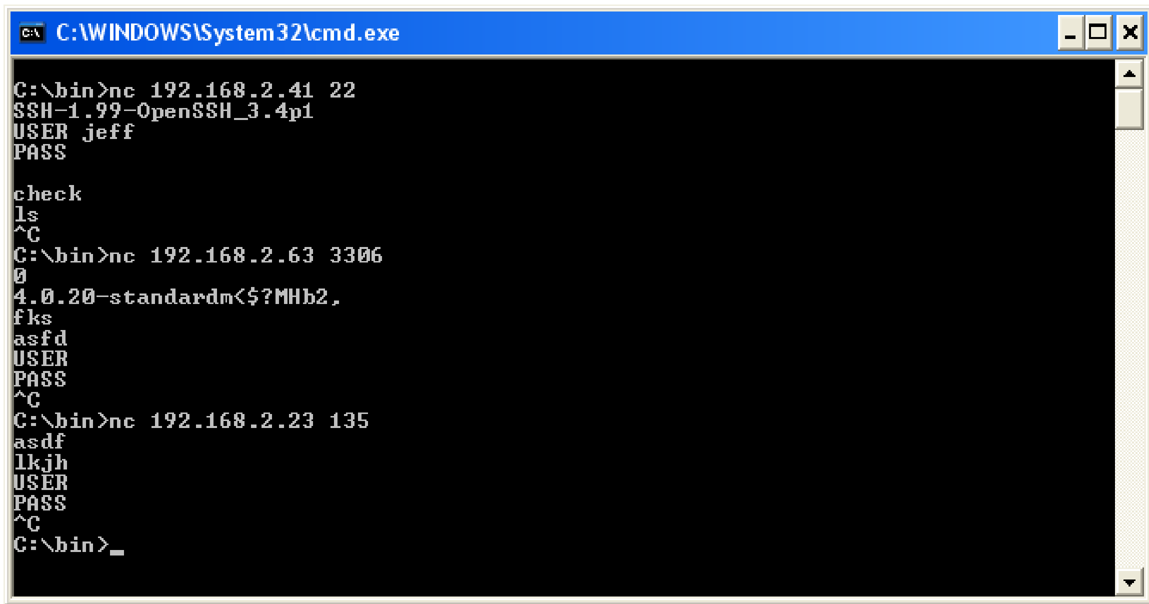

Figure 5.9. Connection to various honeypot services from external host.

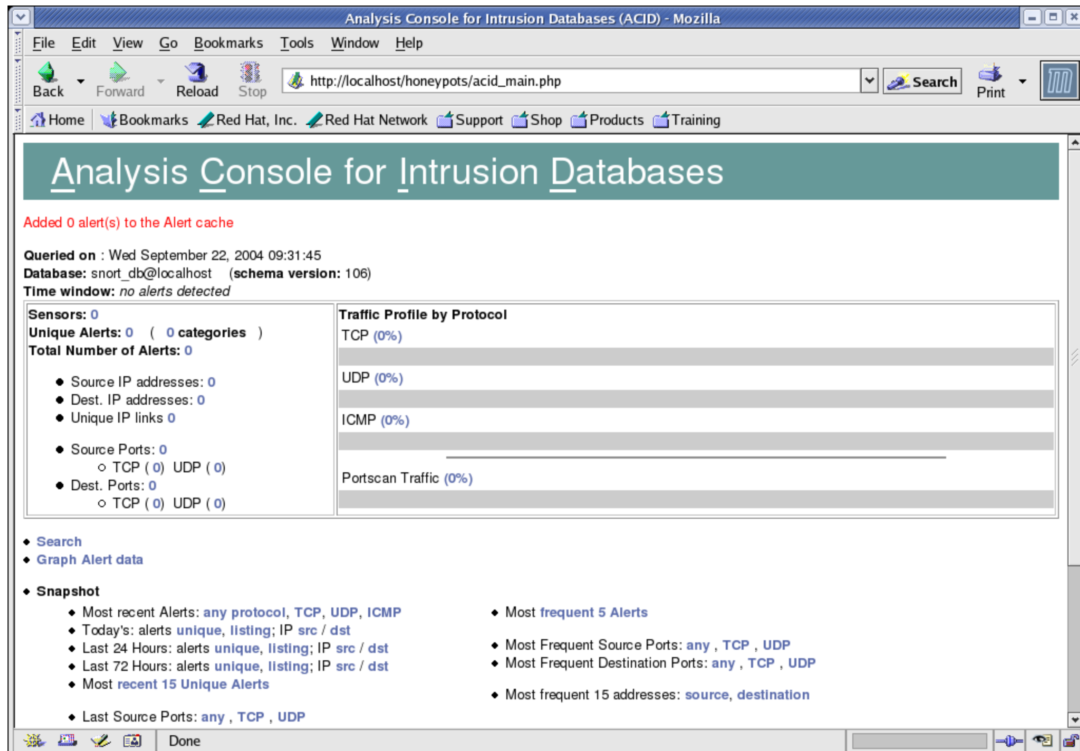

Figure 5.10. ACID console for the honeypots prior to any activity.

In addition to the Snort sensor deployed on the dynamic honeypot interface, a Snort sensor was also deployed on the bridge, using only the default rule set. The bridge sensor was a precautionary measure for times when the LAN was exposed to the Internet, and also served as a comparison for some of the results of the dynamic honeypot intrusion detection system.

5.4.1 Controlled Intrusions

Since the dynamic honeypot treats internal and external hosts differently, a set of preliminary tests from both the internal and external network was conducted. These consisted of simple interactions, designed to demonstrate what alarms might look like. The tests consisted of using Nmap to scan the 192.168.2.0/24 subnet on port 80 only and connecting to the actual web server and the honeypot web server. In addition to these tests an actual exploit against the Windows 2000 desktop and its associated honeypot was launched, and a backdoor was installed on the Windows 2000 desktop using netcat.

Figure 5.11 shows the ACID listing of all the alerts generated during the tests[1]. The ICMP PING NMAP alerts come from the default Snort rules, and are consistent with the fact that NMAP was used against the network. The honeypot traffic alerts are generated by the rule that was added to implement anomaly detection using the dynamic honeypots. These alerts were expected as a result of the tests, and there presence indicates that the intrusion detection alert mechanism was operating correctly.

Alerts 4 through 9 were generated when the Windows XP system (on the external network at 192.168.1.100) connected to one of the honeypots (192.168.2.41) with port 80 open. Using ACID it is possible to view the individual packet that caused the alert,

[1] 2 ICMP alerts and one honeypot traffic alert were deleted so that all the alerts would fit on the screen at one time

simply by clicking on the ID for that packet. This feature of ACID will make using the output of the dynamic honeypot intrusion detection system easy to use for incidence response. Figure 5.12 shows the result of clicking on ID #8-(5-369).

Alerts 25, 26, and 27 pertain to the exploit that was launched, first against the honeypot, and then successively against the actual Windows 2000 host. By examining the packets that caused the alert, at least part of the actual exploit code can be seen. Interestingly Snort did not generate any other alerts, which means had this attack been against an actual host, Snort would not have detected it. In fact, the bridge Snort sensor failed to generate any alarm related to the exploit. Figure 5.13 shows the packet from alert number 26, with the exploit payload.

ACID: Query Results - Mozilla

File Edit View Go Bookmarks Tools Window Help

Displaying alerts 1-28 of 28 total

ID	< Signature >	< Timestamp >	< Source Address >	< Dest. Address >	< Layer 4 Proto >
#0-(5-358)	honeypot traffic	2004-09-22 11:43:46	192.168.1.100:4506	192.168.2.41:80	TCP
#1-(5-352)	[arachNIDS][snort] ICMP PING NMAP	2004-09-22 11:42:02	192.168.1.100	192.168.2.41	ICMP
#2-(5-353)	[arachNIDS][snort] ICMP PING NMAP	2004-09-22 11:43:11	192.168.1.100	192.168.2.23	ICMP
#3-(5-373)	honeypot traffic	2004-09-22 11:53:52	192.168.1.100:4524	192.168.2.41:80	TCP
#4-(5-355)	[arachNIDS][snort] ICMP PING NMAP	2004-09-22 11:43:12	192.168.1.100	192.168.2.17	ICMP
#5-(5-372)	honeypot traffic	2004-09-22 11:53:52	192.168.1.100:4524	192.168.2.41:80	TCP
#6-(5-371)	honeypot traffic	2004-09-22 11:53:52	192.168.1.100:4524	192.168.2.41:80	TCP
#7-(5-370)	honeypot traffic	2004-09-22 11:53:51	192.168.1.100:4524	192.168.2.41:80	TCP
#8-(5-369)	honeypot traffic	2004-09-22 11:53:51	192.168.1.100:4524	192.168.2.41:80	TCP
#9-(5-368)	honeypot traffic	2004-09-22 11:53:51	192.168.1.100:4524	192.168.2.41:80	TCP
#10-(5-364)	[arachNIDS][snort] ICMP PING NMAP	2004-09-22 11:48:37	192.168.2.62	192.168.2.23	ICMP
#11-(5-365)	[arachNIDS][snort] ICMP PING NMAP	2004-09-22 11:48:37	192.168.2.62	192.168.2.17	ICMP
#12-(5-366)	[arachNIDS][snort] ICMP PING NMAP	2004-09-22 11:48:38	192.168.2.62	192.168.2.41	ICMP
#13-(5-374)	honeypot traffic	2004-09-22 11:56:48	192.168.2.62:58392	192.168.2.41:80	TCP
#14-(5-375)	honeypot traffic	2004-09-22 11:56:48	192.168.2.62:58392	192.168.2.41:80	TCP
#15-(5-376)	[snort] ATTACK-RESPONSES 403 Forbidden	2004-09-22 11:56:48	192.168.2.41:80	192.168.2.62:58392	TCP
#16-(5-377)	honeypot traffic	2004-09-22 11:56:48	192.168.2.62:58392	192.168.2.41:80	TCP
#17-(5-378)	honeypot traffic	2004-09-22 11:56:48	192.168.2.62:58392	192.168.2.41:80	TCP
#18-(5-379)	honeypot traffic	2004-09-22 11:56:48	192.168.2.62:58392	192.168.2.41:80	TCP
#19-(5-380)	honeypot traffic	2004-09-22 11:56:48	192.168.2.62:58392	192.168.2.41:80	TCP
#20-(5-381)	honeypot traffic	2004-09-22 11:56:48	192.168.2.62:58392	192.168.2.41:80	TCP
#21-(5-382)	honeypot traffic	2004-09-22 11:56:48	192.168.2.62:58392	192.168.2.41:80	TCP
#22-(5-383)	honeypot traffic	2004-09-22 11:56:48	192.168.2.62:58392	192.168.2.41:80	TCP
#23-(5-384)	honeypot traffic	2004-09-22 11:56:48	192.168.2.62:58392	192.168.2.41:80	TCP
#24-(5-385)	honeypot traffic	2004-09-22 11:56:48	192.168.2.62:58392	192.168.2.41:80	TCP
#25-(5-388)	honeypot traffic	2004-09-22 15:19:50	192.168.1.101:32789	192.168.2.23:135	TCP
#26-(5-389)	honeypot traffic	2004-09-22 15:19:50	192.168.1.101:32789	192.168.2.23:135	TCP
#27-(5-390)	honeypot traffic	2004-09-22 15:20:26	192.168.1.101:32789	192.168.2.23:135	TCP

Figure 5.11. ACID listing of alerts after initial test traffic.

To simulate some worm activity, the exploit[2] was systematically launched from Linux workstation on the internal network against all hosts on the internal network. Similar alerts were observed, but this time one of Snorts default rules, NETBIOS DCERPC IsystemActivator bind attempt, did fire when the exploit was launched against the honeypot with port 135 (the attack port) open. The “honeypot traffic” rule also fired. In this case the dynamic honeypot intrusion detection system detected and generated an alert as a result of an actual exploit attempt against one of its honeypots.

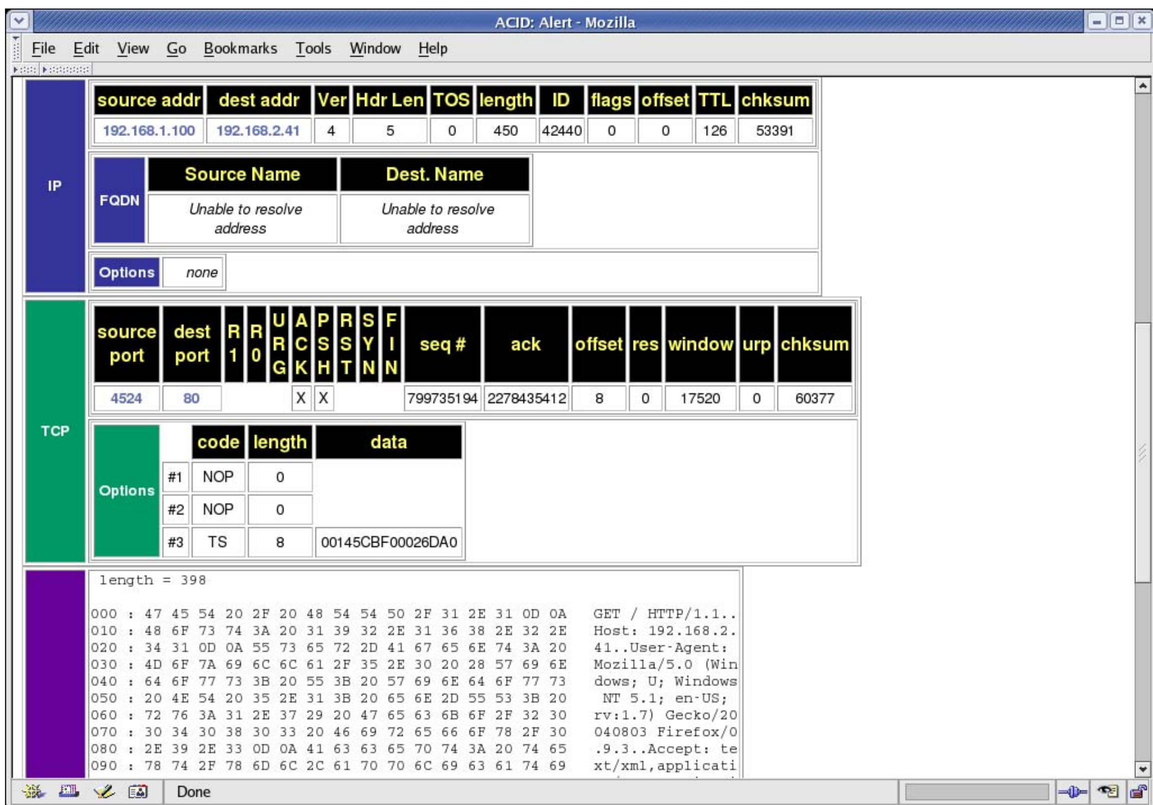

Figure 5.12. ACID view of an individual packet that
caused an alert.

These initial tests show the potential for intrusion detection using a dynamic honeypot. Initially non-malicious traffic was used to observe how the dynamic honeypot

[2] This exploit was used by the original Blaster worm.

intrusion detection system would respond to anomalous traffic. All of the anomalous traffic generated an alert. A known exploit attempt launched from the external network against a honeypot generated an alert, while no alert was generated by the conventional intrusion detection system deployed on bridge.

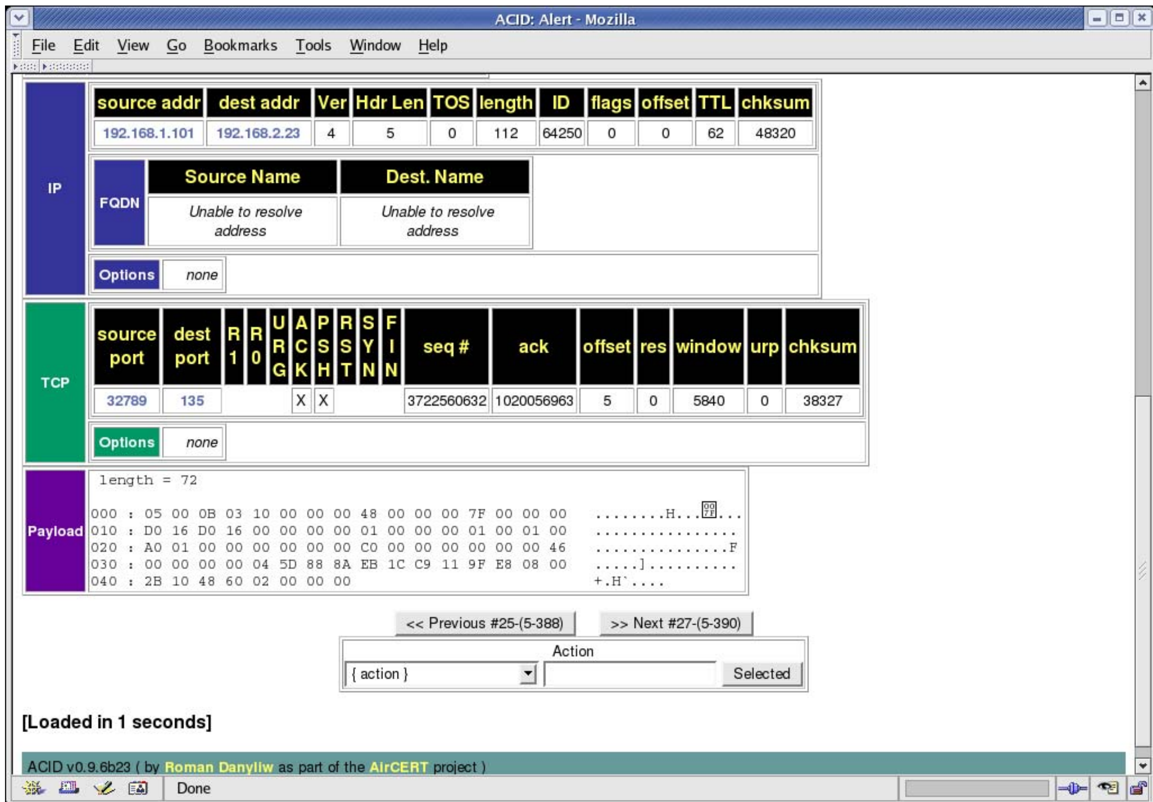

Figure 5.13. ACID view of an exploit packet.

An interesting result came from re-examining the honeypot definitions after the tests had been conducted. As a result of periodic updating, the dynamic honeypot reconfigures its honeypots. The new configuration, listed in appendix B, reflected the open ports created by the attack. One of the honeypots now had port 666 and 17666 open. 666 was the port of the bindshell created by the attack, and 17666 was the port on which the backdoor was listening.

5.4.2 Real World intrusions

The next round of testing involved exposing parts of the network to the Internet and potential real attacks. This was done in two phases, the first phase exposed some of the honeypots to potential attacks, and the second phase exposed an actual system. Both tests are in a sense a simulation of a mis-configured firewall. Since the ISP was only providing one IP address it was necessary to use NAT on both routers. This required a slight configuration change, but one that is essentially invisible to the dynamic honeypot.

The first test forwarded some traffic from the Internet to three of the honeypots. Web and FTP requests will be forwarded to 192.168.2.41, which is the honeypot associated with the Server. Requests on ports 135 and 139, NET BIOS, were forwarded to 192.168.2.47, the honeypot associated with the Windows 2000 desktop. Finally, any requests on port 3306, MySQL, were forwarded to 192.168.2.63, the honeypot associated with the Linux workstation. The test was run for 12 hours, between 10 am. and 10 pm. on September 23.

A total 492 alerts were generated, all on port 80. Figure 5.14 shows the ACID home page at the conclusion of the test. There were 7 unique alerts, all from the same source. Figure 5.15 is a listing from ACID of each unique alert. The majority of the alerts, 351, were generated by the custom honeypot traffic alert rule, the rest are from the Snort's default rule set. All of the attacks came from the same source, between 10:30 and 11:15. The attacks largely consisted of directory traversal attempts and cmd.exe attempts. None of the alerts were generated by legitimate traffic, anecdotal evidence of a low false positive rate.

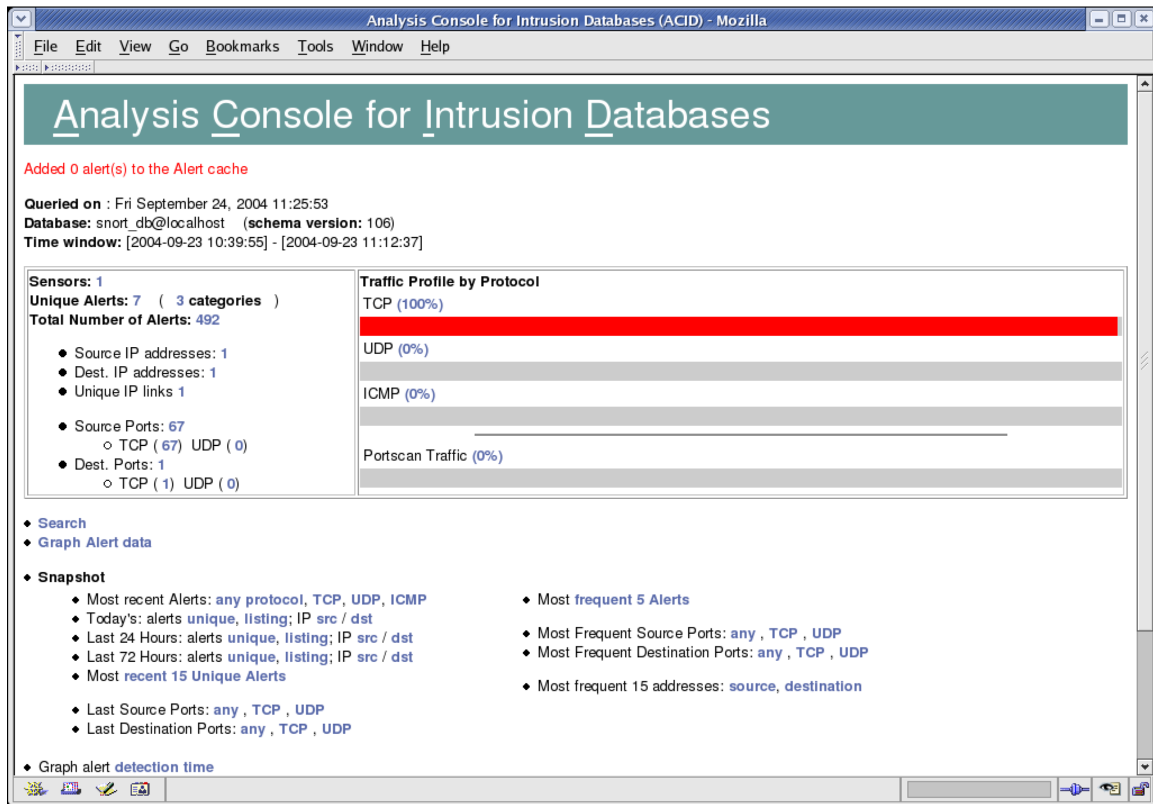

Figure 5.14. ACID console after some external traffic is allowed to reach the honeypots.

The second test involved exposing some of the actual systems to the Internet. Since the Windows 2000 Desktop had already been shown to have at least one vulnerability, it was placed on a DMZ, removing any firewall or NAT protection. The test was carried out for three days. While there was interaction with the Windows system, mostly on the UDP messenger service, no successful attacks were launched against it. It was not possible therefore, to test the intrusion detection abilities of the dynamic honeypot with this test.

ACID: Alert Listing - Mozilla

File Edit View Go Bookmarks Tools Window Help

ACID **Alert Listing** Home Search | AG Maintenance

[Back]

Added 0 alert(s) to the Alert cache

Queried DB on : Fri September 24, 2004 11:09:05

Meta Criteria	time >= [09 / 21 / 2004] [11 : * : *] ...clear...
IP Criteria	*any*
Layer 4 Criteria	*none*
Payload Criteria	*any*

Displaying alerts 1-7 of 7 total

	Signature	Classification	Total #	Sensor #	Src. Addr.	Dest. Addr.	First	Last
	[snort] WEB-IIS cmd.exe access	web-application-attack	68 (14%)	1	1	1	2004-09-23 11:12:19	2004-09-23 11:12:35
	honeypot traffic	*unclassified*	351 (71%)	1	1	1	2004-09-23 10:39:55	2004-09-23 11:12:37
	[arachNIDS][snort] WEB-MISC http directory traversal	attempted-recon	40 (8%)	1	1	1	2004-09-23 11:12:19	2004-09-23 11:12:35
	[cve][icat][snort] WEB-IIS unicode directory traversal attempt	web-application-attack	15 (3%)	1	1	1	2004-09-23 11:12:19	2004-09-23 11:12:33
	[snort] (http_inspect) DOUBLE DECODING ATTACK	*unclassified*	13 (3%)	1	1	1	2004-09-23 11:12:19	2004-09-23 11:12:32
	[cve][icat][snort] WEB-IIS unicode directory traversal attempt	web-application-attack	3 (1%)	1	1	1	2004-09-23 11:12:22	2004-09-23 11:12:35
	[cve][icat][snort] WEB-IIS unicode directory traversal attempt	web-application-attack	2 (0%)	1	1	1	2004-09-23 11:12:27	2004-09-23 11:12:35

Action

Figure 5.15. Listing of the unique alerts generated while exposing the honeypot to the Internet.

CHAPTER VI

CONCLUSIONS AND FUTURE DIRECTIONS

6.1 Conclusions

Computers and networks are becoming a more central part of our everyday lives, they providing us with the convenience of e-commerce and e-banking, and the necessities of electricity and water. In response to this growing dependence are both the difficulty and the need for better computer security. Good security is best achieved through a combination of security technologies, policies and procedures. Today firewalls and anti-virus software are an absolute necessity and intrusion detection systems are becoming more predominant. It is almost certain that good computer security in the future will rely on an ever-increasing variety of security technologies that will include new types of intrusion detection systems.

In this thesis a novel type of intrusion detection system, based on a dynamic honeypot, was explored. The dynamic honeypot is a plug and play honeypot solution that eliminates the need for time consuming configuration and maintenance. Once deployed on a network, an intrusion detection system can be set up by monitoring traffic to the honeypot. The intrusion detection system implements anomaly detection since all traffic to a honeypot is anomalous. Moreover, since the dynamic honeypot requires little configuration or supervision, neither does intrusion detection.

The dynamic honeypot implementation was successful in deciding how many honeypots to deploy, what they should look like, and where they should be deployed. It

was discovered that while passive network analysis is effective for identifying hosts on a network, it was not as effective at identifying the open ports associated with those hosts. Using virtual honeypot technology, the dynamic honeypot was able to deploy the honeypots once they were configured. Once deployed, alerts generated by the dynamic honeypot intrusion detection system were indeed some type of anomaly. Therefore the dynamic honeypot did appropriately identify intrusive behavior, even if that behavior did not always take the form of an exploit, exploit attempt, or attack.

Using a dynamic honeypot for intrusion detection and network monitoring was never intended to replace existing intrusion detection systems. Instead it was designed to supplement these and other security systems. Therefore it is not necessary that it identify every intrusion, which it did not, only that it identifies some intrusions that might not be identified by other security systems, which it did.

Discerning the meaning or significance of the alerts generated by the dynamic honeypot intrusion detection system is left up to the system administrator. For example, if a connection is made to a honeypot web server from an internal host, the dynamic honeypot intrusion detection system, via the ACID console, will report this with one or more alerts. From the alert the administrator will immediately know the source IP address of the anomaly. By clicking on the appropriate link she can also see the packet that caused the alert. In addition there may be other alerts from Snort's rule set indicating that the packet matches one of its signatures. From the contents of the packet or a Snort alert, the administrator begins the process of deciding whether this is some employee, maliciously or benignly poking around the network; or whether an internal host has been compromised in some way.

The system has other potential uses beyond the obvious network monitoring capability. A dynamic honeypot intrusion detection system might be used in the area of risk assessment. The passive network analysis component alone can be useful in this area. It automatically provides a detailed listing of the hosts on the network and which ports are open. An automated and independent source of this information, such as the dynamic honeypot with an added GUI that presents the up to date passive network analysis results, would be valuable in verifying what network administrators claim to be their network and host configurations. In addition, the alert mechanism could provide to those carrying out risk analysis clandestine information about potential holes in the perimeter defenses and insider threats.

Another very interesting potential use could be for the collection of forensic data. A dynamic honeypot could be placed on an organizations network, and configured such that its logs, etc., meet all legal requirement for admissible evidence. In addition those logs, etc., could be structured such that finding relevant forensic data was very straightforward. If a serious security event occurred it would (hopefully) include interaction with a honeypot. In such situation evidence would quickly and readily be available, and have very little recovery cost. In addition, assuming there was sufficient evidence, the IT department could focus on repairing any damage without being concerned over destroying evidence.

6.2 Directions For Future Research

Because it has the ability to detect previously unknown attacks, anomaly based intrusion detection is highly desirable. However it is proving difficult to effectively implement. While the dynamic honeypot based intrusion detection system described in

this thesis does not provide network wide, comprehensive intrusion detection, it does provide anomaly based intrusion detection. Rather than simply reporting any traffic to the honeypot as an intrusion, intelligent data analysis techniques could be applied to the anomalous traffic. This idea was proposed earlier, but not explored, and is an avenue of future research with great possibilities.

There are several directions such research might take. The simplest direction, mentioned previously, would be to develop a set of rules that could be applied to the captured honeypot traffic. This approach could be a real-time system, where the rules processed honeypot traffic as it was captured and generated an ongoing report. Or a collection of honeypot traffic could be analyzed, after it was captured, generating a static periodic report. In either case the rules would provide high-level threat evaluation data, and make recommendations for incidence response.

Another avenue of exploration is the development of a system that analyzes the captured honeypot traffic in an attempt to write detection rules for a conventional intrusion detection system. For example, say Snort is deployed as a NIDS, and then a dynamic honeypot intrusion detection system deployment captures traffic to the honeypot, analyzes it, and generates additional detection rules for Snort. These rules could be automatically added the running NIDS. This could potentially create a self-learning comprehensive intrusion detection system.

Another potential direction for further research would be to explore combining the honeypot configurations, the hosts, which host are associated with which honeypot, and the honeypot traffic into a comprehensive graphical reporting mechanism. As was shown in one of the tests, passive network analysis picked up on the bindshell and

backdoor created by the RPCDOM exploit. Having this information in an easy to read and understand format that shows all the relevant associations could provide valuable information to a network administrator, especially for a large network.

REFERENCES

[1] L. Spitzner, Honeypots *Tracking Hackers*, Addison-Wesley, Boston, 2003.

[2] L. Spitzner, "Dynamic Honeypots"
http://www.securityfocus.com/infocus/1731

[3] L. Spitzner, "Problems and Challenges with Honeypots"
http://www.securityfocus.com/infocus/1757

[4] L. Spitzner, "Know Your Enemy:Passive Fingerprinting"
http://honeynet.org/papers/finger/

[5] Spitzner, L, "Honeypots: catching the insider threat", Computer Security Applications Conference, 2003. Proceedings. 19th Annual , 2003 Pages:170 - 179

[6] Provos, Niels, "A Virtual Honeypot Framework", CITI Technical Report 03-1, October 21, 2003, (Center for Information Technology Integration, University of Michigan)

[7] Laurent Oudot, "Fighting Internet Worms With Honeypots"
http://wwwsecuirtyfocus.com/infocus/1740

[8] http://lcamtuf.coredump.cx/p0f.shtml

[9] M. Zalewski and William Stearns. "Passive OS Fingerprinting Tool", www.stearns.org/p0f/README.

[10] J. Sherif, T. Dearmond, " Intrusion Detection: Systems and Models" Proceedings of the Eleventh IEEE International Workshop on Enabling technologies: Infrastructure for Collaborative Enterprises (WETICE'02) (2002).

[11] D. Schwartz, S. Stoecklin, and E. Yilmaz. "A Case-Based Approach to Network Intrusion Detection", Proceedings on the Fifth International conference on Information Fusion, vol.2 8-11, july 2002.

[12] P. Harmer, P. Williams, G. Gunsch, G. Lamont. "An artificial Immune system Architecture for Computer Secutity Applications", IEEE Transactions on Evolutionary Computation, vol. 6, no 3., June 2002.

[13] J. Copeland, R. Garcia, "Real-time Anomaly Detection using Soft-computing techniques" SoutheastCon 2001. Proceedings. IEEE , 30 March-1 April 2001 Pages:105 – 108.

[14] B. Gao, H. Ma, Y. Yang, "HMMS (Hidden Markov Models) Based on Anomaly Intrusion Detection Method" Proceedings of the First international conference on Machine Learning and Cybernetics, November 4-5, 2002. 381-385.

[15] Lance Spitzner, "The Honeynet Project: Trapping the Hackers", IEEE Security and Privacy, March/April 2003, vol. 1, number 2, 15-23.

[16] B. McCarty, "Botnets: Big and Bigger", IEEE Security and Privacy, July/.Aug. 87-90.

[17] D. Forte, "Part 1: Deploying Honeypots: Project background and implications," Network Security, Vol. 2003, Issue 7, July 2003, 13-14.

[18] D. Forte, "Part II: Honeypots in Detail: the Variations," Network Security, Vol. 2003, Issue 7, July 2003, 14-15.

[19] "Symantec push honeypot as add-on to IDS," Network Security, vol. 2003, Issue 7, July 2003, 2-3.

[20] M. Botha, R. Von Solms, K. Perry, E. Loubser, G. Yamoyany, "The Utilization of Artificial intelligence in a Hybrid Intrusion Detection System," Proceedings of SAICSIT, 2002, pages 149-155.

[21] J. Levine, R. LaBella, H. Owen, D. Contis, B. Culver, "The use of Honeynets to Detect Exploited Systems Across Large Enterprise Networks," Proceedings of the IEEE Workshop on Information Assurance, West Point, NY, June, 2003, 92 – 99.

[23] F. Zang, S. Zhou, Z. Quin, J. Liu, "Honeypot: a Supplemented Active Defense System for Network Security," Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies, August 27-29, 2003, 231-235.

[24] D. Denning, "An Intrusion Detection Model," IEEE Transactions on Software Engineering, 13, 2, 222-232, 1967.

[25] D. Dasgupta, H. Brian, "Mobil Security Agents for Network Traffic Analysis," DARPA Information Survivability Conference & Exposition II, 2001. DISCEX '01. Proceedings , Volume: 2 , 12-14 June 2001
Pages:332 - 340 vol.2

[26] F. Carrettoni, S. Castano, G. Martella, P. Samarati, "RETISS: A Real Time Security System For Threat Detection Using Fuzzy Logic," Proceedings of the

25th Annual 1991 IEEE International Carnahan Conference on Security Technology, 1-3 Oct. 1991
Pages:161 – 167.

[27] H. Debar, M. Becker, D. Siboni, “A Neural Network Component for an Intrusion Detection System,” Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy, 1992., 4-6 May 1992
240 – 250.

[28] T. Lunt, R. Jagannathan, R. Lee, A. Whitehurst, “Knowledge-Based Intrusion Detection” Proceedings of the Annual AI Systems in Government Conference, Washington D.C., March 27-31, 1989, 102-107.

[29] Skoudis, Ed. 2002. Counter Hack. New Jersey: Prentice Hall.

[30] E. Hamed, “An Agent Based Intrusion Detection System Using Fuzzy Logic For Computer System Threat Evaluation,” Phd. Dissertation, University of Louisville, Louisville, Kentucky, 2001.

[31] W. Stallings. 2003. Network Security Essentials, New Jersey, Prentice Hall.

[32] M. Bishop. 2002. Computer Security, New York, Addison-Wesley.

[33] C. Stoll, The Cuckoo’s Egg: Tracking a Spy Through the Maze of Computer Espionage. NewYork, Pocket Books, 1990.

[34] W. Cheswick “An Evening with Berferd In Which a Cracker is Lured, Endured, and Studied,” CD-ROM accompanying: L. Spitzner, Honeypots *Tracking Hackers*, Addison-Wesley, Boston, 2003.

[36] Anderson, J. P. “Computer Security Threat Monitoring and Surveillance.” Technical Report, James P. Anderson Co., Fort Washington, Pennsylvania, 1980.

[37] N. Karwetz, “Anti-Honeypot Technology,” IEEE Security and Privacy, January/February 2004, 76-79.

[38] “Honeypots-Not just sticking to research,” Network Security, Vol 2002, Issue 10, October 2002, p. 20.

[39] A. Chuakin, “Honeynets: High Value Security Data,” Network Security, Vol. 2003, Issue 8, August 2003, 11-15.

[40] C. Rong, G. Yang, “Honeypots in Balckhat Mode and its Implications,” PDCAT’2003, Proceedings of the Fourth International Conference, August 27-29, 2003, 185-188.

[41] N. Weiler, “Honeypots for Distributed Denial of Service Attacks,” Proceedings of the Eleventh IEEE International Workshop on Enabling Technologies: Infrastructure for collaborative Entrerprises, 2002.

[42] A. Hofmann, C. Schmitz, B. Sick, “Rule Extraction from Neural Networks for Intrusion Detection in Computer Networks,” Proceedings of IEEE International Conference on Systems, Man and Cybernetics, v2, 2003, p 1259 – 1265.

[43] S. Yeldi, S. Gupta, T. Ganacharya, S. Doshi, D. Bahirat, R. Ingle, A. Roychowdhary, “Enhancing Network Intrusion Detection System with Honeypot,” IEEE Conference on Convergent Technologies for the Asia-Pacific Region, Oct 15-17 2003, TENCON v 4, Bangalore, pp 1521-1526.

[44] K. Ilgun, R. Kemmer, P. Porras, “State transition Analysis: A Rule-Based Intrusion Detection Approach,” IEEE Transactions on Software Engineering, v 21, no. 3, March 1995, pp 181-199.

[45] R. Kemmerer, G. Vigna, “Intrusion Detection: A brief History and Overview,” Computer, v 35, n SUPPL, 2002 pp27-30.

[46] R. Bace, P. Mell, “Special Publication on Intrusion Detection Systems,” Tech. Report SP 800-31, National Institute of Standards and Technology, Gaithersburg, Md. Nov 2001.

[47] R. Chinchani, S. Upadhyaya, K. Kwait, “Towards a Scaleable Implementation of a User Level Anomaly Detection System,” IEEE Proceedings on Military Communications Conference MILCOM v2, oct 7-10 2002, California, pp.1503-1508.

[48] J. McGibney, “Intrusion Detection Systems and Honeypots,” http://www.seinit.org/documents/SEINIT_4_INET04_IDS.pdf

[49] www.enseirb.fr/~malfre/dynamic_honeypot

[50] R. Bauman “Honeyd – A low involvement Honeypot in Action,” http://security.rbaumann.net/papers.php?sel=5

[51] L. Spitzner, “Honeypots are they Illegal?” http://www.securityfocus.com/infocus/1703

[52] L. Spizner “Know your enemy: Gen II Honeynets,” http://honeynet.org/papers/index.html

[53] L. Spitzner, “Know your enemy: Honeynets,” http://honeynet.org/papers/index.html

[54] J. Corey, “Local honeypot indentification,” http://phrack.nl/phrack62/p62-0x07.txt

[55] http://www.snort.org

[56] http://www.honeyd.org

[57] J. Hieb, “Towards a Dynamic Honeypot Solution”

[58] Y. Mai, R. Upadrashta, X. Su, “J-Honeypot: A java-based network deception tool with monitoring and intrusion detection”, ITCC v.1, Las Vegas, Nevada, April 5-7 2004.

[59] H. Aljifri, “IP Traceback: A New Denial-of-Service Deterrent?” IEEE Secuirty & Privacy, MAY/JUNE 2003 24-31.

[60] M. Tanase, “Transparent, Bridging and In-line Firewall Devices”, http://www.securityfoucus.com/infocus/1737.

[61] S. Barlas, A. Earls, M. Fitzgerald, J. Ledford, D. McCafferty, “Mission:Critical”, Information Security, September 2004 pg. 26.

[62] L. Gordon, M. Loeb, W. Lucyshyn, R. Richardson, “2004 CSI/FBI Computer Crime and Security Survey”, Computer Security Institute, http://www.gocsi.com.

[63] I. Kuwatly, M. Sraj, Z. Masri, “A Dynamic Honeypot Design for Intrusion Detection,” http://webfea.fea.aub.edu.lb/proceesings/2004/SRC-ECE-04.pdf.

Appendix A

The contents of several tables in the honeypot database indicating various network configurations, and the honeyd configuration file generated by the dynamic honeypot for that network configuration. The membership threshold used to determine host groupings is stated for each test as a percentage of the average distance between the hosts.

Test 1, using 75% of average distance for the threshold:

Honeypot database tables:

```
mysql> select inet_NTOA(ipaddr) as ipaddr,os,count,last from host;
+---------------+-------------------+-------+---------------------+
| ipaddr        | os                | count | last                |
+---------------+-------------------+-------+---------------------+
| 192.168.2.22  | Windows 2000 SP2+ |     3 | 2004-09-19 17:57:25 |
| 192.168.2.62  | Linux 2.4/2.6     |    43 | 2004-09-21 11:14:37 |
| 192.168.2.16  | Linux 2.4/2.6     |   282 | 2004-09-21 11:15:27 |
| 192.168.2.40  | Linux 2.4/2.6     |     5 | 2004-09-20 08:44:56 |
| 192.168.2.202 | AIX 4.3.2         |    25 | 2004-09-21 12:05:23 |
| 192.168.2.200 | AIX 4.3.3-5.2     |    23 | 2004-09-21 11:56:23 |
+---------------+-------------------+-------+---------------------+
6 rows in set (0.00 sec)

mysql> select inet_NTOA(ipaddr) as ipaddr,port from ports;
+---------------+------+
| ipaddr        | port |
+---------------+------+
| 192.168.2.16  |   22 |
| 192.168.2.16  |  443 |
| 192.168.2.16  | 1241 |
| 192.168.2.16  | 3306 |
| 192.168.2.22  |  135 |
| 192.168.2.22  |  139 |
| 192.168.2.22  |  445 |
| 192.168.2.22  | 1025 |
| 192.168.2.23  |  135 |
| 192.168.2.23  |  139 |
| 192.168.2.23  | 1414 |
| 192.168.2.40  |   21 |
| 192.168.2.40  |   22 |
| 192.168.2.40  |   80 |
| 192.168.2.40  |  111 |
| 192.168.2.40  |  443 |
| 192.168.2.62  |   22 |
| 192.168.2.62  |  111 |
| 192.168.2.62  | 3306 |
| 192.168.2.62  | 6000 |
| 192.168.2.200 |   22 |
| 192.168.2.200 |   80 |
| 192.168.2.200 |  443 |
| 192.168.2.202 |   22 |
| 192.168.2.202 |   80 |
| 192.168.2.202 |  443 |
+---------------+------+
26 rows in set (0.02 sec)
```

```
mysql> select hpid,inet_NTOA(ipaddr) as ipaddr from honeyhosts;
+------+---------------+
| hpid | ipaddr        |
+------+---------------+
|   10 | 192.168.2.22  |
|   20 | 192.168.2.40  |
|   20 | 192.168.2.62  |
|   30 | 192.168.2.16  |
|   40 | 192.168.2.200 |
|   40 | 192.168.2.202 |
+------+---------------+
6 rows in set (0.00 sec)

mysql> select hpid,inet_NTOA(ipaddr) as ipaddr,os from honeypots;
+------+---------------+----------------------------------+
| hpid | ipaddr        | os                               |
+------+---------------+----------------------------------+
|   10 | 192.168.2.23  | Windows 2000 SP2                 |
|   20 | 192.168.2.32  | Linux 2.4.16 - 2.4.18            |
|   30 | 192.168.2.17  | Linux 2.4.16 - 2.4.18            |
|   40 | 192.168.2.194 | AIX 4.3.2.0-4.3.3.0 on an IBM RS/* |
+------+---------------+----------------------------------+
4 rows in set (0.00 sec)

mysql> select hpid,port,proxy from honeyports;
+------+------+---------------+
| hpid | port | proxy         |
+------+------+---------------+
|   10 |  135 | 192.168.2.22  |
|   10 |  139 | 192.168.2.22  |
|   10 |  445 | 192.168.2.22  |
|   10 | 1025 | 192.168.2.22  |
|   20 |   21 | 192.168.2.40  |
|   20 |   22 | 192.168.2.62  |
|   20 |   80 | 192.168.2.40  |
|   20 |  111 | 192.168.2.62  |
|   20 |  443 | 192.168.2.40  |
|   20 | 3306 | 192.168.2.62  |
|   20 | 6000 | 192.168.2.62  |
|   30 |   22 | 192.168.2.16  |
|   30 |  443 | 192.168.2.16  |
|   30 | 1241 | 192.168.2.16  |
|   30 | 3306 | 192.168.2.16  |
|   40 |   22 | 192.168.2.202 |
|   40 |   80 | 192.168.2.202 |
|   40 |  443 | 192.168.2.202 |
+------+------+---------------+
18 rows in set (0.00 sec)
```

Honeyd configuration file used by the dynamic honeypot.

```
create default
set default personality "Windows NT4 / Win95 / Win98"
set default default tcp action block
set default default udp action block
set default default icmp action block

create honeypot10intern
set honeypot10intern personality "Windows 2000 SP2"
set honeypot10intern default tcp action block
set honeypot10intern default udp action block
set honeypot10intern default icmp action open
add honeypot10intern tcp port 135 proxy 192.168.2.22:135
add honeypot10intern tcp port 139 proxy 192.168.2.22:139
add honeypot10intern tcp port 445 proxy 192.168.2.22:445
add honeypot10intern tcp port 1025 proxy 192.168.2.22:1025
create honeypot10extern
set honeypot10extern personality "Windows 2000 SP2"
set honeypot10extern default tcp action block
```

```
set honeypot10extern default udp action block
set honeypot10extern default icmp action open
add honeypot10extern tcp port 135 open
add honeypot10extern tcp port 139 open
add honeypot10extern tcp port 445 open
add honeypot10extern tcp port 1025 open
dynamic honeypot10
add honeypot10 use honeypot10intern if source ip = 192.168.2.0/24
add honeypot10 otherwise use honeypot10extern
bind 192.168.2.23 honeypot10
create honeypot20intern
set honeypot20intern personality "Linux 2.4.16 - 2.4.18"
set honeypot20intern default tcp action block
set honeypot20intern default udp action block
set honeypot20intern default icmp action open
add honeypot20intern tcp port 21 proxy 192.168.2.40:21
add honeypot20intern tcp port 22 proxy 192.168.2.62:22
add honeypot20intern tcp port 80 proxy 192.168.2.40:80
add honeypot20intern tcp port 111 proxy 192.168.2.62:111
add honeypot20intern tcp port 443 proxy 192.168.2.40:443
add honeypot20intern tcp port 3306 proxy 192.168.2.62:3306
add honeypot20intern tcp port 6000 proxy 192.168.2.62:6000
create honeypot20extern
set honeypot20extern personality "Linux 2.4.16 - 2.4.18"
set honeypot20extern default tcp action block
set honeypot20extern default udp action block
set honeypot20extern default icmp action open
add honeypot20extern tcp port 21 "./scripts/ftp.sh $ipsrc $sport"
add honeypot20extern tcp port 22 "./scripts/ssh.sh $ipsrc $sport"
add honeypot20extern tcp port 80 "./scripts/iis.sh $ipsrc $sport"
add honeypot20extern tcp port 111 open
add honeypot20extern tcp port 443 open
add honeypot20extern tcp port 3306 "./scripts/mysql.sh $ipsrc $sport"
add honeypot20extern tcp port 6000 open
dynamic honeypot20
add honeypot20 use honeypot20intern if source ip = 192.168.2.0/24
add honeypot20 otherwise use honeypot20extern
bind 192.168.2.32 honeypot20
create honeypot30intern
set honeypot30intern personality "Linux 2.4.16 - 2.4.18"
set honeypot30intern default tcp action block
set honeypot30intern default udp action block
set honeypot30intern default icmp action open
add honeypot30intern tcp port 22 proxy 192.168.2.16:22
add honeypot30intern tcp port 443 proxy 192.168.2.16:443
add honeypot30intern tcp port 1241 proxy 192.168.2.16:1241
add honeypot30intern tcp port 3306 proxy 192.168.2.16:3306
create honeypot30extern
set honeypot30extern personality "Linux 2.4.16 - 2.4.18"
set honeypot30extern default tcp action block
set honeypot30extern default udp action block
set honeypot30extern default icmp action open
add honeypot30extern tcp port 22 "./scripts/ssh.sh $ipsrc $sport"
add honeypot30extern tcp port 443 open
add honeypot30extern tcp port 1241 open
add honeypot30extern tcp port 3306 "./scripts/mysql.sh $ipsrc $sport"
dynamic honeypot30
add honeypot30 use honeypot30intern if source ip = 192.168.2.0/24
add honeypot30 otherwise use honeypot30extern
bind 192.168.2.17 honeypot30
create honeypot40intern
set honeypot40intern personality "AIX 4.3.3.0 on an IBM RS/*"
set honeypot40intern default tcp action block
set honeypot40intern default udp action block
set honeypot40intern default icmp action open
create honeypot40extern
set honeypot40extern personality "AIX 4.3.3.0 on an IBM RS/*"
set honeypot40extern default tcp action block
set honeypot40extern default udp action block
set honeypot40extern default icmp action open
dynamic honeypot40
```

```
add honeypot40 use honeypot40intern if source ip = 192.168.2.0/24
add honeypot40 otherwise use honeypot40extern
bind 192.168.2.201 honeypot40
```

Test 2, using 75% of average distance for threshold:

Honeypot database tables:

```
mysql> select inet_NTOA(ipaddr) as ipaddr,os,count,last from host;
+---------------+-------------------+-------+---------------------+
| ipaddr        | os                | count | last                |
+---------------+-------------------+-------+---------------------+
| 192.168.2.22  | Windows 2000 SP2+ |     3 | 2004-09-19 17:57:25 |
| 192.168.2.62  | Linux 2.4/2.6     |    43 | 2004-09-21 11:14:37 |
| 192.168.2.16  | Linux 2.4/2.6     |   282 | 2004-09-21 11:15:27 |
| 192.168.2.40  | Linux 2.4/2.6     |     5 | 2004-09-20 08:44:56 |
| 192.168.2.202 | AIX 4.3.2         |    25 | 2004-09-21 12:05:23 |
| 192.168.2.200 | AIX 4.3.3-5.2     |    23 | 2004-09-21 11:56:23 |
| 192.168.2.23  | Windows XP Pro SP1 |   22 | 2004-09-21 12:40:18 |
+---------------+-------------------+-------+---------------------+
7 rows in set (0.00 sec)

mysql> select inet_NTOA(ipaddr) as ipaddr,port from ports;
+---------------+------+
| ipaddr        | port |
+---------------+------+
| 192.168.2.16  |   22 |
| 192.168.2.16  |  443 |
| 192.168.2.16  | 1241 |
| 192.168.2.16  | 3306 |
| 192.168.2.22  |  135 |
| 192.168.2.22  |  139 |
| 192.168.2.22  |  445 |
| 192.168.2.22  | 1025 |
| 192.168.2.23  |  135 |
| 192.168.2.23  |  139 |
| 192.168.2.23  | 1414 |
| 192.168.2.40  |   21 |
| 192.168.2.40  |   22 |
| 192.168.2.40  |   80 |
| 192.168.2.40  |  111 |
| 192.168.2.40  |  443 |
| 192.168.2.62  |   22 |
| 192.168.2.62  |  111 |
| 192.168.2.62  | 3306 |
| 192.168.2.62  | 6000 |
| 192.168.2.200 |   22 |
| 192.168.2.200 |   80 |
| 192.168.2.200 |  443 |
| 192.168.2.202 |   22 |
| 192.168.2.202 |   80 |
| 192.168.2.202 |  443 |
+---------------+------+
26 rows in set (0.01 sec)

mysql> select hpid,inet_NTOA(ipaddr) as ipaddr from honeyhosts;
+------+---------------+
| hpid | ipaddr        |
+------+---------------+
|   10 | 192.168.2.22  |
|   10 | 192.168.2.23  |
|   20 | 192.168.2.40  |
|   20 | 192.168.2.62  |
|   30 | 192.168.2.16  |
|   40 | 192.168.2.200 |
|   40 | 192.168.2.202 |
+------+---------------+
7 rows in set (0.00 sec)
```

```
mysql> select hpid,inet_NTOA(ipaddr) as ipaddr,os from honeypots;
+------+---------------+----------------------------------------------------+
| hpid | ipaddr        | os                                                 |
+------+---------------+----------------------------------------------------+
|   10 | 192.168.2.20  | Windows XP Professional RC1+ through final release |
|   20 | 192.168.2.32  | Linux 2.4.16 - 2.4.18                              |
|   30 | 192.168.2.17  | Linux 2.4.16 - 2.4.18                              |
|   40 | 192.168.2.194 | AIX 4.3.2.0-4.3.3.0 on an IBM RS/*                 |
+------+---------------+----------------------------------------------------+
4 rows in set (0.00 sec)

mysql> select hpid,port,proxy from honeyports;
+------+------+---------------+
| hpid | port | proxy         |
+------+------+---------------+
|   10 |  135 | 192.168.2.23  |
|   10 |  139 | 192.168.2.23  |
|   10 |  445 | 192.168.2.22  |
|   10 | 1025 | 192.168.2.22  |
|   10 | 1414 | 192.168.2.23  |
|   20 |   21 | 192.168.2.40  |
|   20 |   22 | 192.168.2.62  |
|   20 |   80 | 192.168.2.40  |
|   20 |  111 | 192.168.2.62  |
|   20 |  443 | 192.168.2.40  |
|   20 | 3306 | 192.168.2.62  |
|   20 | 6000 | 192.168.2.62  |
|   30 |   22 | 192.168.2.16  |
|   30 |  443 | 192.168.2.16  |
|   30 | 1241 | 192.168.2.16  |
|   30 | 3306 | 192.168.2.16  |
|   40 |   22 | 192.168.2.202 |
|   40 |   80 | 192.168.2.202 |
|   40 |  443 | 192.168.2.202 |
+------+------+---------------+
19 rows in set (0.01 sec)
```

Honeyd configuration file used by the dynamic honeypot

```
create default
set default personality "Windows NT4 / Win95 / Win98"
set default default tcp action block
set default default udp action block
set default default icmp action block

create honeypot10intern
set honeypot10intern personality "Windows XP Professional RC1+ through final release"
set honeypot10intern default tcp action block
set honeypot10intern default udp action block
set honeypot10intern default icmp action open
add honeypot10intern tcp port 135 proxy 192.168.2.23:135
add honeypot10intern tcp port 139 proxy 192.168.2.23:139
add honeypot10intern tcp port 445 proxy 192.168.2.22:445
add honeypot10intern tcp port 1025 proxy 192.168.2.22:1025
add honeypot10intern tcp port 1414 proxy 192.168.2.23:1414
create honeypot10extern
set honeypot10extern personality "Windows XP Professional RC1+ through final release"
set honeypot10extern default tcp action block
set honeypot10extern default udp action block
set honeypot10extern default icmp action open
add honeypot10extern tcp port 135 open
add honeypot10extern tcp port 139 open
add honeypot10extern tcp port 445 open
add honeypot10extern tcp port 1025 open
add honeypot10extern tcp port 1414 open
dynamic honeypot10
add honeypot10 use honeypot10intern if source ip = 192.168.2.0/24
add honeypot10 otherwise use honeypot10extern
bind 192.168.2.20 honeypot10
create honeypot20intern
```

```
set honeypot20intern personality "Linux 2.4.16 - 2.4.18"
set honeypot20intern default tcp action block
set honeypot20intern default udp action block
set honeypot20intern default icmp action open
add honeypot20intern tcp port 21 proxy 192.168.2.40:21
add honeypot20intern tcp port 22 proxy 192.168.2.62:22
add honeypot20intern tcp port 80 proxy 192.168.2.40:80
add honeypot20intern tcp port 111 proxy 192.168.2.62:111
add honeypot20intern tcp port 443 proxy 192.168.2.40:443
add honeypot20intern tcp port 3306 proxy 192.168.2.62:3306
add honeypot20intern tcp port 6000 proxy 192.168.2.62:6000
create honeypot20extern
set honeypot20extern personality "Linux 2.4.16 - 2.4.18"
set honeypot20extern default tcp action block
set honeypot20extern default udp action block
set honeypot20extern default icmp action open
add honeypot20extern tcp port 21 "./scripts/ftp.sh $ipsrc $sport"
add honeypot20extern tcp port 22 "./scripts/ssh.sh $ipsrc $sport"
add honeypot20extern tcp port 80 "./scripts/iis.sh $ipsrc $sport"
add honeypot20extern tcp port 111 open
add honeypot20extern tcp port 443 open
add honeypot20extern tcp port 3306 "./scripts/mysql.sh $ipsrc $sport"
add honeypot20extern tcp port 6000 open
dynamic honeypot20
add honeypot20 use honeypot20intern if source ip = 192.168.2.0/24
add honeypot20 otherwise use honeypot20extern
bind 192.168.2.32 honeypot20
create honeypot30intern
set honeypot30intern personality "Linux 2.4.16 - 2.4.18"
set honeypot30intern default tcp action block
set honeypot30intern default udp action block
set honeypot30intern default icmp action open
add honeypot30intern tcp port 22 proxy 192.168.2.16:22
add honeypot30intern tcp port 443 proxy 192.168.2.16:443
add honeypot30intern tcp port 1241 proxy 192.168.2.16:1241
add honeypot30intern tcp port 3306 proxy 192.168.2.16:3306
create honeypot30extern
set honeypot30extern personality "Linux 2.4.16 - 2.4.18"
set honeypot30extern default tcp action block
set honeypot30extern default udp action block
set honeypot30extern default icmp action open
add honeypot30extern tcp port 22 "./scripts/ssh.sh $ipsrc $sport"
add honeypot30extern tcp port 443 open
add honeypot30extern tcp port 1241 open
add honeypot30extern tcp port 3306 "./scripts/mysql.sh $ipsrc $sport"
dynamic honeypot30
add honeypot30 use honeypot30intern if source ip = 192.168.2.0/24
add honeypot30 otherwise use honeypot30extern
bind 192.168.2.17 honeypot30
create honeypot40intern
set honeypot40intern personality "AIX 4.3.2.0-4.3.3.0 on an IBM RS/*"
set honeypot40intern default tcp action block
set honeypot40intern default udp action block
set honeypot40intern default icmp action open
add honeypot40intern tcp port 22 proxy 192.168.2.202:22
add honeypot40intern tcp port 80 proxy 192.168.2.202:80
add honeypot40intern tcp port 443 proxy 192.168.2.202:443
create honeypot40extern
set honeypot40extern personality "AIX 4.3.2.0-4.3.3.0 on an IBM RS/*"
set honeypot40extern default tcp action block
set honeypot40extern default udp action block
set honeypot40extern default icmp action open
add honeypot40extern tcp port 22 "./scripts/ssh.sh $ipsrc $sport"
add honeypot40extern tcp port 80 "./scripts/iis.sh $ipsrc $sport"
add honeypot40extern tcp port 443 open
dynamic honeypot40
add honeypot40 use honeypot40intern if source ip = 192.168.2.0/24
add honeypot40 otherwise use honeypot40extern
bind 192.168.2.194 honeypot40
```

Test 3 using 99% of average distance for threshold:

Honeypot database tables:

```
mysql> select inet_NTOA(ipaddr) as ipaddr,os,count,last from host;
+---------------+--------------------+-------+---------------------+
| ipaddr        | os                 | count | last                |
+---------------+--------------------+-------+---------------------+
| 192.168.2.22  | Windows 2000 SP2+  |     3 | 2004-09-19 17:57:25 |
| 192.168.2.62  | Linux 2.4/2.6      |    44 | 2004-09-21 13:14:40 |
| 192.168.2.16  | Linux 2.4/2.6      |   282 | 2004-09-21 11:15:27 |
| 192.168.2.40  | Linux 2.4/2.6      |     5 | 2004-09-20 08:44:56 |
| 192.168.2.202 | AIX 4.3.2          |    25 | 2004-09-21 12:05:23 |
| 192.168.2.200 | AIX 4.3.3-5.2      |    23 | 2004-09-21 11:56:23 |
| 192.168.2.23  | Windows XP Pro SP1 |    22 | 2004-09-21 12:40:18 |
+---------------+--------------------+-------+---------------------+
7 rows in set (0.00 sec)

mysql> select inet_NTOA(ipaddr) as ipaddr,port from ports;
+---------------+------+
| ipaddr        | port |
+---------------+------+
| 192.168.2.16  |   22 |
| 192.168.2.16  |  443 |
| 192.168.2.16  | 1241 |
| 192.168.2.16  | 3306 |
| 192.168.2.22  |  135 |
| 192.168.2.22  |  139 |
| 192.168.2.22  |  445 |
| 192.168.2.22  | 1025 |
| 192.168.2.23  |  135 |
| 192.168.2.23  |  139 |
| 192.168.2.23  | 1414 |
| 192.168.2.40  |   21 |
| 192.168.2.40  |   22 |
| 192.168.2.40  |   80 |
| 192.168.2.40  |  111 |
| 192.168.2.40  |  443 |
| 192.168.2.62  |   22 |
| 192.168.2.62  |  111 |
| 192.168.2.62  | 3306 |
| 192.168.2.62  | 6000 |
| 192.168.2.200 |   22 |
| 192.168.2.200 |   80 |
| 192.168.2.200 |  443 |
| 192.168.2.202 |   22 |
| 192.168.2.202 |   80 |
| 192.168.2.202 |  443 |
+---------------+------+
26 rows in set (0.00 sec)

mysql> select hpid,inet_NTOA(ipaddr) as ipaddr from honeyhosts;
+------+---------------+
| hpid | ipaddr        |
+------+---------------+
|   10 | 192.168.2.16  |
|   10 | 192.168.2.40  |
|   10 | 192.168.2.62  |
|   20 | 192.168.2.22  |
|   20 | 192.168.2.23  |
|   30 | 192.168.2.200 |
|   30 | 192.168.2.202 |
+------+---------------+
7 rows in set (0.00 sec)

mysql> select hpid,inet_NTOA(ipaddr) as ipaddr,os from honeypots;
+------+---------------+----------------------------------------------------+
| hpid | ipaddr        | os                                                 |
+------+---------------+----------------------------------------------------+
```

```
|   10 | 192.168.2.46  | Linux 2.4.16 - 2.4.18                              |
|   20 | 192.168.2.21  | Windows XP Professional RC1+ through final release |
|   30 | 192.168.2.194 | AIX 4.3.2.0-4.3.3.0 on an IBM RS/*                 |
+------+---------------+----------------------------------------------------+
3 rows in set (0.00 sec)

mysql> select hpid,port,proxy from honeyports;
+------+------+---------------+
| hpid | port | proxy         |
+------+------+---------------+
|   10 |   22 | 192.168.2.62  |
|   10 |  443 | 192.168.2.40  |
|   10 | 1241 | 192.168.2.16  |
|   10 | 3306 | 192.168.2.62  |
|   10 |   21 | 192.168.2.40  |
|   10 |   80 | 192.168.2.40  |
|   10 |  111 | 192.168.2.62  |
|   10 | 6000 | 192.168.2.62  |
|   20 |  135 | 192.168.2.23  |
|   20 |  139 | 192.168.2.23  |
|   20 |  445 | 192.168.2.22  |
|   20 | 1025 | 192.168.2.22  |
|   20 | 1414 | 192.168.2.23  |
|   30 |   22 | 192.168.2.202 |
|   30 |   80 | 192.168.2.202 |
|   30 |  443 | 192.168.2.202 |
+------+------+---------------+
16 rows in set (0.01 sec)
```

Honeyd configuration file used by the dynamic honeypot.

```
create default
set default personality "Windows NT4 / Win95 / Win98"
set default default tcp action block
set default default udp action block
set default default icmp action block

create honeypot10intern
set honeypot10intern personality "Linux 2.4.16 - 2.4.18"
set honeypot10intern default tcp action block
set honeypot10intern default udp action block
set honeypot10intern default icmp action open
add honeypot10intern tcp port 21 proxy 192.168.2.40:21
add honeypot10intern tcp port 22 proxy 192.168.2.62:22
add honeypot10intern tcp port 80 proxy 192.168.2.40:80
add honeypot10intern tcp port 111 proxy 192.168.2.62:111
add honeypot10intern tcp port 443 proxy 192.168.2.40:443
add honeypot10intern tcp port 1241 proxy 192.168.2.16:1241
add honeypot10intern tcp port 3306 proxy 192.168.2.62:3306
add honeypot10intern tcp port 6000 proxy 192.168.2.62:6000
create honeypot10extern
set honeypot10extern personality "Linux 2.4.16 - 2.4.18"
set honeypot10extern default tcp action block
set honeypot10extern default udp action block
set honeypot10extern default icmp action open
add honeypot10extern tcp port 21 "./scripts/ftp.sh $ipsrc $sport"
add honeypot10extern tcp port 22 "./scripts/ssh.sh $ipsrc $sport"
add honeypot10extern tcp port 80 "./scripts/iis.sh $ipsrc $sport"
add honeypot10extern tcp port 111 open
add honeypot10extern tcp port 443 open
add honeypot10extern tcp port 1241 open
add honeypot10extern tcp port 3306 "./scripts/mysql.sh $ipsrc $sport"
add honeypot10extern tcp port 6000 open
dynamic honeypot10
add honeypot10 use honeypot10intern if source ip = 192.168.2.0/24
add honeypot10 otherwise use honeypot10extern
bind 192.168.2.46 honeypot10
create honeypot20intern
set honeypot20intern personality "Windows XP Professional RC1+ through final release"
set honeypot20intern default tcp action block
```

```
set honeypot20intern default udp action block
set honeypot20intern default icmp action open
add honeypot20intern tcp port 135 proxy 192.168.2.23:135
add honeypot20intern tcp port 139 proxy 192.168.2.23:139
add honeypot20intern tcp port 445 proxy 192.168.2.22:445
add honeypot20intern tcp port 1025 proxy 192.168.2.22:1025
add honeypot20intern tcp port 1414 proxy 192.168.2.23:1414
create honeypot20extern
set honeypot20extern personality "Windows XP Professional RC1+ through final release"
set honeypot20extern default tcp action block
set honeypot20extern default udp action block
set honeypot20extern default icmp action open
add honeypot20extern tcp port 135 open
add honeypot20extern tcp port 139 open
add honeypot20extern tcp port 445 open
add honeypot20extern tcp port 1025 open
add honeypot20extern tcp port 1414 open
dynamic honeypot20
add honeypot20 use honeypot20intern if source ip = 192.168.2.0/24
add honeypot20 otherwise use honeypot20extern
bind 192.168.2.21 honeypot20
create honeypot30intern
set honeypot30intern personality "AIX 4.3.2.0-4.3.3.0 on an IBM RS/*"
set honeypot30intern default tcp action block
set honeypot30intern default udp action block
set honeypot30intern default icmp action open
add honeypot30intern tcp port 22 proxy 192.168.2.202:22
add honeypot30intern tcp port 80 proxy 192.168.2.202:80
add honeypot30intern tcp port 443 proxy 192.168.2.202:443
create honeypot30extern
set honeypot30extern personality "AIX 4.3.2.0-4.3.3.0 on an IBM RS/*"
set honeypot30extern default tcp action block
set honeypot30extern default udp action block
set honeypot30extern default icmp action open
add honeypot30extern tcp port 22 "./scripts/ssh.sh $ipsrc $sport"
add honeypot30extern tcp port 80 "./scripts/iis.sh $ipsrc $sport"
add honeypot30extern tcp port 443 open
dynamic honeypot30
add honeypot30 use honeypot30intern if source ip = 192.168.2.0/24
add honeypot30 otherwise use honeypot30extern
bind 192.168.2.194 honeypot30
```

Test 4, using 75% of the average distance as the threshold:

Honeypot database tables:

```
mysql> select inet_NTOA(ipaddr) as ipaddr,os,count,last from host;
+---------------+--------------------+-------+---------------------+
| ipaddr        | os                 | count | last                |
+---------------+--------------------+-------+---------------------+
| 192.168.2.22  | Windows 2000 SP2+  |     3 | 2004-09-19 17:57:25 |
| 192.168.2.62  | Linux 2.4/2.6      |    44 | 2004-09-21 13:14:40 |
| 192.168.2.16  | Linux 2.4/2.6      |   282 | 2004-09-21 11:15:27 |
| 192.168.2.40  | Linux 2.4/2.6      |     5 | 2004-09-20 08:44:56 |
| 192.168.2.202 | AIX 4.3.2          |    25 | 2004-09-21 12:05:23 |
| 192.168.2.200 | AIX 4.3.3-5.2      |    23 | 2004-09-21 11:56:23 |
| 192.168.2.23  | Windows XP Pro SP1 |    22 | 2004-09-21 12:40:18 |
| 192.168.3.100 | OpenBSD 3.3-3.4    |    22 | 2004-09-21 13:39:18 |
| 192.168.3.102 | OpenBSD 3.3-3.4    |    22 | 2004-09-21 13:39:28 |
| 192.168.3.104 | OpenBSD 3.3-3.4    |    22 | 2004-09-21 13:39:32 |
+---------------+--------------------+-------+---------------------+
10 rows in set (0.00 sec)

mysql> select inet_NTOA(ipaddr) as ipaddr,port from ports;
+---------------+------+
| ipaddr        | port |
+---------------+------+
| 192.168.2.16  |   22 |
| 192.168.2.16  |  443 |
```

```
| 192.168.2.16  | 1241 |
| 192.168.2.16  | 3306 |
| 192.168.2.22  |  135 |
| 192.168.2.22  |  139 |
| 192.168.2.22  |  445 |
| 192.168.2.22  | 1025 |
| 192.168.2.23  |  135 |
| 192.168.2.23  |  139 |
| 192.168.2.23  | 1414 |
| 192.168.2.40  |   21 |
| 192.168.2.40  |   22 |
| 192.168.2.40  |   80 |
| 192.168.2.40  |  111 |
| 192.168.2.40  |  443 |
| 192.168.2.62  |   22 |
| 192.168.2.62  |  111 |
| 192.168.2.62  | 3306 |
| 192.168.2.62  | 6000 |
| 192.168.2.200 |   22 |
| 192.168.2.200 |   80 |
| 192.168.2.200 |  443 |
| 192.168.2.202 |   22 |
| 192.168.2.202 |   80 |
| 192.168.2.202 |  443 |
| 192.168.3.100 |   21 |
| 192.168.3.100 |   80 |
| 192.168.3.102 |   21 |
| 192.168.3.102 |   80 |
| 192.168.3.104 |   21 |
| 192.168.3.104 |   80 |
+---------------+------+
32 rows in set (0.01 sec)

mysql> select hpid,inet_NTOA(ipaddr) as ipaddr from honeyhosts;
+------+---------------+
| hpid | ipaddr        |
+------+---------------+
|   10 | 192.168.2.16  |
|   10 | 192.168.2.40  |
|   10 | 192.168.2.62  |
|   20 | 192.168.2.22  |
|   20 | 192.168.2.23  |
|   30 | 192.168.2.200 |
|   30 | 192.168.2.202 |
|   40 | 192.168.3.100 |
|   40 | 192.168.3.102 |
|   40 | 192.168.3.104 |
+------+---------------+
10 rows in set (0.00 sec)

mysql> select hpid,inet_NTOA(ipaddr) as ipaddr,os from honeypots;
+------+---------------+----------------------------------------------------+
| hpid | ipaddr        | os                                                 |
+------+---------------+----------------------------------------------------+
|   10 | 192.168.2.46  | Linux 2.4.16 - 2.4.18                              |
|   20 | 192.168.2.21  | Windows XP Professional RC1+ through final release |
|   30 | 192.168.2.194 | AIX 4.3.2.0-4.3.3.0 on an IBM RS/*                 |
|   40 | 192.168.3.108 | OpenBSD 3.0 (x86 or SPARC)                         |
+------+---------------+----------------------------------------------------+
4 rows in set (0.00 sec)

mysql> select hpid,port,proxy from honeyports;
+------+------+---------------+
| hpid | port | proxy         |
+------+------+---------------+
|   10 |   22 | 192.168.2.62  |
|   10 |  443 | 192.168.2.40  |
|   10 | 1241 | 192.168.2.16  |
|   10 | 3306 | 192.168.2.62  |
|   10 |   21 | 192.168.2.40  |
|   10 |   80 | 192.168.2.40  |
```

```
|   10 |  111 | 192.168.2.62  |
|   10 | 6000 | 192.168.2.62  |
|   20 |  135 | 192.168.2.23  |
|   20 |  139 | 192.168.2.23  |
|   20 |  445 | 192.168.2.22  |
|   20 | 1025 | 192.168.2.22  |
|   20 | 1414 | 192.168.2.23  |
|   30 |   22 | 192.168.2.202 |
|   30 |   80 | 192.168.2.202 |
|   30 |  443 | 192.168.2.202 |
|   40 |   21 | 192.168.3.104 |
|   40 |   80 | 192.168.3.104 |
+------+------+---------------+
18 rows in set (0.01 sec)
```

Honeyd configuration file used by the dynamic honeypot.

```
mysql> notee
create default
set default personality "Windows NT4 / Win95 / Win98"
set default default tcp action block
set default default udp action block
set default default icmp action block

create honeypot10intern
set honeypot10intern personality "Linux 2.4.16 - 2.4.18"
set honeypot10intern default tcp action block
set honeypot10intern default udp action block
set honeypot10intern default icmp action open
add honeypot10intern tcp port 21 proxy 192.168.2.40:21
add honeypot10intern tcp port 22 proxy 192.168.2.62:22
add honeypot10intern tcp port 80 proxy 192.168.2.40:80
add honeypot10intern tcp port 111 proxy 192.168.2.62:111
add honeypot10intern tcp port 443 proxy 192.168.2.40:443
add honeypot10intern tcp port 1241 proxy 192.168.2.16:1241
add honeypot10intern tcp port 3306 proxy 192.168.2.62:3306
add honeypot10intern tcp port 6000 proxy 192.168.2.62:6000
create honeypot10extern
set honeypot10extern personality "Linux 2.4.16 - 2.4.18"
set honeypot10extern default tcp action block
set honeypot10extern default udp action block
set honeypot10extern default icmp action open
add honeypot10extern tcp port 21 "./scripts/ftp.sh $ipsrc $sport"
add honeypot10extern tcp port 22 "./scripts/ssh.sh $ipsrc $sport"
add honeypot10extern tcp port 80 "./scripts/iis.sh $ipsrc $sport"
add honeypot10extern tcp port 111 open
add honeypot10extern tcp port 443 open
add honeypot10extern tcp port 1241 open
add honeypot10extern tcp port 3306 "./scripts/mysql.sh $ipsrc $sport"
add honeypot10extern tcp port 6000 open
dynamic honeypot10
add honeypot10 use honeypot10intern if source ip = 192.168.2.0/23
add honeypot10 otherwise use honeypot10extern
bind 192.168.2.46 honeypot10
create honeypot20intern
set honeypot20intern personality "Windows XP Professional RC1+ through final release"
set honeypot20intern default tcp action block
set honeypot20intern default udp action block
set honeypot20intern default icmp action open
add honeypot20intern tcp port 135 proxy 192.168.2.23:135
add honeypot20intern tcp port 139 proxy 192.168.2.23:139
add honeypot20intern tcp port 445 proxy 192.168.2.22:445
add honeypot20intern tcp port 1025 proxy 192.168.2.22:1025
add honeypot20intern tcp port 1414 proxy 192.168.2.23:1414
create honeypot20extern
set honeypot20extern personality "Windows XP Professional RC1+ through final release"
set honeypot20extern default tcp action block
set honeypot20extern default udp action block
set honeypot20extern default icmp action open
add honeypot20extern tcp port 135 open
```

```
add honeypot20extern tcp port 139 open
add honeypot20extern tcp port 445 open
add honeypot20extern tcp port 1025 open
add honeypot20extern tcp port 1414 open
dynamic honeypot20
add honeypot20 use honeypot20intern if source ip = 192.168.2.0/23
add honeypot20 otherwise use honeypot20extern
bind 192.168.2.21 honeypot20
create honeypot30intern
set honeypot30intern personality "AIX 4.3.2.0-4.3.3.0 on an IBM RS/*"
set honeypot30intern default tcp action block
set honeypot30intern default udp action block
set honeypot30intern default icmp action open
add honeypot30intern tcp port 22 proxy 192.168.2.202:22
add honeypot30intern tcp port 80 proxy 192.168.2.202:80
add honeypot30intern tcp port 443 proxy 192.168.2.202:443
create honeypot30extern
set honeypot30extern personality "AIX 4.3.2.0-4.3.3.0 on an IBM RS/*"
set honeypot30extern default tcp action block
set honeypot30extern default udp action block
set honeypot30extern default icmp action open
add honeypot30extern tcp port 22 "./scripts/ssh.sh $ipsrc $sport"
add honeypot30extern tcp port 80 "./scripts/iis.sh $ipsrc $sport"
add honeypot30extern tcp port 443 open
dynamic honeypot30
add honeypot30 use honeypot30intern if source ip = 192.168.2.0/23
add honeypot30 otherwise use honeypot30extern
bind 192.168.2.194 honeypot30
create honeypot40intern
set honeypot40intern personality "OpenBSD 3.0 (x86 or SPARC)"
set honeypot40intern default tcp action block
set honeypot40intern default udp action block
set honeypot40intern default icmp action open
add honeypot40intern tcp port 21 proxy 192.168.3.104:21
add honeypot40intern tcp port 80 proxy 192.168.3.104:80
create honeypot40extern
set honeypot40extern personality "OpenBSD 3.0 (x86 or SPARC)"
set honeypot40extern default tcp action block
set honeypot40extern default udp action block
set honeypot40extern default icmp action open
add honeypot40extern tcp port 21 "./scripts/ftp.sh $ipsrc $sport"
add honeypot40extern tcp port 80 "./scripts/iis.sh $ipsrc $sport"
dynamic honeypot40
add honeypot40 use honeypot40intern if source ip = 192.168.2.0/23
add honeypot40 otherwise use honeypot40extern
bind 192.168.3.108 honeypot40
```

Appendix B

A honeyd configuration file generated during testing.

```
create default
set default personality "Windows NT4 / Win95 / Win98"
set default default tcp action block
set default default udp action block
set default default icmp action block

create honeypot10intern
set honeypot10intern personality "Linux 2.4.16 - 2.4.18"
set honeypot10intern default tcp action block
set honeypot10intern default udp action block
set honeypot10intern default icmp action open
add honeypot10intern tcp port 22 proxy 192.168.2.16:22
add honeypot10intern tcp port 111 proxy 192.168.2.16:111
add honeypot10intern tcp port 443 proxy 192.168.2.16:443
add honeypot10intern tcp port 1241 proxy 192.168.2.16:1241
add honeypot10intern tcp port 3306 proxy 192.168.2.16:3306
add honeypot10intern tcp port 32767 proxy 192.168.2.16:32767
create honeypot10extern
set honeypot10extern personality "Linux 2.4.16 - 2.4.18"
set honeypot10extern default tcp action block
set honeypot10extern default udp action block
set honeypot10extern default icmp action open
add honeypot10extern tcp port 22 "./scripts/ssh.sh $ipsrc $sport"
add honeypot10extern tcp port 111 open
add honeypot10extern tcp port 443 open
add honeypot10extern tcp port 1241 open
add honeypot10extern tcp port 3306 "./scripts/mysql.sh $ipsrc $sport"
add honeypot10extern tcp port 32767 open
dynamic honeypot10
add honeypot10 use honeypot10intern if source ip = 192.168.2.0/26
add honeypot10 otherwise use honeypot10extern
bind 192.168.2.17 honeypot10
create honeypot20intern
set honeypot20intern personality "Linux 1.0.9"
set honeypot20intern default tcp action block
set honeypot20intern default udp action block
set honeypot20intern default icmp action open
add honeypot20intern tcp port 21 proxy 192.168.2.40:21
add honeypot20intern tcp port 22 proxy 192.168.2.40:22
add honeypot20intern tcp port 80 proxy 192.168.2.40:80
add honeypot20intern tcp port 111 proxy 192.168.2.40:111
add honeypot20intern tcp port 443 proxy 192.168.2.40:443
add honeypot20intern tcp port 28839 proxy 192.168.2.40:28839
add honeypot20intern tcp port 32767 proxy 192.168.2.40:32767
create honeypot20extern
set honeypot20extern personality "Linux 1.0.9"
set honeypot20extern default tcp action block
set honeypot20extern default udp action block
set honeypot20extern default icmp action open
add honeypot20extern tcp port 21 "./scripts/ftp.sh $ipsrc $sport"
add honeypot20extern tcp port 22 "./scripts/ssh.sh $ipsrc $sport"
add honeypot20extern tcp port 80 "./scripts/iis.sh $ipsrc $sport"
add honeypot20extern tcp port 111 open
add honeypot20extern tcp port 443 open
add honeypot20extern tcp port 28839 open
add honeypot20extern tcp port 32767 open
dynamic honeypot20
add honeypot20 use honeypot20intern if source ip = 192.168.2.0/26
add honeypot20 otherwise use honeypot20extern
bind 192.168.2.41 honeypot20
create honeypot30intern
set honeypot30intern personality "Windows 2000 Professional"
set honeypot30intern default tcp action block
```

```
set honeypot30intern default udp action block
set honeypot30intern default icmp action open
add honeypot30intern tcp port 135 proxy 192.168.2.46:135
add honeypot30intern tcp port 139 proxy 192.168.2.46:139
add honeypot30intern tcp port 445 proxy 192.168.2.46:445
add honeypot30intern tcp port 666 proxy 192.168.2.46:666
add honeypot30intern tcp port 667 proxy 192.168.2.46:667
add honeypot30intern tcp port 700 proxy 192.168.2.46:700
add honeypot30intern tcp port 1025 proxy 192.168.2.46:1025
add honeypot30intern tcp port 17666 proxy 192.168.2.46:17666
create honeypot30extern
set honeypot30extern personality "Windows 2000 Professional"
set honeypot30extern default tcp action block
set honeypot30extern default udp action block
set honeypot30extern default icmp action open
add honeypot30extern tcp port 135 open
add honeypot30extern tcp port 139 open
add honeypot30extern tcp port 445 open
add honeypot30extern tcp port 666 open
add honeypot30extern tcp port 667 open
add honeypot30extern tcp port 700 open
add honeypot30extern tcp port 1025 open
add honeypot30extern tcp port 17666 open
dynamic honeypot30
add honeypot30 use honeypot30intern if source ip = 192.168.2.0/26
add honeypot30 otherwise use honeypot30extern
bind 192.168.2.47 honeypot30
create honeypot40intern
set honeypot40intern personality "Linux 2.4.16 - 2.4.18"
set honeypot40intern default tcp action block
set honeypot40intern default udp action block
set honeypot40intern default icmp action open
add honeypot40intern tcp port 22 proxy 192.168.2.62:22
add honeypot40intern tcp port 111 proxy 192.168.2.62:111
add honeypot40intern tcp port 3306 proxy 192.168.2.62:3306
add honeypot40intern tcp port 6000 proxy 192.168.2.62:6000
create honeypot40extern
set honeypot40extern personality "Linux 2.4.16 - 2.4.18"
set honeypot40extern default tcp action block
set honeypot40extern default udp action block
set honeypot40extern default icmp action open
add honeypot40extern tcp port 22 "./scripts/ssh.sh $ipsrc $sport"
add honeypot40extern tcp port 111 open
add honeypot40extern tcp port 3306 "./scripts/mysql.sh $ipsrc $sport"
add honeypot40extern tcp port 6000 open
dynamic honeypot40
add honeypot40 use honeypot40intern if source ip = 192.168.2.0/26
add honeypot40 otherwise use honeypot40extern
bind 192.168.2.63 honeypot40
```

Appendix C

This appendix contains the dynamic honeypot configuration rules.

There will be four steps to configuring the honeypots and a database to hold information about the each host: its IP-address, its operating-system, and its open-ports. There is also a table in the database to associate hosts with groups. Each group will become one honeypot and has an operating-system-fingerprint, an IP-address, and a set of open-ports. The database also contains a list of all valid values for the operating-system-fingerprint. There is a comparison function called *similar*, that compares two hosts and returns true if they are similar to one another. It is based on the IP-address and the operating-system of each host. There is another comparison function called *like*, the compares an operating-system and an operating-system-fingerprint, and returns true if they are alike. It is based on simple string comparison.

Rules are listed in order of priority, meaning that the rule that appears earliest in the list has the highest priority. The rule with highest priority is the rule to be applied. In cases where the one rule's conditions are a superset of another's, the rule whose conditions are the super set will be used. Initially there are no groups, and the step is group-hosts.

Rules

1.1	If	the step is group-host
		there is a host, A, that is not a member of a group
		there is a host, B, that is a member of group, X
		A is *similar* to B.
	Then	A is a member of group X
1.2	If	the step is group-host
		there is a host, A, that is not a member of a group.
	then	create a new group Y, A is a member of group Y.
1.3	If	the step is group-host
	Then	step is select-os.
2.1	If	the step is select-os
		group X does not have an operating-system-fingerprint
		M is a valid operating-system-fingerprint
		host A is in group X
		the operating-system of A is *like* M
	Then	the operating-system-fingerprint of X is M
2.2	If	the step is select-os
	Then	the step is select-ip
3.1	If	the step is select-ip
		group X does not have an IP-address
		host A is a member of group X
		Q is the IP-address of A with one bit changed
		Q is not an IP address of any host
		Q is not an IP-address of any group
	Then	the IP-address of group X is Q
3.2	If	the step is select-ip
	Then	the step is select-ports
4.1	If	the step is select-ports
		group X does not have any open-ports
		P is all open-ports of any host that is a member of group X
	Then	the open-ports of group X is P
4.2	If	the step is select-ports
	Then	stop

Appendix D

Possible alarm rules for the dynamic honeypot based intrusion detection system.

Operation:

A sensor will capture all traffic to the dynamic honeypots, and each packet to a honeypot will be considered an anomalous event. Each event will have a source, destination, etc. All events will be stored in a database. The following rules will operate on a subset of these events, called the active_events. Active_events are all events that have occurred since a specified period of time in the past, i.e.: active_events are all events that have occurred in the last 24 hours. The current_event is the event most recently generated.

Rules:

Basic Rules or transaction level rules:

#1.1

If the total number of active_events exceeds the threshold max_events, then generate a high number of anomalies alarm ("There has been a significant increase in the traffic to your honeypots").

#1.2

If the total number of active_events exceeds the threshold max_events, and one source is associated with more than X% of all active_events then generate a high traffic one source alarm ("there has been a significant increase in the traffic to your honeypots, the majority of which originated with <src>").

#1.3

If the total number of active-events exceeds the threshold max_events, and one destination address is more than X% of all events then generate high traffic one destination alarm ("there has been a significant increase in the traffic to your honeypots, Y% of which has been with <dst>").

#1.4

If the total number of active_events exceeds the threshold max_events and one dest port is more than X% of all events then generate high traffic single port alarm("there has been

significant increase in the traffic to your honeypots, Y% of which was directed at port <des port>, potentially scanning for a new vulnerability”).

#1.5

If the total number of active_events exceeds the threshold max_events and one source is associated with more then X% of all active_events and one destination is associated with more than XX % of all active_events generate high traffic persistent attacker alarm (“there has been significant increase in the traffic to your honeypots, Y% of which came from <src> and YY% went to <dest>”).

#1.6

If any active_event is from the external network then generate perimeter penetration alarm (“Packet(s) from the external network <net_id> have reached a honeypot: <list of relevant events>”). (**depending on the network architecture and firewall rules, this rule might not apply and can be disabled)

#1.7

If the total number of active_events having the same source is greater than X, and the percent of these with the same destination port is greater than Y then generate scanning for vulnerability alarm (“Source <src> appears to be scanning for a specific vulnerability on port <dest port>, <list relevant events>”).

Session rules:

Trigger:

If current_event is part of a legitimate session (currently only TCP) and this is the last packet in the session, or penultimate packet in the session, then extract the entire session stream, remember the source and destination, *check against rules* (yields alerts), *calculate bytes/packet*, and activate session rules.

Check against rules: Run only this session through a signature based IDS and capture the output as alerts. If such an IDS is deployed on the network use the same rules.

Calculate bytes/packet: Calculate the total number of bytes transmitted, and the total number of packets used to transmit that data, and determine the bytes/packet (possible fragmentation).

#2.1

IF alerts != null then generate known threat detected alarm ("the following session occurred between source and destination at <time>. One or more potential threats was identified <print alarms> <print session>").

#2.2

IF alerts != null and bytes/packet is < X then generate known exploit and tcp fragmentation detected alarm ("The following session between src and dest occurred at <time>. The bytes transmitted per package was unusually low (<val>), and one or more potential threats was identified. <print alarms> <print session>").

#2.3

IF session port = http and alerts == null and URI != / , /index.htm, …etc., then generate suspicious http session alarm ("the following http session occurred between <src> and <dest> at <time>. It is not a normally recognized convention, but failed to sound any defined IDS alarms.").

#2.4

IF alerts == null then generate supicious session alarm ("the following session occurred between <src> and one of your honeypots (<dest>) at <time>. It failed to trigger any defined IDS alarms.").

Appendix E

Output of nmap scan of the network, prior to honeypot deployment. The scan is preformed from outside the network.

```
Nmap -sT -e eth0 192.168.2.0/24 -F

Starting nmap 3.50 ( http://www.insecure.org/nmap/ ) at 2004-09-20 11:56 EDT
Host 192.168.2.0 seems to be a subnet broadcast address (returned 3 extra pings).
Skipping host.
Interesting ports on 192.168.2.1:
(The 1214 ports scanned but not shown below are in state: filtered)
PORT    STATE  SERVICE
53/tcp  closed domain
80/tcp  open   http
515/tcp open   printer

Interesting ports on 192.168.2.16:
(The 1213 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
22/tcp   open  ssh
443/tcp  open  https
1241/tcp open  nessus
3306/tcp open  mysql

Interesting ports on 192.168.2.22:
(The 1213 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
1025/tcp open  NFS-or-IIS

Interesting ports on 192.168.2.40:
(The 1212 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE
21/tcp  open  ftp
22/tcp  open  ssh
80/tcp  open  http
111/tcp open  rpcbind
443/tcp open  https

Interesting ports on 192.168.2.62:
(The 1213 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
22/tcp   open  ssh
111/tcp  open  rpcbind
3306/tcp open  mysql
6000/tcp open  X11

Host 192.168.2.255 seems to be a subnet broadcast address (returned 3 extra pings).
Skipping host.
Nmap run completed -- 256 IP addresses (5 hosts up) scanned in 60.636 seconds
```

Appendix F

Results of an nmap scan of the network, preformed from outside the network.

```
Nmap -sT 192.168.2.0/24 -F


Starting nmap 3.50 ( http://www.insecure.org/nmap ) at 2004-09-20 16:20 Eastern Standard
Time
Host 192.168.2.0 seems to be a subnet broadcast address (returned 1 extra pings).
Skipping host.
Interesting ports on 192.168.2.1:
(The 1210 ports scanned but not shown below are in state: filtered)
PORT     STATE SERVICE
21/tcp   open  ftp
25/tcp   open  smtp
110/tcp  open  pop3
389/tcp  open  ldap
515/tcp  open  printer
1002/tcp open  windows-icfw
1720/tcp open  H.323/Q.931

Interesting ports on 192.168.2.16:
(The 1209 ports scanned but not shown below are in state: filtered)
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
25/tcp   open  smtp
110/tcp  open  pop3
389/tcp  open  ldap
1002/tcp open  windows-icfw
1241/tcp open  nessus
1720/tcp open  H.323/Q.931

Interesting ports on 192.168.2.22:
(The 1210 ports scanned but not shown below are in state: filtered)
PORT     STATE SERVICE
21/tcp   open  ftp
25/tcp   open  smtp
110/tcp  open  pop3
139/tcp  open  netbios-ssn
389/tcp  open  ldap
1002/tcp open  windows-icfw
1720/tcp open  H.323/Q.931

Interesting ports on 192.168.2.40:
(The 1208 ports scanned but not shown below are in state: filtered)
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
25/tcp   open  smtp
80/tcp   open  http
110/tcp  open  pop3
389/tcp  open  ldap
443/tcp  open  https
1002/tcp open  windows-icfw
1720/tcp open  H.323/Q.931

Interesting ports on 192.168.2.62:
(The 1209 ports scanned but not shown below are in state: filtered)
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
25/tcp   open  smtp
110/tcp  open  pop3
111/tcp  open  rpcbind
```

```
389/tcp  open  ldap
1002/tcp open  windows-icfw
1720/tcp open  H.323/Q.931

Host 192.168.2.255 seems to be a subnet broadcast address (returned 3 extra pings).
Skipping host.
Nmap run completed -- 256 IP addresses (5 hosts up) scanned in 2141.656 seconds Nmap run
completed -- 256 IP addresses (4 hosts up) scanned in 1667.297 seconds
```

Appendix G

Initial honeyd configuration file generated by the dynamic honeypot.

```
create default
set default personality "Windows NT4 / Win95 / Win98"
set default default tcp action block
set default default udp action block
set default default icmp action block

create honeypot10intern
set honeypot10intern personality "Linux 2.4.16 - 2.4.18"
set honeypot10intern default tcp action block
set honeypot10intern default udp action block
set honeypot10intern default icmp action open
add honeypot10intern tcp port 22 proxy 192.168.2.16:22
add honeypot10intern tcp port 443 proxy 192.168.2.16:443
add honeypot10intern tcp port 1241 proxy 192.168.2.16:1241
add honeypot10intern tcp port 3306 proxy 192.168.2.16:3306
create honeypot10extern
set honeypot10extern personality "Linux 2.4.16 - 2.4.18"
set honeypot10extern default tcp action block
set honeypot10extern default udp action block
set honeypot10extern default icmp action open
add honeypot10extern tcp port 22 "./scripts/ssh.sh $ipsrc $sport"
add honeypot10extern tcp port 443 open
add honeypot10extern tcp port 1241 open
add honeypot10extern tcp port 3306 "./scripts/mysql.sh $ipsrc $sport"
dynamic honeypot10
add honeypot10 use honeypot10intern if source ip = 192.168.2.0/26
add honeypot10 otherwise use honeypot10extern
bind 192.168.2.17 honeypot10
create honeypot20intern
set honeypot20intern personality "Windows 2000 SP2"
set honeypot20intern default tcp action block
set honeypot20intern default udp action block
set honeypot20intern default icmp action open
add honeypot20intern tcp port 135 proxy 192.168.2.22:135
add honeypot20intern tcp port 139 proxy 192.168.2.22:139
add honeypot20intern tcp port 445 proxy 192.168.2.22:445
add honeypot20intern tcp port 1025 proxy 192.168.2.22:1025
create honeypot20extern
set honeypot20extern personality "Windows 2000 SP2"
set honeypot20extern default tcp action block
set honeypot20extern default udp action block
set honeypot20extern default icmp action open
add honeypot20extern tcp port 135 open
add honeypot20extern tcp port 139 open
add honeypot20extern tcp port 445 open
add honeypot20extern tcp port 1025 open
dynamic honeypot20
add honeypot20 use honeypot20intern if source ip = 192.168.2.0/26
add honeypot20 otherwise use honeypot20extern
bind 192.168.2.23 honeypot20
create honeypot30intern
set honeypot30intern personality "Linux 2.4.16 - 2.4.18"
set honeypot30intern default tcp action block
set honeypot30intern default udp action block
set honeypot30intern default icmp action open
add honeypot30intern tcp port 21 proxy 192.168.2.40:21
add honeypot30intern tcp port 22 proxy 192.168.2.40:22
add honeypot30intern tcp port 80 proxy 192.168.2.40:80
add honeypot30intern tcp port 111 proxy 192.168.2.40:111
add honeypot30intern tcp port 443 proxy 192.168.2.40:443
create honeypot30extern
set honeypot30extern personality "Linux 2.4.16 - 2.4.18"
set honeypot30extern default tcp action block
```

```
set honeypot30extern default udp action block
set honeypot30extern default icmp action open
add honeypot30extern tcp port 21 "./scripts/ftp.sh $ipsrc $sport"
add honeypot30extern tcp port 22 "./scripts/ssh.sh $ipsrc $sport"
add honeypot30extern tcp port 80 "./scripts/iis.sh $ipsrc $sport"
add honeypot30extern tcp port 111 open
add honeypot30extern tcp port 443 open
dynamic honeypot30
add honeypot30 use honeypot30intern if source ip = 192.168.2.0/26
add honeypot30 otherwise use honeypot30extern
bind 192.168.2.41 honeypot30
create honeypot40intern
set honeypot40intern personality "Linux 2.4.16 - 2.4.18"
set honeypot40intern default tcp action block
set honeypot40intern default udp action block
set honeypot40intern default icmp action open
add honeypot40intern tcp port 22 proxy 192.168.2.62:22
add honeypot40intern tcp port 111 proxy 192.168.2.62:111
add honeypot40intern tcp port 3306 proxy 192.168.2.62:3306
add honeypot40intern tcp port 6000 proxy 192.168.2.62:6000
create honeypot40extern
set honeypot40extern personality "Linux 2.4.16 - 2.4.18"
set honeypot40extern default tcp action block
set honeypot40extern default udp action block
set honeypot40extern default icmp action open
add honeypot40extern tcp port 22 "./scripts/ssh.sh $ipsrc $sport"
add honeypot40extern tcp port 111 open
add honeypot40extern tcp port 3306 "./scripts/mysql.sh $ipsrc $sport"
add honeypot40extern tcp port 6000 open
dynamic honeypot40
add honeypot40 use honeypot40intern if source ip = 192.168.2.0/26
add honeypot40 otherwise use honeypot40extern
bind 192.168.2.63 honeypot40
```

Appendix H

Source Code listings

```
#include<cstdlib>
#include<iostream>
#include</usr/include/mysql/mysql.h>
#ifndef DBIF
#include"datetime.h"

/*
dbif.h
written by Jeff Hieb, July 2004

This is the class definition for the class dbif
class dbif is a database interface class for
a MySQL database.
*/

class dbif {
	MYSQL *mysql;  //data base handle
	MYSQL_RES *result;//stores query results
	public:
	unsigned int num_rows;//number of rows in result
	unsigned int num_fields;//number of fiels in result
	dbif();
		// connect to the database using these parms.
	void connect(char * host,
		char * user,
		char * pass,
		char * name,
		unsigned int port,
		char * socket,
		unsigned int flags);
		// disconnect from the database
	void disconnect();
		//  query the database using supplied query
	void query(char * query);
		// return the current row of the results
	MYSQL_ROW get_row();
		// return the row number row from results
	MYSQL_ROW get_row(unsigned int row);
	void print_results();
		// insert host information into the
		// dynamic honeypot database
	void dyhpinsert(datetime time,char * ipaddr, char*os);
		// insert port information into the
		// dynamic honeypot database
	void portinsert(char *ip,char * port);
		// return true if ip is in the flock table.
	bool isflock(char * ip);
};

#define DBIF
#endif
```

```
#include"dbif.h"
/*
dbif.cpp
written by jeff hieb, July 2004

dbif class method implementations
*/

using namespace std;

dbif::dbif() {
	mysql = NULL;
	result = NULL;
	num_rows = 0;
	num_fields = 0;
}

void dbif::connect(char * host,
		char * user,
		char * pass,
		char * name,
		unsigned int port,
		char * socket,
		unsigned int flags)
{
if ((mysql = mysql_init(NULL)) == NULL) {
	cout << "failed to inititialize\n";
	exit(0);
	}
if (mysql_real_connect(mysql,host,user,pass,name,port,socket,flags) == NULL ){
	cout << "connection failed " << mysql_error(mysql);
	exit (0);
	}
}


void dbif::query(char * query)
{
// first free any previous results.
mysql_free_result(result);
// prefrom query, checking for errors
if(mysql_query(mysql, query) == 0) {
	result = mysql_store_result(mysql);
	if (result != NULL) {
			// set number of rows
		num_rows = mysql_num_rows(result);
			// set number of fields
		num_fields = mysql_field_count(mysql);
		}
	}
else {		//  if an error occured, notify and exit
	cout << query << "\n";
	cout << "Query failed: " << mysql_error(mysql) << "\n";
	exit (0);
	}
}


MYSQL_ROW dbif::get_row() {
return mysql_fetch_row(result);
}

MYSQL_ROW dbif::get_row(unsigned int row) {
if (row < num_rows)
	return mysql_fetch_row(result);
else
	return NULL;
}
```

```
void dbif::print_results() {
unsigned int f;
MYSQL_ROW row;
if (result != NULL) {
	while ((row = mysql_fetch_row(result)) != NULL) {
		f= 0;
		while (f<mysql_num_fields(result)) {
			if (f>0) cout << "\t";
			cout << row[f];
			f++;
			}
		cout << "\n";
		}
	}
else {
	cout << "empty result set\n";
	}
}
bool dbif::isflock(char * ip)
{
char buf[255];

sprintf(buf,"select * from flock where ipaddr = INET_ATON('%s')",ip);
query(buf);
if (num_rows > 0)
	return true;
return false;
}

void dbif::dyhpinsert(datetime time,char* ipaddr,char* os)
{

char buf[255];

// see if there is already an entry for this host
sprintf(buf,"select count from host where ipaddr=INET_ATON('%s') and os='%s'",ipaddr,os);
query(buf);

// if no then insert a new entry, with count = 1
if (num_rows == 0) {
	sprintf(buf,"insert into host
values(INET_ATON('%s'),'%s',1,'%s')",ipaddr,os,time.asMysqlString());
	query(buf);
	}
// if yes, update the count field.
else {
	sprintf(buf,"update host set count=count+1,last='%s' where ipaddr =
INET_ATON('%s') and os = '%s'",time.asMysqlString(),ipaddr,os);
	query(buf);
	}
}

void dbif::portinsert(char* ip,char* port)
{

char buf[255];

sprintf(buf,"select * from ports where ipaddr=INET_ATON('%s') and port='%s'",ip,port);
query(buf);

if (num_rows == 0) {
	sprintf(buf,"insert into ports values(INET_ATON('%s'),%s,NULL,NULL)",ip,port);
	query(buf);
	}
}

void dbif::disconnect()
{
mysql_close(mysql);
}
```

```
/*
datetime.h

written by Jeff Hieb July, 2004.

This is class definition file for the class datetime

*/

#ifndef MYDATETIME

#include<ctime>
#include<cstdlib>
#include<string>

class datetime {
        struct tm time; // system time
        public:
                // standard constructor, sets time to current time
        datetime();
                // constructor, sets time to mysqlstring
        datetime(char* mysqlstring);
                // set time of first operator equal to
                // time of second operator
        datetime operator = (datetime op2);
                // returns true if both datetime objects
                // have the same time
        friend bool operator==(datetime op1,datetime op2);
                // returns true if first datetime object
                // is earlier than the second datetime object
        friend bool operator<(datetime op1,datetime op2);
                // returns time as a MySQL string
        char* datetime::asMysqlString();
                // subtracts x hours from time.
        void datetime::minusHours(int x);
};

#define MYDATETIME

#endif
```

```
#include"datetime.h"
#include<iostream>
datetime::datetime(){
        time_t theTime;
std::time(&theTime);
        time = (*localtime(&theTime));
        }

datetime::datetime(char *mysqlstring) {
        char temp[10];
        int yr,mo,da,hr,mi,se;
       mysqlstring[4]=' ';
       mysqlstring[7]=' ';
       mysqlstring[10]=' ';
       mysqlstring[13]=' ';
       mysqlstring[16]=' ';

       sscanf(mysqlstring,"%d %d %d %d %d %d",&yr,&mo,&da,&hr,&mi,&se);
        time.tm_year = yr-1900;
        time.tm_mon = mo - 1;
        time.tm_mday = da;
        time.tm_hour = hr;
        time.tm_min = mi;
        time.tm_sec = se;
        }
datetime datetime::operator=(datetime op2) {
        time = op2.time;
        return *this;
}

bool operator==(datetime op1,datetime op2) {
        if ((op1.time.tm_year == op2.time.tm_year) &&
               (op1.time.tm_mon == op2.time.tm_year) &&
               (op1.time.tm_mday == op2.time.tm_mday) &&
               (op1.time.tm_hour == op2.time.tm_hour) &&
               (op1.time.tm_min == op2.time.tm_min) &&
               (op1.time.tm_sec == op2.time.tm_sec))
                return true;
        return false;
}

bool operator<(datetime op1, datetime op2) {
double result;
op1.time.tm_isdst = op2.time.tm_isdst;
result = difftime(mktime(&(op2.time)),mktime(&(op1.time)));
if (result > 0)
       return true;
return false;
}

char * datetime::asMysqlString() {
       char buf[80];
       char * ret;
        sprintf(buf,"%d-%d-%d %d:%d:%d",time.tm_year + 1900,\
        time.tm_mon + 1,time.tm_mday,time.tm_hour, \
        time.tm_min, time.tm_sec);
       ret = (char *)malloc(strlen(buf) + 1);
       strcpy(ret,buf);
        return ret;
}
void datetime::minusHours(int x){
        if (time.tm_hour > x)
                time.tm_hour -= x;
        else {
                x -= time.tm_hour;
                time.tm_hour = 23;
                if (time.tm_mday > 1) {
                        time.tm_mday --;
                        }
                else {
```

```
                switch (time.tm_mon) {
                        case 0:
                                time.tm_year --;
                                time.tm_mon = 11;
                                time.tm_mday = 31;
                                break;
                        case 2:
                                time.tm_mon = 1;
                                time.tm_mday = 28;
                                break;
                        case 1:
                        case 3:
                        case 5:
                        case 7:
                        case 8:
                        case 10:
                                time.tm_mon--;
                                time.tm_mday = 31;
                                break;
                        default:
                                time.tm_mon--;
                                time.tm_mday = 30;
                        }
                }
        minusHours(x);
        }
}
```

```
#include<iostream>
#include<fstream>
#include<cstdlib>
#include"datetime.h"
#include"dbif.h"
/*
passive.cpp
written by jeff hieb, july 2004

extract host IP address, date and operating system
from p0f output and place it in the dynamic honeypot data base.
*/
using namespace std;

datetime gettime(char * buffer)
{
char month[4];
char monthnum[3];
char date[20];
int i;

month[0] = buffer[5];month[1] = buffer[6];month[2]=buffer[7];month[3]='\0';
if (strcasecmp(month,"Jan") == 0){ monthnum[0] = '0'; monthnum[1] = '1';}
if (strcasecmp(month,"Feb") == 0){ monthnum[0] = '0'; monthnum[1] = '2';}
if (strcasecmp(month,"Mar") == 0){ monthnum[0] = '0'; monthnum[1] = '3';}
if (strcasecmp(month,"Apr") == 0){ monthnum[0] = '0'; monthnum[1] = '4';}
if (strcasecmp(month,"May") == 0){ monthnum[0] = '0'; monthnum[1] = '5';}
if (strcasecmp(month,"Jun") == 0){ monthnum[0] = '0'; monthnum[1] = '6';}
if (strcasecmp(month,"Jul") == 0){ monthnum[0] = '0'; monthnum[1] = '7';}
if (strcasecmp(month,"Aug") == 0){ monthnum[0] = '0'; monthnum[1] = '8';}
if (strcasecmp(month,"Sep") == 0){ monthnum[0] = '0'; monthnum[1] = '9';}
if (strcasecmp(month,"Oct") == 0){ monthnum[0] = '1'; monthnum[1] = '0';}
if (strcasecmp(month,"Nov") == 0){ monthnum[0] = '1'; monthnum[1] = '1';}
if (strcasecmp(month,"Dec") == 0){ monthnum[0] = '1'; monthnum[1] = '2';}

for (i=0;i<4;i++)
        date[i]=buffer[21+i];
date[4] = ' ';
date[5] = monthnum[0];
date[6] = monthnum[1];
date[7] = ' ';
date[8]= buffer[9];
date[9]=buffer[10];
date[10] = ' ';
for (i=11;i<19;i++)
        date[i] = buffer[i+1];
date[19] = '\0';

return datetime((char*)date);
}

void getip(char* buffer, char *addr)
{
int i;
for (i=0;i<20;i++)
        addr[i] = '\0';

for (i = 27;i<45;i++){
        if(buffer[i] == ':') break;
        addr[i -27] = buffer[i];
        }
addr[i] = '\0';
}

char * getos(char * buffer)
{
char temp[80];
char *temp2;
int i;
temp2 = index(buffer, '-') + 2;
```

```
for (i = 0;i < 80;i++)
        {
        if (temp2[i] == '(' || temp2[i] == ',' || temp2[i] == '[' || i > strlen(temp2)) {
                temp[i] = '\0';break;
                }
        temp[i] = temp2[i];
        }
temp2 = (char*)malloc(strlen(temp) + 1);
strcpy(temp2,temp);
return temp2;
}

void getipport(char* buffer,char*ip,char*port)
{
int i,j;
for (i=0;i<20;i++)
         ip[i] = '\0';
i = 0;
for(j=0;j<4;j++)
        {
        while(isdigit(buffer[i])) ip[i]=buffer[i++];
        ip[i++]='.';
        }
ip[--i]='\0';
j=++i;
while(isdigit(buffer[i])) port[i-j]=buffer[i++];
port[i-j] = '\0';

}



int  main() {

char buf[255];
datetime time;
char ip[20];
char *os;
int c;
char *ipport;
char port[16];
dbif hpdb;



hpdb.connect("localhost","p0f","xyz","dyhp_db",3306,NULL,0);

cin.getline(buf,255);
while (cin) {
        if (buf[0] == '<'){
                time = gettime(buf);
                getip(buf, ip);
                os = getos(buf);
                if (hpdb.isflock(ip))
                        {
                        hpdb.dyhpinsert(time,ip,os);
                        }

                free(os);
                }
        cin.getline(buf,255);
        }


return 0;
}
```

```
#include<iostream>
#include<fstream>
#include<cstdlib>
#include"datetime.h"
#include"dbif.h"
/*
port.cpp
written by jeff hieb, july 2004

extract port number and IP address from tcpdump data and
insert into dynamic honeypot database.
*/
using namespace std;


void getipport(char* buffer,char*ip,char*port)
{
int i,j;
for (i=0;i<20;i++)
        ip[i] = '\0';
i = 0;
for(j=0;j<4;j++)
       {
       while(isdigit(buffer[i])) ip[i]=buffer[i++];
       ip[i++]='.';
       }
ip[--i]='\0';
j=++i;
while(isdigit(buffer[i])) port[i-j]=buffer[i++];
port[i-j] = '\0';

}



int  main() {

char buf[255];
datetime time;
char ip[20];
char *os;
int c;
char *ipport;
char port[16];
dbif hpdb;



hpdb.connect("localhost","p0f","xyz","dyhp_db",3306,NULL,0);

cin.getline(buf,255);
while ( cin) {
       ipport = index(buf,'P') +2;
       getipport(ipport,ip,port);
       if (hpdb.isflock(ip))
              {
              hpdb.portinsert(ip,port);
              }
       cin.getline(buf,255);
       }

return 0;
}
```

```
#include<iostream>
#include<fstream>
#include<cstdlib>
#include"datetime.h"
#include"dbif.h"
/*
flock.cpp
writen by jeff hieb, july 2004

recieve ip addresses from standard input and place them
in the flock table.
*/

using namespace std;


int  main() {

char buf[255];
datetime time;
char ip[20];
char *os;
int c;
char *ipport;
char port[16];
dbif hpdb;



hpdb.connect("localhost","p0f","xyz","dyhp_db",3306,NULL,0);
cin >> ip;
while (cin) {
        if (strcmp(ip,"0.0.0.0") != 0)
                {
                sprintf(buf,"replace into flock values (INET_ATON('%s'),null)",ip);
                hpdb.query(buf);
                }
        cin >> ip;
}

return 0;
}
```

```
/*
main.cpp
written by Jeff Hieb, July 2004

created the dynamic honeypot object,
calles update, and lauched Honeyd, Snort, and Arpd
waits and loops
*/

#include<netinet/in.h>
#include<arpa/inet.h>
#include<cstdlib>
#include"dynhp.h"


using namespace std;


int get_honeydpid()
// get the processes id of honeyd
{
ifstream fp;
fp.open("/var/run/honeyd.pid");

int pid;
fp >> pid;
fp.close();
return pid;
}

int get_snortpid()
// get the process id of Snort
{
ifstream fp;
fp.open("/var/run/snort_eth0.pid");

int pid;

fp >> pid;

fp.close();
return pid;
}

int get_arpdpid()
// get the process id of arpd
{
ifstream fp;

fp.open("/var/run/arpd.pid");

int pid;

fp >> pid;

fp.close();
return pid;
}


void config_snort(dynhp * thehp)
// write the addtional snort configuration file
// "dynhp.config" that contains the variable
// $HOME_NET, a list of the honeypot IP addresses
{
uint32_t * hpips;
int * hpports;
int numips,numports;
ofstream sensor;
struct in_addr ip;
```

```
hpips = thehp->get_hp_ip(numips);
hpports = thehp->get_hp_ports(numports);
sensor.open("dynhp.config");
sensor << "VAR HOME_NET [";
ip.s_addr = ntohl(hpips[0]);
sensor << inet_ntoa(ip);
for(int i =1;i<numips;i++)
        {
        ip.s_addr = ntohl(hpips[i]);
        sensor << "," << inet_ntoa(ip);
        }
sensor << "]\n";
sensor << "VAR HP_PORTS ";
for(int i = 0;i<numports;i++)
        {
        sensor << hpports[i] << " ";
        }
sensor << "\n";
sensor.close();
delete [] hpips;
delete [] hpports;
}

int main (int argc, char *argv)
{
int honeydpid,snortpid,arpdpid;
char command1[40], command2[40], command3[40];
char test[40];

// create the dynamic honeypot object,
// IP address is hard coded, could be passed through command line
dynhp mydynhp("192.168.2.16");

while(true){

// update the honeypot definitions
mydynhp.update();
// configure snort
config_snort(&mydynhp);
// write the honeyd config file
mydynhp.write_config();

// start aprd, snort, and honeyd
system("arpd");
system("snort -D -l ./hplogs -c snorthp.conf");
system("honeyd -l /home/sysjeff/dynhp/hplogs/honeyd.log -p nmap-os-fingerprints -f
honeyd.conf");

// wait one day

system("sleep 24h");

// get process ids
arpdpid = get_arpdpid();
honeydpid = get_honeydpid();
snortpid = get_snortpid();

sprintf(command1,"kill  %d",honeydpid);
sprintf(command2,"kill  %d",snortpid);
sprintf(command3,"kill  %d",arpdpid);

// clean up
mydynhp.rm_hp_hosts();

// kill services
system(command1);
system(command2);
system(command3);
} // end of while
}
```

```
/*
dynhp.h
written by jeff hieb, july 2004

this is the definition file for the class dynhp
dynhp is the main dynhp engine.
*/

#include<iostream>
#include<fstream>
#include<cstdlib>
#include"dbif.h"
#include<netinet/in.h>
#include<arpa/inet.h>
#include<iostream>
#include<fstream>
using namespace std;

class dynhp {
        unsigned long addr_sp_size;// based on hosts IP address
        int num_hosts;
        int density ;// % of average distance to be used for threshold
        int threshold;// man IP address distance between to hosts
                      // in the same group
        long interval;// number of seconds prior to current time
                      // beyond which hosts information
                      // should not be considered
        int hostbits; // number of bits used to distinguish hosts.
        char *ip;     // IP address of the local interface
        char net[20]; // network address
        char buf[255];// buffer used by various methods.
        MYSQL_ROW row;
        dbif database;// dynamic honeypot database interface
        ofstream config;// file stream for writting configuraiton files
        void calc_net();// calculate the network address
        void selectip(int hpid);
        void selectos(int hpid);
        void selectport(int hpid);
public:
        dynhp();
        ~dynhp();
        dynhp(char * theNet);
        void get_addr_sp_size();
        void get_num_hosts();
        void calc_threshold();
        int is_in_honeypot(uint32_t ip);
                // return the honeypot id for the address ip
                // or -1 if not in a honeypot group
        void update();
                // update configuration information
        void rm_hp_hosts();
                // delete any honeypot data from the
                // hosts tables
        void configurehpip();
        void configurehpos();
        void configurehpport();
        void write_config();
        void write_config(int hpid);
        int member(uint32_t ip);
                // return the honeypot group to which ip
                // belongs, or -1 if none is found
        int recurmem(uint32_t ip,char*os,int t,int hpid);
                // recursive membership, restrict ip
                // distance till 1 or 0 groups
        void add_host(uint32_t ip, int hpid);
        void mk_new_hp(uint32_t ip);
        uint32_t *get_hp_ip(int & num);
        int *get_hp_ports(int & num);
};
```

```
/*
dynhp.cpp
written by jeff hieb, July 2004

these are the method implementations
for the class dynhp, the dynamic honeypot engine
*/
#include"dynhp.h"

using namespace std;

dynhp::dynhp()
// default constructor, use IP = 0.0.0.0, and net = 0.0.0.0
{
ip = new char[strlen("0.0.0.0") +1];
strcpy(net,"0.0.0.0");
density = 25; // default setting, translates to 75%
database.connect("localhost","p0f","xyz","dyhp_db",3306,NULL,0);
interval = 10 * 1000000;//period of time, in seconds that
                       //hosts are considered active.
}

dynhp::dynhp(char * theIP)
// constructor, theIP is the IP address from the interface that
// honeyd will listen on
{
ip = new char[strlen(theIP) +1];
strcpy(ip,theIP);
density = 25;
database.connect("localhost","p0f","xyz","dyhp_db",3306,NULL,0);
interval = 10 * 1000000;
}


dynhp::~dynhp()
{
rm_hp_hosts();

database.query("delete from honeyhosts");
database.query("delete from honeypots");
database.query("delete from honeyports");
}


void dynhp::get_addr_sp_size()
// determine the address space size
{
int i,j;
uint32_t a,b,x,y,mask;
bool flag = false;
MYSQL_ROW cursor;
mask = 0x80000000;

// get all the IP addresses of the hosts
sprintf(buf,"select ipaddr from host");
database.query(buf);
if (database.num_rows < 2)
        exit (0);
cursor = database.get_row();
if (cursor == NULL) exit(0);
a = atoll(cursor[0]);
//  loop until a bit is found that is the same for all IP addresses
//  j is that bit, or until all 32 bits have been examined.
for (j = 0; j< 32; j++)
        {
        x = a<<j & mask;
        for (i = 1; i<database.num_rows;i++)
                {
                cursor = database.get_row(i);
                b=atoll(cursor[0]);
```

```
                if (x != (b<<j & mask))
                        {
                        flag = true;
                        break;
                        }
                }
        if (flag) break;
        }
hostbits = j; // j is the number of bits used to distingish the hosts
        // addr_sp_size is the total number of hosts
        // possible using only j bits
addr_sp_size = (unsigned long) pow((double)2,(31-j)+1);

}

void dynhp::get_num_hosts()
{
sprintf(buf,"select count(distinct ipaddr) from host where last > now() - %d",interval);
database.query(buf);
row = database.get_row();
if (row == NULL)
        exit(0);
num_hosts = atoi(row[0]);
}



void dynhp::calc_threshold()
// determine the threshold value,
// from the address space size an the number of hosts.
{
int avg_dist;
double local_d;
double mydensity;

avg_dist = addr_sp_size / num_hosts;
density = density % 100;
mydensity = (double)(100- density);
local_d = mydensity / 100;
threshold = (int)(local_d * avg_dist);

}

void dynhp::calc_net()
{
struct in_addr ipaddr;
uint32_t temp;
char * out;

ipaddr.s_addr = inet_addr(ip);
ipaddr.s_addr = ntohl(ipaddr.s_addr);
temp = (uint32_t) (pow((double)2,(32 -hostbits)) - 1);
ipaddr.s_addr = ipaddr.s_addr & (~temp);
ipaddr.s_addr = ntohl(ipaddr.s_addr);
sprintf(net,"%s/%d",inet_ntoa(ipaddr),hostbits);
}

int dynhp::is_in_honeypot(uint32_t ip)
{

int a;
sprintf(buf,"select hpid from honeyhosts where ipaddr = %u",ip);
database.query(buf);
if (database.num_rows > 0)
        {
        row = database.get_row();
        a = atoi(row[0]);
        }
else a = -1;

return a;
```

```
}


void dynhp::rm_hp_hosts()
// remove any honeypots that have fingerprinted and
// inserted into the host table
{

int c,i;
uint32_t *iplist;

sprintf(buf,"select distinct ipaddr from honeypots");
database.query(buf);
c=database.num_rows;
if (c==0) return;
iplist = new uint32_t[c];

for (i = 0;i<c;i++)
        {
        row = database.get_row();
        iplist[i] = atoll(row[0]);
        }
for (i = 0;i<c;i++)
        {
        sprintf(buf,"delete from flock where ipaddr = %u",iplist[i]);
        database.query(buf);
        sprintf(buf,"delete from host where ipaddr = %u",iplist[i]);
        database.query(buf);
        }
delete [] iplist;

}


void dynhp::update()
// update the honeypot configurations
{
int i,j;
int c;
int hpid,hpid_temp;
uint32_t ip,*iplist;

// clear any previous work
rm_hp_hosts();

database.query("delete from honeyhosts");
database.query("delete from honeypots");
database.query("delete from honeyports");

// set up the parameters
get_addr_sp_size();
get_num_hosts();
calc_threshold();
calc_net();


// get all the relevant hosts from the host table
// and put them in an array called iplist
sprintf(buf,"select distinct ipaddr from host where last > now() - %d order by ipaddr",
interval);
database.query(buf);
c = database.num_rows;
iplist = new uint32_t[c];
for (i = 0;i<c;i++)
        {
        row = database.get_row();
        iplist[i] = atoll(row[0]);
        }
// partition the hosts into groups
for (i=0;i<c;i++)
        {
```

```
        ip = iplist[i];
        hpid = member(ip);
        if (hpid == -1)
                {
                mk_new_hp(ip);
                }
        else
                {
                add_host(ip,hpid);
                }
        }

delete [] iplist;
// configure the honeypots
configurehpip();
configurehpos();
configurehpport();

}


int dynhp::member(uint32_t ip)
// determine the group to which the host ip belongs
// and retrun it, if no group is identified, return -1
{
char ostype[80];
int hpid;
int x,y,i;

// get the most common operating system finger print
// for this IP address
sprintf(buf,"select os from host where ipaddr = %u order by count desc",ip);
database.query(buf);
if (database.num_rows == 0) return -1;
row = database.get_row();
i=0;
// parse out the intial word, to be the operating system type
while(isalnum(row[0][i]) && i < 40)
        {
        ostype[i]=row[0][i];
        i++;
        }
ostype[i]='%';
ostype[i+1]='\0';

// find all honeypot groups (hpid) with hosts whose  IP address
// is within threshold, and
// whose os is stringwise similar to the operating system type
sprintf(buf,"select t1.hpid from honeypots as t1,honeyhosts as t2,host as t3 where
t1.hpid = t2.hpid and t2.ipaddr = t3.ipaddr and t2.ipaddr between %u and %u and t3.os
like '%s'",ip-threshold,ip+threshold,ostype);
database.query(buf);

// if none return -1
if (database.num_rows==0) return -1;
// if only one honeypot group, return this group id
if (database.num_rows==1)
        {
        row = database.get_row();
        return atoi(row[0]);
        }
// if there is more than one group, reduce the threshold by one half
// and continue recursively till one group can be identified.
if (database.num_rows > 1)
        {
        row = database.get_row();
        hpid =  atoi(row[0]);
        return  recurmem(ip,ostype,(int)threshold/2,hpid);
        }
}
```

```
int dynhp::recurmem(uint32_t ip, char * os, int t, int hpid)
// recursive memebership function
{
int l_hpid;

sprintf(buf,"select t1.hpid from honeypots as t1,honeyhosts as t2,host as t3 where
t1.hpid = t2.hpid and t2.ipaddr = t3.ipaddr and t2.ipaddr between %u and %u and t2.ipaddr
!= %u and t3.os like '%s'",ip,ip-t,ip+t,os);
database.query(buf);

if (database.num_rows == 1)
        {
        row = database.get_row();
        return atoi(row[0]);
        }
else if (database.num_rows == 0)
        {
        return hpid;
        }
else    {
        return recurmem(ip,os,(int)t/2,hpid);
        }
}


void dynhp::add_host(uint32_t ip, int hpid)
// associates ip to hpid.
{
sprintf(buf,"insert into honeyhosts values(%d,%u)",hpid,ip);
database.query(buf);
}

void dynhp::mk_new_hp(uint32_t ip)
// create a new honeypot group
{

int hpid;

database.query("select hpid from honeypots order by hpid");
if (database.num_rows == 0)
        hpid = 10;
else    {
        row = database.get_row(database.num_rows - 1);
        hpid = atoi(row[0]) + 10;
        }
sprintf(buf,"insert into honeypots values (%d,NULL,NULL)",hpid);
database.query(buf);
add_host(ip,hpid);
}


uint32_t * dynhp::get_hp_ip(int & num)
// return an array of IP addresses that contains
// all the honeypot IP addresses
{
uint32_t * iplist;
int c;
database.query("select ipaddr from honeypots where ipaddr is not NULL");
c = database.num_rows;
iplist = new uint32_t[c];
for (int i = 0;i<c;i++)
        {
        row = database.get_row();
        iplist[i] = atoll(row[0]);
        }
num = c;
return iplist;
}

int * dynhp::get_hp_ports(int & num)
```

```
// return an array that contains all the
// open ports on all the honeypots
{


int c,*portlist;
database.query("select distinct port from honeyports");
c = database.num_rows;
portlist = new int[c];
for (int i = 0;i<c;i++)
        {
        row = database.get_row();
        portlist[i] = atoi(row[0]);
        }
num = c;
return portlist;
}






void dynhp::configurehpip()
// establish the IP address for each honeypot
{

int *hpid,c;
database.query("select hpid from honeypots where ipaddr is NULL");
c = database.num_rows;
hpid = new int[c];
for(int i =0;i<c;i++)
        {
        row = database.get_row();
        hpid[i] = atoi(row[0]);
        }
for(int i = 0;i<c;i++)
        {
        selectip(hpid[i]);
        }
delete [] hpid;

}


void dynhp::selectip(int hpid)
// determine the ip address for the honeypot indicated by hpid
{
int i,j;
uint32_t *iplist,a,b,x,y,newip;
bool flag = false;
int lowbit;
int offset;
unsigned int val;

// get all the host IP addresses associated with this honeypot
sprintf(buf,"select ipaddr from honeyhosts where hpid = %d",hpid);
database.query(buf);
int c = database.num_rows;
if (c == 0) return;

// put them in an array
iplist = new uint32_t[c];
for (i=0;i<c;i++)
        {
        row = database.get_row();
        iplist[i] = atoll(row[0]);
        }

// find the first (low order) bit that is not the
// same for each host. j is the bit number
```

```
if (c == 1) flag = true;
for (j = 0; j<=hostbits; j++)
        {
        x = iplist[0]>>j & 01;
        for (i = 1; i<c;i++)
                {
                b=iplist[i];
                if (x != (b<<j & 01))
                        {
                        flag = true;
                        break;
                        }
                }
        if (flag) break;
        }
// set the low bit to be j
lowbit = j;
// generate a random number between one and c, the number of hosts
offset = (int)(c *(rand()/(RAND_MAX + 1.0)));
// continue to flip successively higher bits of each host
// till and IP address is found that is not in use,
for (j=lowbit;j<hostbits;j++)
        {
        val = 0;
        for (int k=lowbit;k<j;k++)
                {
                val += (unsigned int)pow((double)2,k);
                for (i=0;i<c;i++)
                        {
                        // get new IP address
                        newip = (~(~iplist[(i+offset) % c] ^ val));
                        // see if it is in use (from flock table)
                        sprintf(buf,"select ipaddr from flock where ipaddr = %u",newip);
                        database.query(buf);
                        // if not in use, store it in the database
                        // and exit
                        if (database.num_rows == 0){
                                sprintf(buf,"update honeypots set ipaddr = %u where hpid =
%d",newip,hpid);
                                database.query(buf);
                                return;
                                }
                        }
                }
        }
delete [] iplist;
}



void dynhp::configurehpos()
// determine the operating system finger print for each honeypot
{

int *hpid,c;
database.query("select hpid from honeypots where os is NULL");
c = database.num_rows;
hpid = new int[c];

for(int i =0;i<c;i++)
        {
        row = database.get_row();
        hpid[i] = atoi(row[0]);
        }

for(int i = 0;i<c;i++)
        {

        selectos(hpid[i]);
        }
```

```
delete [] hpid;
}

void dynhp::selectos(int hpid)
// determine the operating system finger print for the honeypot
// indicated by hpid
{

char * os_type,*os_temp;
char  os_finger[255];
int os_len;

// get the operating systems for each host that is
// a memeber of the honeypot group hpid
sprintf(buf,"select h.os from host h,honeyhosts t, honeypots p where t.ipaddr = h.ipaddr
and t.hpid = p.hpid and p.hpid = '%d' and h.os != 'UNKNOWN' order by h.count desc",hpid);
database.query(buf);

// parse out the operating system type
// begin string comparision to possible dynamic honeypot
// operating system fingerprints, stored in table osfinger
if (database.num_rows > 0)
        {
        row = database.get_row();
        os_len = strlen(row[0]) + 1;
        os_type = new char[os_len];
        os_temp = new char[os_len];
        for (int i=0;i<os_len;i++) os_temp[i] = '\0';
        strcpy(os_type,row[0]);
        strncpy(os_temp,row[0],3);
        sprintf(buf,"select name from osfinger where name like '%s%%'",os_temp);
        database.query(buf);
        // if more than one fingerprint is identified
        // then use more of the operating system string to
        // narrow the possibilities
        if (database.num_rows >0)
                {
                row = database.get_row();
                strcpy(os_finger,row[0]);
                for (int j = 3; j<os_len;j++)
                        {
                        os_temp[j] = os_type[j];
                        sprintf(buf,"select name from osfinger where name like
'%s%%'",os_temp);
                        database.query(buf);
                        if (database.num_rows==0)
                                break;
                        row = database.get_row();
                         strcpy(os_finger,row[0]);
                        }
                sprintf(buf,"update honeypots set os = '%s' where hpid =
'%d'",os_finger,hpid);
                database.query(buf);
                }
        delete [] os_type;
        delete [] os_temp;
        }
}

void dynhp::configurehpport()
// determine the open ports for each honeypot group
{

int *hpid,c;
sprintf(buf,"delete from honeyports");
database.query(buf);

database.query("select hpid from honeypots");
c = database.num_rows;
hpid = new int[c];
```

```
for(int i =0;i<c;i++)
        {
        row = database.get_row();
        hpid[i] = atoi(row[0]);
        }
for(int i = 0;i<c;i++)
        {
        selectport(hpid[i]);
        }

delete [] hpid;
}


void dynhp::selectport(int hpid)
// determine the open ports for the honeypot group hpid.
{

int *p;
uint32_t *ip;

sprintf(buf,"select distinct p.port, p.ipaddr from ports p,honeypots h, honeyhosts k
where p.ipaddr = k.ipaddr and k.hpid = h.hpid and h.hpid = %d",hpid);

database.query(buf);
int c = database.num_rows;
p = new int[c];
ip = new uint32_t[c];

for (int j = 0;j<c;j++)
        {
        row = database.get_row();
        p[j] = atoi(row[0]);
        ip[j] = atoll(row[1]);
        }

for (int i = 0;i<c; i++)
        {
        sprintf(buf,"replace into honeyports values
(%d,%d,INET_NTOA(%u),NULL)",hpid,p[i],ip[i]);
        database.query(buf);
        }
}

void dynhp::write_config()
// white the honeyd configuration file "honeyd.conf"
// using the dynamic honeypot database.
{

int c;
int *hpid;

config.open("honeyd.conf");
config << "create default\n";
config << "set default personality \"Windows NT4 / Win95 / Win98\"\n";
config << "set default default tcp action block\n";
config << "set default default udp action block\n";
config << "set default default icmp action block\n\n";


database.query("select hpid from honeypots where ipaddr is not NULL and os is not NULL");
c = database.num_rows;
hpid = new int[c];
for(int i =0;i<c;i++)
        {
        row = database.get_row();
        hpid[i] = atoi(row[0]);
        }
for(int j=0;j<c;j++)
        {
        write_config(hpid[j]);
```

```
        }

config.close();
}


void dynhp::write_config(int hpid)
{
int c,i;
char * per,*ip;
int * ports;
char **proxys;

sprintf(buf,"select ipaddr from honeyhosts where hpid = %d",hpid);
database.query(buf);
if (database.num_rows == 0) return;

sprintf(buf,"select os,INET_NTOA(ipaddr) from honeypots where hpid = %d ",hpid);
database.query(buf);

c = database.num_rows;
if (c == 0) return;

row = database.get_row();

per = new char[strlen(row[0]) + 1];
ip = new char[strlen(row[1]) +1];
strcpy(per, row[0]);
strcpy(ip,row[1]);


sprintf(buf,"select port,proxy from honeypots p,honeyports k where p.hpid = %d and p.hpid
= k.hpid",hpid);
database.query(buf);
c=database.num_rows;
ports = new int[c];
proxys = new char*[c];

for(i=0;i<c;i++)
        {
        row = database.get_row();
        ports[i] = atoi(row[0]);
        proxys[i] = new char[strlen(row[1]) + 1];
        strcpy(proxys[i],row[1]);
        }

config << "create honeypot" << hpid << "intern\n";
config << "set honeypot" << hpid << "intern" << " personality \"" << per << "\"\n";
config << "set honeypot" << hpid << "intern" << " default tcp action block\n";
config << "set honeypot" << hpid << "intern" << " default udp action block\n";
config << "set honeypot" << hpid << "intern" << " default icmp action open\n";
for (i=0;i<c; i++)
        {
        row = database.get_row(i);
        config << "add honeypot" << hpid << "intern tcp port " << ports[i];
        config << " proxy " << proxys[i] << ":" << ports[i] << "\n";
        }


config << "create honeypot" << hpid << "extern\n";
config << "set honeypot" << hpid << "extern" << " personality \"" << per << "\"\n";
config << "set honeypot" << hpid << "extern" << " default tcp action block\n";
config << "set honeypot" << hpid << "extern" << " default udp action block\n";
config << "set honeypot" << hpid << "extern" << " default icmp action open\n";
for (i=0;i<c; i++)
         {
         row = database.get_row(i);
         config << "add honeypot" << hpid << "extern tcp port " << ports[i];
        sprintf(buf,"select script from scripts where port = %d",ports[i]);
        database.query(buf);
        if (database.num_rows > 0) {
```

```
                row = database.get_row();
                config << " \"" << row[0] << "\"\n";
                }
        else {
                config << " open\n";
                }
        }

config << "dynamic honeypot" << hpid << "\n";
config << "add honeypot" << hpid << " use honeypot" << hpid;
config << "intern if source ip = " << net << "\n";
config << "add honeypot" << hpid << " otherwise use honeypot" << hpid;
config << "extern\n";
config << "bind " << ip << " honeypot" << hpid << "\n";
```

CURRICULUM VITAE

Jeffrey L. Hieb

497 Sacree Rd. j_hieb@insightbb.com
Shelbyville, Kentucky 40065 (502) 418-6106

EDUCATION

Masters of Science, Computer Engineering and Computer Science, Expected December 2004
University of Louisville, Louisville, Kentucky
Thesis: Anomaly Based Intrusion Detection For Network Monitoring Using a Dymamic Honeypot
Advisor: Dr. James Graham

Bachelor of Science, Computer Science, June 1992
Furman University, Greenville, South Carolina

Bachelor of Arts, Philosophy, June 1992
Furman University, Greenville, South Carolina

HONORS/AFFILIATIONS

Phi Beta Kappa
Upsilon Pi Epsilon
Furman Advantage Research Fellow, Data Structures in C++, 1991
Furman Advantage Teaching Assistant, Introduction to Computer Science, 1991
Furman Advantage Research Fellow, Object Oriented Programming Design, 1989

RESEARCH INTERESTS

- Computer Security
- Artificial Intelligence
- Cognitive Science

PUBLICATIONS

Improving the SmallTalk Browser: A Case Study in SmallTalk Development, Proceedings of 28th Annual Southeast Regional Conference of ACM

WORK EXPERIENCE

Plant Supervisor / Plant Manager, 1992 – 2002
Hieb Concrete Products Inc., Shelbyville, Kentucky.