2009

Evaluating the effectiveness of an intrusion prevention / honeypot hybrid

Lucas Tamagna-Darr

Evaluating the Effectiveness of an Intrusion Prevention / Honeypot Hybrid

October 1, 2009

By

Lucas Tamagna-Darr

Thesis Committee

Yin Pan

Bo Yuan

Charles Border

Thesis submitted in partial fulfillment of the requirements

for the degree of

Master of Science in

Computer Security and Information Assurance

Rochester Institute of Technology

B. Thomas Golisano College

of

Computing and Information Sciences

Department of Network Security and Systems Administration

August, 2009

Acknowledgments

I would like to thank my Thesis committee for their participation on this project, and the guidance they have provided me.

I would like to thank my Parents for providing me with the hardware for the research and their constant support throughout this project.

I would like to thank everyone who took the time to read through my paper and offer me advice on fixes to be made.

Lastly, I would like to thank Heather who has been my motivation for completing this work. You have been there for me throughout this research, keeping me motivated when I didn't want to work, and taking me out when I couldn't work anymore.

Abstract

An intrusion prevention system is a variation of an intrusion detection system that drops packets that are anomalous based on a chosen criteria. An intrusion prevention system is typically placed on the outer perimeter of a network to prevent intruders from reaching vulnerable machines inside the network, though it can also be placed inside the network in front of systems requiring extra security measures. Unfortunately, intrusion prevention systems, even when properly configured, are susceptible to both false positives and false-negatives. The risk of false positives typically leads organizations to deploy these systems with the prevention capability disabled and only focus on detection.

In this paper I propose an expansion to current intrusion prevention systems that combines them with the principles behind honeypots to reduce false positives while capturing attack traffic to improve prevention rules. In an experiment using the Snort-inline intrusion prevention system, I was able to reduce the rate of false positives to zero without negatively impacting the rate of false-negatives. I was further able to capture a successful attack in a way that minimized disruption to legitimate users but allowed the compromised system to be later analyzed to find weaknesses, improve prevention rules, and prevent future attacks.

Contents

1 Introduction

Intrusion prevention systems (IPSs) are an extension of intrusion detection systems (IDSs) that, rather than simply logging alerts, typically drop any network traffic that appears to be anomalous. IPSs are typically used on the outer perimeter of a network to prevent any malicious traffic from reaching potentially vulnerable systems inside the network that may contain sensitive information.

In order to analyze the performance of an IPS, there are several parameters that can be looked at, depending on what is most important to the network the IPS is being deployed upon. The following parameters are commonly used when discussing the fine tuning of any IPS and can be further used to analyze the successfulness of any IPS.

- Latency - Latency is the time it takes for a packet to travel through the IPS to the destination system and return to the user. This is typically measured in round-trip time (RTT).
- False Negative - A false-negative is any malicious traffic that makes it through the IPS to the production network. A false-negative is dangerous because it represents an instance where an attacker penetrates the network and successfully exploits a vulnerable system.
- False Positive - A false-positive is any legitimate traffic that the IPS drops because it appears to be anomalous. A false-positive is harmful to a network because it can deny legitimate users access to the offered services. This can result in lost customers due to inaccessible services.

The two methods most frequently used in intrusion prevention systems are rule based and anomaly based prevention. This paper focuses on the Snort-inline IPS which uses rule-based prevention. Snort-inline is a customized build of the Snort IDS which drops malicious traffic based on a set of pre-specified rules. As is common with many IPS systems, one of the steps taken when deploying Snort-inline is to fine tune the rules to achieve the desired balance between false-positives, false-negatives, and minimized latency.

A relatively unexplored method of intrusion prevention is to combine IPSs with a form of honeypots utilizing bait-and-switch principles. Bait-and-switch is a principle that exposes one system to users while convincing them they are using a different system. Honeypots are great systems for analyzing the activities of attackers. By offering attackers a vulnerable system to exploit, it is possible to distract them from the full production system as well as to monitor their activities to improve defenses and prevent any further attacks. A honeypot typically has several layers of monitoring in order to capture any actions an attacker takes. These layers include monitoring local activity on the honeypot as well as network connections to the honeypot. A weakness of honeypots is that there is no way to be sure an attacker will choose the honeypot over a system in the production network, which can sometimes negate their usefulness. It can also be costly to maintain a full honeynet (a network of honeypots), as it

requires extra hardware and a great deal of effort to monitor logs. While the principles of bait-and-switch are commonly used with malicious intent, it can be a valuable principle in improving the outer defenses of a network.

In my system I utilize bait-and-switch principles in an IPS in order to transparently reroute any anomalous traffic to my honeynet. By doing this, the system gains the advantages of a honeynet without requiring an attacker to "find" the honeynet. For my testing I created a set of poorly constructed set of Snort rules based on network traffic related to the phpBB forum that, in a typical IPS would lead to dropped traffic resulting in false-positives. In our hybrid system the traffic is rerouted to the honeynet, which will be referred to as the "shadow DMZ." The ultimate goal of this research was to develop a proof of concept for a hybrid honeypot that is capable of reducing the rate of false-positives while only having a minimal, if any impact on the rate of false-negatives.

1.1 Objective

1. Use a combination of tools to reduce false-positives in an Intrusion Prevention System (Snort-inline) without removing rules between tests. Keeping the rules for both sets of tests ensures that there is no reduction in false-negatives.

2. The system must be capable of gracefully and automatically recovering from attack.

   (a) Gracefully - After attack there must be a compromised system to analyze for evidence. In addition, the original system must be returned to a safe state so that service interruption for legitimate users is minimal.

   (b) Automatically - The process of recovering from an attack should be fully automated. This ensures that the attacker is instantly locked out and services are restored without the delay of manual intervention.

1.2 Limitations and Assumptions

The proposed system has the following limitations.

1. The system relies on the IPS to perform the bait-and-switch function. In its current form, I only know of this functionality existing in Snort-inline.

2. IPS rules are still reliant on manual development. As a result, it is still possible to circumvent the IPS through careful testing and probing. If a rule does not exist to match against a set of packets, the traffic will make it to the production network whether it is malicious or not.

3. Although recovering from a successful attack against the Shadow DMZ is automated, investigation of how the attack occurred is still manual.

4. Because of the limitations of hardware and its ability to handle large amounts of data, I developed a custom set of rules that would create false-positives when trying to access the web front end, as well as a rule to catch a directory traversal attempt. These rules did not directly reflect the available rules from Snort and other vendors.

The following assumptions and decisions were made that mitigate several of these limitations.

Although the bait-and-switch capabilities only exist in Snort-inline, it is a feature that could be added on to any IPS. With open source tools it is possible that the community could write a third party tool to create the same functionality. In a proprietary IPS, the vendor would need to release this functionality as an add-on feature.

The need to rely on manual development of rules is an inherent weakness of most IPSs. With further development of this system, it may be possible that rules could be automatically generated and refined based on attacks against the network. While this system does not look to automate rule creation in an IPS, it can improve the performance in anomaly based IPSs. By monitoring the activity of anomalous traffic in the Shadow DMZ, the network activity baseline can be updated on a near real-time basis. Both of these topics are covered in more depth in the future work section.

The third obstacle is tied to obstacle two and would require further development beyond what this system hopes to achieve. There are tools that exist that can monitor network traffic and build rules based on attacks, but they were only developed for specific honeypots. In the future one of these tools could be modified or new tools can be developed to analyze the traffic in the shadow DMZ in order to create new rules and refine existing ones.

The fourth limitation does not have any effect on the validity of the tests. The rules developed for testing were intentionally created to have a significant effect on the rate of false-positives in the first phase of testing. In a production network the rules, from whichever vendor is chosen, could be modified in the same manner as in this research and achieve the same effect.

1.3 Contributions

This research makes the following contributions.

1. While the bait-and-switch preprocessor has been out for some time, I did not see much discussion on its use for reducing false-positives in an IPS. This research tests the ability of the tools to accomplish this goal in a way that does not increase the rate of false-negatives.

2. The research also utilizes the advantages of virtualization to react to a successful attack in a way that not only preserves evidence, but also returns the system to a safe state in addition to blocking further attempts by the same attacker. This is a process that could be reproduced in other systems in order to gracefully recover from an attack.

3. The proposed system provides a foundation for further research, especially in the area of anomaly based intrusion prevention. There is also the opportunity to use this system to explore learning based intrusion prevention systems through automated rule creation.

1.4 Chapter Summary

Chapter 2 discusses tools and similar articles that are related to this research. Included are several of the tools that were used for this research including Snort-inline, the bait-and-switch preprocessor, honeynets, and virtualization tools as well as research related to the topic of shadow, or hybrid, honeynets.

Chapter 3 details the architecture of the network that I developed for testing. This includes the architecture of a typical IPS protected network and the network with the modified IPS.

Chapter 4 discusses the methodology I used for testing. It details how I tested for false-positives as well as the method I used for tracking an attack and recovering from it in a controlled manner.

Chapter 5 summarizes the results as well as the contributions that are made by this research and looks at several ways that this research could be further expanded.

2 Literature Review

In this chapter we analyze the tools that were used for this research. These include Snort-inline, the bait-and-switch preprocessor, honeynets, and virtualization tools. This is followed by a look at several articles that are relevant to this work. While there has been a significant amount of research related to honeypots and intrusion prevention systems specifically, the combination of these tools with the purpose of reducing false-positives seems to be relatively unexplored. It is important to understand the tools that the proposed system was built on in order to understand how the proposed system will work.

2.1 Honeypots

While honeypots are in no way a new concept, their use in computer security and information technology in general is relatively new. Two key resources to study when planning a honeypot are articles published by Lance Spitzner as well as articles from the Honeynet project. According to Spitzner, a honeypot is "an information resource whose value lies in unauthorized or illicit use of that resource."[12] While this is not exactly the definition that can be applied to the shadow DMZ in my research, as it is also used by legitimate users, the proposed system shares many of the properties of a honeypot. In the case of the proposed system a honeypot is the system that receive the redirected suspicious traffic. The true value of any honeypot, whether the classic definition or the case in this research, is in the data that is gathered during an attack.

There are several advantages that honeypots have over other intrusion detection systems, but one of the greatest is the ability to log traffic even in an environment where traffic is encrypted. This is possible because a majority of the logging that takes place in a honeypot is local where the activity is no longer encrypted. In the area of computer honeypots there are both low interaction and high interaction honeypots, both with advantages and disadvantages.

2.1.1 Low Interaction Honeypots

A low-interaction honeypot works by emulating services and operating systems in software. When attackers connect to a low-interaction honeypot, they are essentially connecting to a script that responds with pre-defined responses.

In Spitzner's article on honeypots, he details the process of a specific low interaction honeypot, honeyd, which "works on the concept of monitoring unused IP space"[12]. When a request comes in for a service on an unused IP, as Spitzner explains, honeyd grabs that request and, if it is emulating the requested service, sends a response. Honeyd tracks all connections as well as requests and responses in a log file for analyzing the activities of the attacker.

A weakness in honeyd is that it can only respond to requests that it is programmed to recognize, and typically does not represent the full range of functions that the emulated service can perform. An experienced attacker can analyze the range of responses in order to determine if they are connected to

honeyd. Spitzner suggests, because of the weaknesses in Honeyd, it is most beneficial in a corporate environment where the risk tolerance associated with the honeypots is typically low.

In order to take advantage of the concepts introduced by Spitzner, I focused on the software's ability to tunnel traffic from the attacker to a physical system, thus minimizing the limitations of Honeyd. In the proposed system, I wanted to use the concept of tunneling to provide users with access to back-end resources through an environment with extended monitoring capabilities.

Low-interaction honeypots are best in environments such as large corporations, where there is value in monitoring attacks, and a low threshold of risk allowance. Because of the way low interaction honeypots are designed, it is possible for an attacker, through extensive fingerprinting, to determine all the IPs on a corporate network that belong to the honeyd process, and thus specifically target production systems. The proposed system overcomes this weakness by transparently redirecting suspicious network traffic to the honeynet. An attacker has no way of knowing if the system they are attacking is the true production DMZ or the shadow DMZ.

2.1.2 High-Interaction Honeypots

The Honeynet project is a group dedicated to continued research of honeynets and the tools associated both with implementing new honeynets and monitoring traffic within the honeynets. According to the honeynet project, high-interaction honeypots are "complex solutions as they involve real operating systems and applications"[12]. Whereas a low-interaction honeypot emulates services, a high-interaction honeypot offers fully installed software and operating systems with no limitations. While this can provide more valuable data, it has an added risk of providing attackers a platform from which to launch further attacks if they are able to compromise the honeypot.

According to Lance Spitzner, a honeynet is "an architecture, an entire network of computers designed to be attacked"[12]. The concept behind a honeynet is a highly controlled architecture that provides an attacker with a valuable asset to target while also providing the defender with an enhanced logging and control capabilities.

The two key principles behind any honeynet are data control and data capture. Data control is important for limiting the risks of a honeynet and is accomplished by limiting the actions attackers can take once they compromise one of the machines in the honeynet. Data control is best accomplished by deploying a gateway through which any data must pass either entering or leaving the honeynet.

Data capture is central to any honeypot, as the reason for deploying any honeypot is to log the activities of an attacker. Data capture is often accomplished in several layers, with packet captures at the gateway as well as local logging capabilities on each of the honeypots. The proposed system follows all of the principles of a high-interaction honeypot. Data control is achieved at the gateway to the shadow DMZ. When an attack is detected, all further connec-

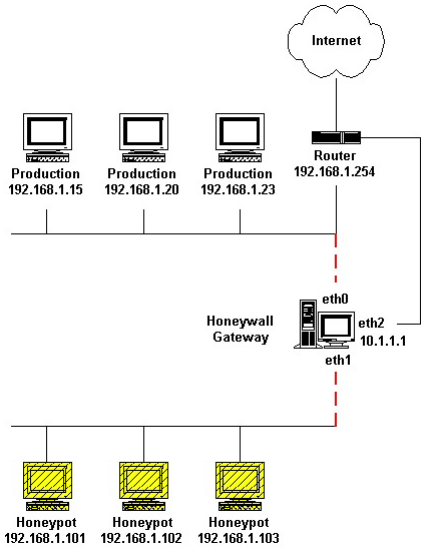

Figure 1: Honeywall network diagram

tions from the attacking IP are blocked to prevent any further damage. Data capture is achieved at the external gateway, as well as the shadow gateway. In further expansions of the proposed system, it would be possible to achieve data control at many other levels which would allow for greater correlation of data in the event of an attack.

A new principle is introduced in the proposed system with the requirement that the system must be capable of gracefully recovering from an attack. As described earlier, a graceful recovery implies that there is both an image of the system that can be used to analyze the actions of the attacker and a process that returns the compromised system to a safe state in order to minimize the impact on legitimate users.

The Honeynet Project maintains a pre-built honeynet called Honeywall which provides all the necessary resources for an easy to deploy honeynet. Honeywall comes in an ISO built on CentOS and is pre-configured with a number of tools for data-control and data-capture.

Figure 1 shows a typical architecture of Honeywall with the external gateway controlling the traffic that passes to the honeynet and production networks. Data control is accomplished at the Honeywall gateway with Snort-inline. The gateway does not restrict traffic as it enters the honeynet but does limit the number of connections that can be made from within the honeynet to any system outside the honeynet. If attackers take control of one of the systems inside the honeynet, they cannot use it to launch further attacks. Data capture is accomplished at several layers. At the gateway, Snort can be configured to log traffic that enters and leaves the honeynet. One of the most powerful monitoring tools available in Honeywall is Sebek which is a kernel module that uses the

principles of a rootkit. Sebek silently logs all local activity on the system on which it is installed and sends the logs over UDP to the Honeywall gateway where it is stored with logs from other systems and logging mechanisms. An extra network interface can be added to the gateway for a connection that is only accessible by a management system.

The Honeynet Project's Honeywall provided me with valuable insight into how to construct the architecture for my system. The proposed system works similarly to Honeywall with several key distinctions. In the proposed system there is currently no system to centrally correlate data, but it would be relatively simple to expand the proposed system to allow for central correlation of data. In addition, the gateway in Honeywall transparently forwards all traffic directly to the honeypots. The proposed system takes the design of Honeywall and puts the production DMZ behind an additional interface on the gateway. The gateway then transparently uses Snort-inline similarly to how it is used on the Honeywall gateway, but with the addition of the bait-and-switch preprocessor to transparently route traffic to the correct DMZ.

Honeypots, both low and high interaction, provided a significant base on which the proposed system is built. Many of the original principles of honeypots persist in the proposed system, while new principles are added in order to achieve the necessary goals.

2.2 Intrusion Detection and Intrusion Prevention

An intrusion detection system (IDS) is a system that is designed to capture intrusion attempts so that measures can be taken to limit damage and prevent future attacks. This is typically accomplished by sending alerts anytime the IDS detects an attack. IDSs can be broken down by where they gather their data and how they check for attacks. An IDS can be either host-based or network-based, meaning it either gathers data by listening to network traffic (network-based) or analyzing logs and other activities on a local machine (host-based). In addition, a host-based or network-based IDS can be either rule-based or anomaly-based. A rule-based IDS compares data against a predefined set of rules to detect attacks. This has the drawback of not being as capable of detecting the latest vulnerabilities. With an anomaly-based IDS it is necessary to establish a baseline of activity before final deployment. The IDS then alerts anytime it detects activity outside of this baseline. This can help in detecting new attacks, but it can be difficult to keep the baseline up to date with what should be expected, which can lead to an increase in false-positives.

In an article on the evolution of IDSs, Neil Desai discusses intrusion prevention systems (IPSs) at different layers of networking. Intrusion prevention systems further the functionality of IDSs by actively responding to attacks. An IPS is "any device (hardware or software) that has the ability to detect attacks, both known and unknown, and prevent the attack from being successful."[2] When an IPS detects an attack the typical reaction is to write the detected activity to a log for a reference and drop the connection associated with the attack. An inline network intrusion prevention system sits between the network

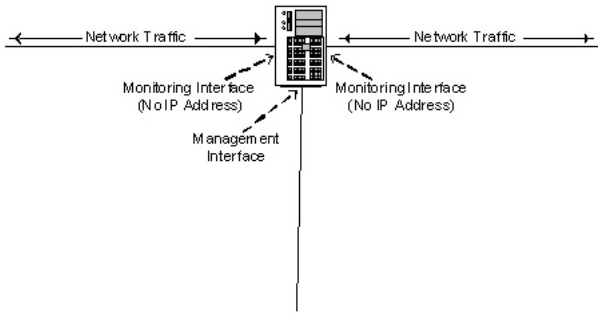

Figure 2: Inline Network Intrusion Prevention System

to be protected and the outside network, silently monitoring the traffic passing between the networks.

An inline intrusion prevention system has at a minimum two network interfaces. As seen in Figure 2, these interfaces are typically bridged and are not given an IP address so that an attacker does not have any way of targeting the IPS. While inline IPSs such as Snort-inline are highly capable of catching known exploits, they are not as successful at handling new and unknown attacks. Even anomaly based inline IPSs are only effective when enough information is known about the protocol to recognize activity that is out of the ordinary. This is the leading cause of false-positives as well as false-negatives. When an IPS is tuned to be more capable of catching newer attacks, it is likely to drop connections that are legitimate. At the same time, many administrators counter this by tuning the IPS to be less restrictive which leads to missing attacks that should be recognized.

The proposed system uses the Snort-inline intrusion prevention system with a preprocessor that was originally created by the developers as a "cheap parlor trick" with the hope that it could someday be used for more effective purposes. Snort-inline is used on the outward facing gateway in order to determine which DMZ traffic should be routed to. With further research it may be possible that IPSs could further evolve to the point where the bait-and-switch capability is found as a common option in products from most IPS vendors.

2.3 An Application Based Hybrid Honeynet

In an article entitled "Detecting Targeted Attacks Using Shadow Honeypots," Anagnostakis et al discuss a system they call a shadow honeypot which is used in parallel with the IDS in order to further analyze the traffic and make a decision on how to handle it. When traffic passes through the IDS, anything that "is considered anomalous is processed by a 'shadow honeypot' to determine the accuracy of the anomaly prediction" [1].

The shadow honeypot discussed in this article is similar to the honeypots previously discussed, except that it is an application honeypot. The honeypots

tested were modified versions of the Apache web server and the Mozilla Firefox web browser which had additional logging capabilities as well as rollback capabilities in case an attack was launched that changes the platform. The Apache web server was deployed on a server farm in parallel with the production Apache web servers. Any anomalous traffic detected by the IDS was sent to the honeypot Apache web server and then underwent further analysis. If it was discovered that the traffic only appeared suspicious but was actually legitimate, the traffic was processed by the honeypot as if it was in the production system. If the traffic was an attack and any changes were made, the traffic was cut off and those changes were rolled back.

This system was useful in reducing false-positives, but because it was built as an application level honeypot, it would have to be individually rebuilt for each application that is to be monitored. This would have required extensive customized code for each application as well as a way to coordinate all of the information once it reached a certain level. In addition, the system from this article was limited in its scope, as it could only detect attacks against Apache or Mozilla Firefox.

The proposed system uses a similar concept to fool an attacker into attacking a honeypot system rather than the production system, but can be used in more general cases. The architecture from Anagnostakis et. al. only works with applications that have been customized to perform more extensive logging and act as honeypots which could be a significant limitation. By creating the shadow honeynet at the operating system level it is possible to deploy any application or operating system as a honeypot with relative ease.

In addition to articles directly exploring honeypots, the Honeynet Project has published a number of articles discussing the uses of honeypots, including how a honeypot is used in a forensic analysis. When dealing with a forensic analysis of a compromised system, the amount of live data that can be captured can prove to be crucial in determining how an attacker got in and what actions were taken. The more monitoring that is available to capture activity, the easier it is to follow an attacker's activities. This includes local logs as well as logs from the IDS and IPS. It is also important that the data collected is archived for a period of time, as "most attacks involve some type of information gathering before the attack is launched"[4]. The information gathering can take place immediately before the attack or can occur over a more extended period of time so as not to attract attention.

In the proposed system the extra monitoring in place provided data at every level employing the honeypot principle of data capture in order to to make a forensic analysis possible. In addition to the logs that are kept on each system, logging is also performed by Snort-inline at the external gateway, as well as the Snort IDS on the shadow gateway. Although it is still a manual process, the correlation of this data provides a detailed trace of the attacker's actions. The external gateway provides the data detailing why the attacker's traffic was initially sent to the shadow DMZ, while the shadow gateway reveals which data from the attack caused the detection rule to fire. In the future, with the addition of a central log management server, it would be much easier to correlate all of

this data in order to provide a more detailed timeline of the attacker's activity.

A related thesis by Yatish Mamniya provides an analysis of the false-positive rate in Snort as an intrusion prevention system (Snort-inline) versus Snort as an intrusion detection system. In the thesis, Mamniya discusses the rate of false-positives that occur when running Snort inline, as well as the need for fine tuning in an IPS. Mamniya's architecture has an external gateway running Snort inline and iptables, as well as running Snort IDS on an internal gateway. This made it possible to determine the traffic entering the external gateway in relation to the traffic entering the internal gateway.

Mamniya measured several factors related to the performance of an IPS, including the false-positive rate. A tool called the Distributed Internet Traffic Generator (D-ITG) was used to generate legitimate traffic. Any traffic from D-ITG that was dropped by the IPS was deemed a false-positive. The results of Mamniya's research showed a false-positive rate of over eighteen percent in a tuned installation of Snort-inline. While I did not use this number as the baseline for may research, it provided valuable insight into the nature of the false-positive problem in IPSs.

The proposed system takes what was learned from this project with the goal of reducing this false-positive rate and minimizing the amount of fine tuning that is necessary in the IPS. By reducing the amount of fine tuning that is necessary, intrusion prevention systems could become more effective in a corporate environment as it would be easier to keep up with evolving attack trends.

While there was minimal research relating to the use of a hybrid honeynet to reduce false-positives in, significant work has been done with all of the components that would go into such a system. By understanding these tools and the principles that guide there use, it was possible to develop an architecture as well as a set of principles and methodologies for the proposed system.

3 Architecture

Through the three phases of testing, the proposed system had two architectures. The first phase was guided by the research into intrusion prevention systems, and represented a typical network with a DMZ that connected the outside world to the private network through an external intrusion prevention system. In the second and third phase of testing the architecture was more representative of the Honeywall gateway from the Honeynet project with several additions. The architecture included the production DMZ, the private network behind the DMZ, as well as the shadow DMZ that received suspicious traffic.

The first step of this research was to set up a test system that would be sure to exhibit false-positives but still be representative of an enterprise network. This included an application that users could interact with, a database on the back-end, and a separation of privately and publicly available networks. As seen in Figure 3, the architecture required multiple layers in the network, as the DMZ and private networks both ran on a VMWare ESXi server. There was a basic architecture that existed in all the phases of the testing. On the

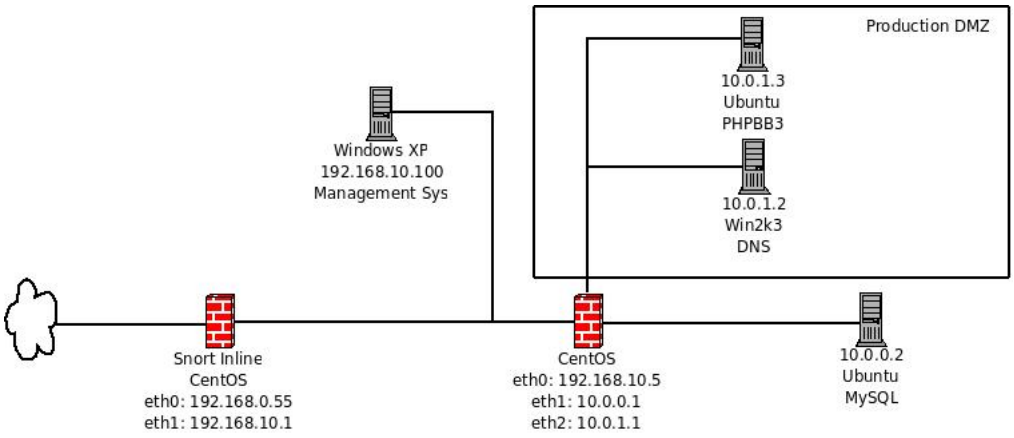

Figure 3: Phase 1 architecture

outer perimeter of the network I set up a firewall running on CentOS 5. This machine served as the gateway between the outside world and the internal network. For testing purposes, the outside world was represented by any machine in the 192.168.0.0/24 subnet. The first level of the internal network was the 192.168.10.0/24 subnet. This subnet contained the management system, both ESXi servers, and the external interface for each of the DMZ (second-level) gateways. Each DMZ network had a gateway machine which acted as a firewall, NAT router, and ran Snort as an IDS. Within the DMZ was a DNS server to answer queries for the internal network, and a web server running phpBB and a PHP script that opened the system to a directory traversal attack. A MySQL server was set up in the private network as well as a Linux client machine. Connections to the private network were restricted by the second-level gateway. Between the three phases of testing changes were made to the base system to reflect the required environment.

3.1 Phase One

The goal of this phase was to create an environment that was somewhat representative of a typical corporate environment and configure the IPS to generate false-positives on a given set of traffic. This allowed me to establish a baseline for testing. Phase one of testing most closely reflected the base setup as seen in Figure 1. Traffic from the outside world was NATed to the 192.168.10.0/24 subnet and was forwarded to the gateway at 192.168.10.5. Any traffic that reached the second-level gateway was NATed to the 10.0.1.0/24 and 10.0.0.0/24 subnets with the former being the DMZ and the latter being the private network behind the DMZ. Connections to the private network were limited by the gateway with traffic to the database server only being permitted if it originated from the web server.

The primary difference from the base configuration was the setup of Snort-inline on the external gateway. In phase one of testing the gateway was configured as a typical IPS. Any traffic passing through the gateway that matched a rule in Snort-inline was dropped. The rules used in Snort-inline were a custom set of rules designed to create false-positives when a user interacted with PHPBB. Based on the custom set of rules, any attempted connection to index.php script of PHPBB would result in all subsequent traffic from the connecting IP being dropped. While these rules would not likely be found in a corporate network, they are representative of an instance where a systems administrator might develop a poorly built set of rules that could at least for a period of time lead to a significant number of false-positives.

In addition to Snort-inline, the gateway also ran IPtables which was used in conjunction with Snort-inline, as well as to perform NAT in order to translate external IP addresses to the mid-layer private network addresses. Initially IPtables did not serve any purpose other than NAT. In between the external gateway and the DMZ gateway I set up a Windows XP machine solely for managing the ESXi servers. This machine was not accessible from the outside world and did not have any bearing on the results of the tests. I set up a second gateway on the DMZ that performed a second level of NAT from the 192.168.10.0/24 subnet to the 10.0.1.0/24 subnet. Two levels of NAT were necessary for this system in preparation for the second phase where there would be two DMZs on separate subnets. The DMZ gateway served as both a router and a firewall to protect the private network. All traffic entering the gateway was either redirected to the DNS Server on port 53, the web server on port 80, or else it was dropped. Direct connections to the private network were only permitted if they originated from the web server and were destined for the database server on port 3306. Any other connections directly to the private network were blocked by the firewall. All connections from the private network to either the machines in the DMZ or any machine in the outside world were permitted. Phase two expands the architecture used in phase one in order to reflect the proposed system.

3.2 Phase Two

The architecture in Figure 4 built upon the first with some significant modifications. Snort-inline was reconfigured to use the bait-and-switch preprocessor in order to reroute traffic based on the rules in Snort. The shadow DMZ was set up on the 10.0.2.0/24 subnet and was also connected directly to the external gateway. The main principle was to set up the shadow DMZ to mirror the production DMZ as closely as possible. This was simplified through virtualization, as it was possible to create clones of the web server and DNS server as well as migrate the copies to the shadow DMZ.

The gateway for the shadow DMZ had an external interface of 192.168.10.6 which was connected to the external network. It also had an internal interface that provided connection to the web server and DNS server inside the DMZ. The DNS server was authoritative for the *.thesis.local domain and answered any requests that were forwarded by the external gateway. The web server had

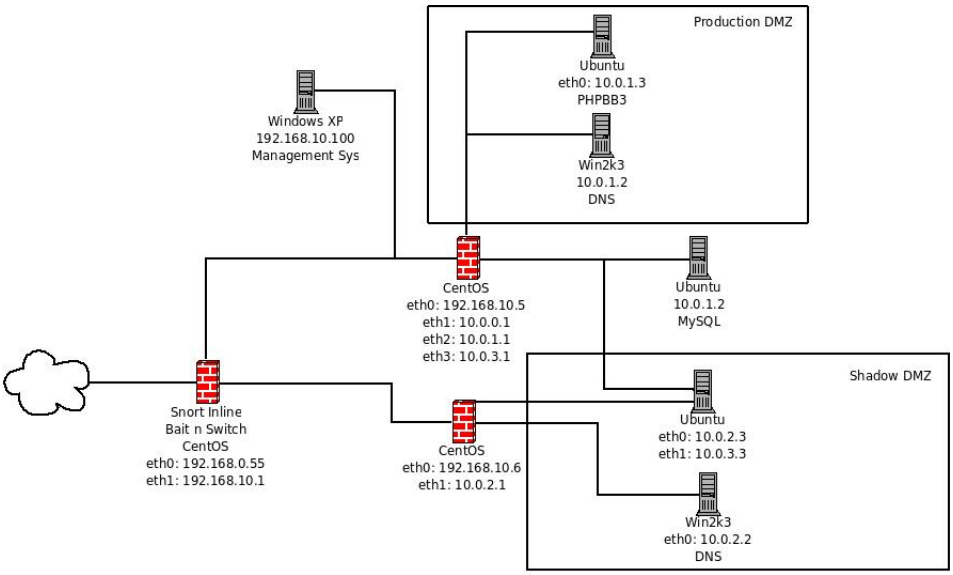

Figure 4: Phase 2 architecture

two interfaces, one at 10.0.2.3 and the other at 10.0.3.3 on the 10.0.3.0 subnet. The second interface was a connection to the gateway on the production DMZ and provided the connection between the shadow web server and the database server in the private network. The firewall on the production DMZ gateway was reconfigured to allow direct connections to the private network from both the production and shadow web servers.

Snort was set up on the shadow DMZ gateway in order to monitor traffic in the DMZ. The goal of this IDS was to monitor traffic exiting the network, and the rules were specifically designed to catch data that would be typically seen in a compromised network. In this specific architecture this was a rule that looked for a request for the /etc/passwd file. Specifically, the rule looks for the pattern 'root:0:0' which is a pattern that always exists in the /etc/passwd file. To ensure that this match is the result of a directory traversal attack on a web application, there is also a match in the rule requiring a 'GET' request. Although this is a basic case, the purpose was to demonstrate certain features of the proposed system and could easily be expanded to look for any other sensitive files, as well as credit card numbers, connections on abnormal ports, or username/passwords. In addition, because the shadow DMZ was set up so that maintenance would simply be performed by re-cloning the machine from the production DMZ, a written policy was established that no remote connections should be made to the machines within the shadow DMZ. Policies such as this could be implemented in the Snort rules to detect RDP, ssh, XTerm, and other remote connections as attacks. A perl script was set up to run on an infinite loop in order to monitor the log from Snort looking for any new alerts and perform

the necessary functions if one is added, as will be discussed in the next section.

Because of the use of NAT, attackers, or even regular users should have no idea whether they are interacting with the production DMZ or the shadow DMZ. It is likely that if this was used in an enterprise network, the machines in the shadow DMZ would have less resources than the production machines. This should not typically affect the performance, however, as the shadow DMZ should see much less traffic than the production DMZ depending on how the rules are set up. Through virtualization it would be possible to build the shadow DMZ on one piece of hardware which would greatly minimize the cost of such an environment.

4 Methodology

In this chapter, I present the proposed methodology for reducing false-positives as well as the steps that were taken to test the capabilities of the system. The first section covers the different aspects of the proposed system as well as how it differs from a typical honeypot or IPS.

4.1 Hybrid Honeypot

The proposed system combines firewalls, intrusion prevention and detection systems, and honeypots in order to achieve a lower false-positive rate than a comparable intrusion prevention system. Before describing how the system works, it is important to analyze exactly how each of the tools was used within the network and why it was necessary.

4.1.1 Firewalls

There were several firewalls used in multiple layers in this network. On the outer gateway that was between the outside world and the DMZ gateways was the primary firewall. Due to some issues with virtualization there were two gateways which sat inline with each other. The outer gateway ran IPtables and had the sole function of translating traffic from the 192.168.1.0/24 network to 192.168.10.5 which was the IP for the external interface on the production DMZ gateway. All traffic passed through the second gateway which had bridged interfaces with no IP. IPtables on this machine was used in conjunction with Snort-inline and initially was configured with a basic accept all rule. The DMZs used IPtables to perform NAT from the 192.168.10.0/24 network to their respective internal networks. Filters were set up on both DMZ gateways so that connections to the DMZs were only permitted on the required ports, in this case 80 for the web-server and 53 for the DNS server. In addition, the production DMZ was set up to filter connections to the private 10.0.0.0/24 network. The rules in this DMZ blocked all connections to the private network except for connections from either the production web-server (10.0.1.3) or shadow web-server (10.0.3.3) to the database in the private network over port 3306.

```
preprocessor bait-and-switch: max_entries 200,log,insert_before
preprocessor bait-and-switch-ignorehosts: 10.0.0.0/8
```

Figure 5: snort_inline.conf bait-and-switch configuration

```
drop tcp $EXTERNAL_NET any -> $HOME_NET any (classtype:misc-attack; msg:"False positive1.";content:"GET";content:"/phpBB3/styles/prosilver/template/styleswitcher.js"; bait-and-switch:600,src,192.168.10.6; sid:100001;)
drop tcp $EXTERNAL_NET any -> $HOME_NET any (classtype:misc-attack; msg:"False positive2.";content:"GET";content:"/phpBB3/styles/prosilver/imageset/site_logo.gif"; bait-and-switch:600,src,192.168.10.6; sid:100002;)
drop tcp $EXTERNAL_NET any -> $HOME_NET any (classtype:misc-attack; msg:"False positive3.";content:"GET";content:"/phpBB3/styles/prosilver/imageset/icon_topic_latest.gif"; bait-and-switch:600,src,192.168.10.6; sid:100003;)
drop tcp $EXTERNAL_NET any -> $HOME_NET any (classtype:misc-attack; msg:"False positive4."; content:"GET";content:"/phpBB3/styles/prosilver/theme/images/bg_header.gif"; bait-and-switch:600,src,192.168.10.6;sid:100004;)
```

Figure 6: bait-and-switch rule configuration

4.1.2 Intrusion Prevention/Detection System

In this network I ran both an intrusion prevention system and an intrusion detection system. The IPS, Snort-inline, was installed on the external gateway. For Snort-inline I created a custom set of rules, as seen in Figure 6 aimed at creating clear false-positives. In addition to the custom set of rules, I configured the IPS to use the bait-and-switch preprocessor by modifying the snort_inline.conf file.

The first preprocessor setting tells Snort the maximum amount of entries, to log dropped packets, and to insert the new firewall rules created by the preprocessor before all other firewall rules. This setting ensures that any traffic from the attacker is dropped. The bait-and-switch requires an added definition in each of the rules that are to be used with the preprocessor which tells it where traffic should be rerouted to.

Snort-inline with bait-and-switch works by checking network traffic against the chosen set of rules and either sending the traffic to its original destination if a rule is not matched, or rerouting the traffic to the shadow DMZ if a rule is matched. The rerouting is accomplished by sending an SNAT and DNAT rule to the firewall installed alongside Snort-inline. This new rule specifies that any connection from the originating IP of the suspicious traffic should be rerouted to the shadow DMZ gateway at 192.168.10.5. Because traffic is routed to one of the two DMZs, there is no case where a connection can be blocked as a direct result

alert tcp 192.168.10.6 80 -> any any (msg:"Successful directory traversal"; content:"root:x:0:0:root:/root"; sid:100050;) alert tcp 192.168.10.6 80 -> any any (msg:"Successful directory traversal"; content:"root:x:0:0:root:/root"; sid:100050;)

Figure 7: Snort Attack Detection Rule

of a rule firing on the IPS. In Figure 6, the snort_inline rules are modified to include the setting 'bait-and-switch:600,src,192.168.10.6'. Bait-and-switch tells the rule engine which configuration setting this is. The second setting is the number of seconds that the firewall rule should exist, in this case 600 seconds. The next setting tells Snort-inline to create an SNAT and DNAT rule and which IP the traffic should be rerouted to.

An IDS was also deployed on the shadow DMZ gateway listening on the internal interface. The purpose of this IDS was to analyze traffic leaving the shadow DMZ for patterns that are common to a successful attack on a network.

The above example rule was the only attack rule I used in my IDS. This rule looks through network traffic for the pattern "root:x:0:0:root:/root" with a source IP of 192.168.10.6 and a source port of 80 leaving the network. This source IP and port means an alert should fire for any web traffic with this pattern leaving the shadow DMZ for any external IP. The pattern chosen is representative of text found in all /etc/passwd files. Alerts from this gateway are how the network determines if a successful attack has taken place, which leads to reactionary measures being taken. Other possible rules could look for remote connections, cross-site scripting attacks, shell code and more. This is safer than the drop rules at the IPS because the traffic must go through several layers and multiple matches before it is dropped. Only when this rule is matched, is the attacker cut off from the network.

4.1.3 Handling Network Traffic

In this section I will outline the steps that were taken by the proposed system for legitimate traffic, suspicious traffic, and attack traffic.

When legitimate traffic reached the external gateway it was NATed from the outside world (192.168.1.0/24) to the first layer of the internal network (192.168.10.0/24). Traffic then transparently passed from the external gateway through the second gateway which was running Snort-inline. If the traffic did not match any of the rules that Snort-inline was using, it was forwarded to the gateway in the production DMZ at 192.168.10.5 where it was NATed to the 10.0.1.0/24 network for either the web-server or the DNS server.

Suspicious traffic and attack traffic began at the same stage as the legitimate traffic being NATed from the outside world to the 192.168.10.0/24 network. It is important to note that attack traffic is classified as suspicious traffic when it first enters the network, thus it is treated the same as all suspicious traffic. The distinction between suspicious and attack traffic is only made when the traffic leaves the shadow DMZ. When suspicious and attack traffic passed through

After trigger
DNAT all – <suspicious_ip> <external IP> <shadow DMZ>
SNAT all – <shadow DMZ> anywhere <external IP>

Figure 8: Bait-and-Switch example

Snort-inline on the second gateway, they were matched against one of the IPS rules. As seen in Figure 8, the bait-and-switch preprocessor then created a new rule in IPtables to NAT all traffic from the originating IP to the 192.168.10.6 address which was the external interface of the gateway in front of the shadow DMZ.

In the case of suspicious traffic for the DNS server, it was simply rerouted to the shadow DNS server in the same manner as it was in the production DMZ, which then answered the DNS requests. Suspicious traffic to the web server arrived at the 10.0.2.3 interface which then made all database requests through the 10.0.3.3 interface. These requests went through the production DMZ gateway and were forwarded to the MySQL server.

As mentioned earlier, attack traffic is handled differently than suspicious traffic only in the final phase. When an attacker succeeds in exploiting a vulnerability and the IDS at the shadow gateway detects a pattern in the traffic passing from the shadow DMZ to the attacker, several steps are taken. A basic perl script was used which continuously read the alert file that was generated by the IDS. The script simply checked the alert file for SID 100050 which was a directory traveresal attack and then took action to cut off the attack.

The perl script took three steps. The first step taken by the perl script was to send a new rule to the Snort-inline gateway with a block rule to drop any traffic originating from or destined for the attacker IP. This step prevents the attacker from making any further connections to either the shadow DMZ or the production DMZ. Blocking traffic destined for the attacker IP prevented the compromised machine from sending any data to the attackers if they installed any sort of program that was meant to track user data.

The second step taken by the perl script was to take a snapshot of the compromised virtual machine. The purpose of the snapshot was to provide valuable data for determining what actions the attacker took to compromise the machine. This data can prove valuable for stopping future attacks.

The final step was to revert the compromised machine back to a known safe state. This allowed legitimate users to continue interacting with the system with minimal downtime. This extended process not only reduced false-positives in the production network, but also made it possible to monitor a successful attack without endangering user information.

4.2 Testing

I tested the proposed system for both false-positives and false-negatives in two phases each. The goal of the first scenario was to develop a set of network

traffic that would be used for both phases of testing and to determine the rate of false-positives and false-negatives for that set of network traffic.

4.2.1 False Positives

In the first phase of testing I used the initial architecture of the system. For this test I requested '/phpBB3' which gave the main page of the forum application. Using the custom Snort rules, I ran tcpdump on the external gateway and on the production DMZ gateway. By comparing the packet capture in the external gateway to the packet capture in the production DMZ gateway, I was able to determine the rate of false-positives in the setup with Snort-inline. The results showed that out of 23 network packets sent to the network, 13 were dropped by Snort-inline. While this is a result of the rules that were used, which were created to demonstrate a high rate of false-positives, it is not inconceivable that an administrator might create a set of rules that would cause a higher rate of false-positives.

In the second phase of testing I made the same requests for '/phpBB3' against the proposed architecture. In using this method I was able to ensure that any differences in the false-positive rate were a direct result of the hybrid honeypot. To determine the rate of false-positives, I again ran tcpdump on both the external gateway and the production DMZ gateway, as well as the shadow DMZ gateway. By comparing the traffic on the external gateway to the traffic that passed through the gateways of both DMZs, it was possible to determine if any traffic was lost. A false-positive would be a case where network traffic was dropped without a block rule being created by a successful attack.

In addition to testing for false-positives it was also necessary to determine the capabilities of the system to gracefully handle a successful attack. In both DMZs I had a script, 'exploit.php', that was meant to simulate a script in an application that could be exploited to reveal files on the remote host. Part of the network traffic from the first phase was to make a GET request for 'exploit.php' and set the 'filename' parameter to ../../../../../../etc/passwd. While I was able to read the /etc/passwd file, I was unable to make any subsequent requests to the desired resources. This test showed that the system was able to prevent an attacker from taking advantage of a successful attack by creating a block rule on the external gateway. After all the network traffic had been replayed, I opened the VM information for the shadow DMZ gateway to determine the state of the virtual machine. The system successfully created a snapshot of the exploited VM called 'Exploited' and reverted the VM back to the known good state snapshot.

4.2.2 False Negatives

I used Nessus, a security scanner, to test for false-negatives. I chose a scan policy that would create a significant amount of traffic that represented several forms of attacks including directory traversals, cross-site scripting, sql injection, and remote code execution. In advance, I treated all traffic from Nessus as

false-negatives to more easily distinguish the results. I performed the testing in two phases, as I did in the test phase for false-positives.

In the first phase of scanning, I chose the following plugin families to ensure that the systems were properly detected and to create false-negatives.

1. Backdoors
2. CGI abuses
3. CGI abuses : XSS
4. DNS
5. Web Servers

The CGI and DNS plugins were chosen because they mostly target ports 53 and 80 which are the only ones that are open on the DMZ gateway. The CGI plugins also have a large number of plugins that actually attempt an exploit which would be the best way to trigger a false-negative. The backdoor plugin family was chosen to generate more attack traffic. In addition to these plugin families, I also enabled web application testing which attempts cross-site scripting and sql injection attacks against the web applications on the target system.

In order to determine the significance of false-positives in the system, I ran Wireshark on the Nessus scanner, and tracked the number of alerts generated by the Snort-inline gateway. In both phases of testing, there were around 50000 packets sent to the target systems. Of the packets that were generated by the scanner in the first round of testing, a total of arund 10000 alerts were triggered on the gateway. Because of the set up of these system, the remaining packets are considered false-negatives, though a significant number could be from version detection plugins which would not generate an alert.

The second round of false-negative testing was performed by using Nessus to ensure the same services and vulnerabilities were detected by running the same scan policy in its second phase of the architecture. I this second round of false-negative testing, a total of around 12000 alerts were triggered on the gateway resulting in less false-negatives. Although it would seem that the same number of alerts should be generated between the two systems, with some of the CGI and web application testing against the first phase, the connection would likely have been dropped once the attack had been detected which is not the case in the second phase. In addition, the web application testing does not always create the same values for parameters, which could potentially cause small differences in the number of alerts generated.

4.3 Differentiating Between Suspicious and Attack Traffic

An important detail that has not been discussed completely is how we differentiate between suspicious traffic and attack traffic. Any traffic that ends up in the shadow DMZ is automatically considered suspicious traffic. All future traffic from the originating host is considered suspicious for ten minutes, though

```
DNAT all – 192.168.0.108 192.168.0.55 192.168.10.6
SNAT all – 192.168.10.6 anywhere 192.168.0.55
```

Figure 9: IPtables NAT rules

this can be configured as needed for the desired environment. The determination of whether traffic is attack traffic is made on the shadow DMZ gateway. This system has Snort installed with rules based on common patterns seen in attacks. In the case of my test system, it was set up with only one rule designed to detect a successful directory traversal attack where the attacker retrieves the '/etc/passwd' file from the target system. When an attack is detected, as discussed previously, a script which runs in a continuous loop automatically sends a block rule to the external gateway to block the attacker IP, and takes a snapshot of the compromised machine. A log was kept to track the Snort logs which were used to determine which exploit the attacker used. Because of the design of the rules on the shadow DMZ, any traffic that triggered a rule was assumed to be an attack. This is a relatively safe assumption as it required the traffic to trigger a rule on both the external gateway in order to be considered suspicious, as well as trigger a rule on the shadow gateway to be considered an attack. It was unlikely that legitimate traffic would trigger rules on both gateways, especially as the rules on the shadow gateway were typically only seen in the event of a successful attack.

5 Results and Future Work

In this section we analyze the results from the testing and look at how this system could be used as a basis for future work.

5.1 Results

The results of the testing were based on two scenarios. The first scenario was legitimate traffic that appeared suspicious due to a poorly written set of rules. The second scenario was meant to represent an instance where an attacker finds a directory traversal vulnerability in a PHP script that could be exploited to read arbitrary files on the affected host.

Scenario one represented a case in which a set of seven rules were created to block access to certain portions of the phpBB forum application. In the first phase of testing, out of the 23 packets sent to the network, 13 were dropped by Snort-inline. This amounted to a false-positive rate of over fifty percent. All traffic in this phase that was deemed suspicious was immediately dropped. In the second phase of testing, the same traffic was replayed against the network. The first set of packets did not match against the Snort rules and were therefore routed to the production DMZ. After the initial set of packets, a request was made for a phpBB page which caused a Snort-inline rule to fire. The result was the addition of a new NAT rule in IPtables, as can be seen in Figure 9.

INPUT Table
DROP all – 192.168.0.108 anywhere
FORWARD Table
DROP all – 192.168.0.108 anywhere

Figure 10: IPtables rules to cut off the attacker.

This rule caused all traffic from the IP 192.168.0.108 to be routed to the shadow DMZ. One flaw that I encountered was that after the new rule was created, I had to reload the page in order for it to be successfully loaded. I suspect this was due to the shadow web server not being aware of the existing connection. Despite this flaw, after reloading the page any further interaction worked as if the connection was being routed through the production DMZ. Because of the requirement to reload the page, a total of 55 packets were sent to the test network. Of these 55 packets all were routed to either the production or shadow DMZ with none being dropped. Although a page reload was required, the results still demonstrate a zero percent false-positive rate. These results could be achieved whether using the rule set from this test, or another rule set that is customized to include the bait-and-switch configuration.

The second scenario was designed to emulate a situation where an attacker discovers and exploits a directory traversal vulnerability to gain access to the '/etc/passwd' file on the web server. In the first phase of testing, the request for '/exploit.php?filename=../../../../../../../../etc/passwd' caused the packet to be dropped. This was typical of an IPS and successfully prevented the attack. The issue, however, was how to prevent or mitigate the effects of this attack without preventing legitimate traffic.

The second phase of testing demonstrated that it was possible to mitigate the effects of the attack. Again, as in the second phase of testing for the first scenario, the request '/exploit.php?filename=../../../../../../../../etc/passwd' matched on a rule in Snort-inline and caused a new NAT rule to be added to the firewall that ran alongside Snort-inline. At this point, the request was classified as suspicious traffic and was routed to the shadow DMZ. The request reached the shadow web server which processed the request and returned the contents of the /etc/passwd file. As the contents of the '/etc/passwd' file passed through the shadow gateway, the Snort rule to detect the contents of the /etc/passwd fired and wrote to the alert.log file. The snortwatch.pl script detected this new alert in the alert.log file and initiated a graceful recovery from the attack. Two new IPtables rules were added as the first rules on the outermost Firewall, as can be seen in figure 10.

This immediately prevented attackers from taking advantage of the successful attack by cutting them off from the network. The second set created a snapshot of the compromised web server with the name 'compromised'. The final step was to revert the web server to a safe snapshot that was taken during the setup phase on all virtual machines. In total it took about one minute from the time of the attack to the completion of reverting to the safe snapshot.

During this time there was limited access to the shadow web server, though this is necessary to return the system to a safe state without losing the evidence of the attack. The final limitation is not as significant. I chose not to explore latency as it did not have any bearing on the rate of false-positives. Because of the way the system relies on an add-on tool, rather than a modification of rules, to reduce false-positives, it does not increase the rate of false-negatives. Any false-negative that exists in the researched system would also exist in an IPS without the added tools.

The result of the false-negatives testing showed that there was no impact on the rate of false-negatives caused by the new system. In the first test phase I saw a total of around 10000 packets that were determined to be false-negatives. In the second phase of testing, with identical traffic, I saw close to the same number of alerts, in this case 12000, logged which led to an lower rate of false-negatives, thogh as mentioned earlier in the paper, this could have several causes and the rate of false-negatives between the two systems will be at a minimum equal. By running Nessus in both phases of testing, I was also able to determine that the same vulnerabilities were found between the two test phases. In addition, because none of the traffic was successful in obtaining the '/etc/passwd file', the graceful recovery process was never initiated, as this process should only occur if an attack is successful. As discussed before, the rate of false-negatives was unaffected because the same rules were used between the two test phases, and the only change was to include the bait-and-switch preprocessor.

The results demonstrated several achievements. Most significantly, the system reduced the false-positive rate to zero without having an effect on the rate of false-negatives. The results also showed a graceful process to recover from attack so that the exploit could be analyzed while legitimate users could safely continue to interact with the systems. While these results were promising, they presented only a start, as the proposed system could be expanded in many ways to provide better protection for the network, while also providing increased capabilities for monitoring attacks.

5.2 Future Work

While the results of testing were promising, there are several fields that these results could be applied to in order to improve the capabilities of the system. Due to the availability of the bait-and-switch preprocessor, the only IPS used was Snort-inline which is a rule-based intrusion prevention system and is therefore only as effective as the rules that are in place.

5.2.1 Further Testing

The tests I performed against this systems were only intended to demonstrate the capabilities of the systems. To further this research it would be interesting to set up a system, with a full set of rules, connected to the Internet and let attackers compromise the system. This would give a better idea of how much traffic would likely be routed to the shadow network.

Another aspect to explore in expanding this system would be to use a full set of prevention rules on a network and monitor the rules to determine which ones are causing the most false positives. This could be accomplished by only placing the bait-and-switch option in rules that are not exact attack signatures. If a lot of traffic enters the shadow network but does not cause a compromise, it may be necessary to re-examine the rule and modify it to reduce its chances of false positives. This could be most useful in a network where the security group creates custom rules that need to be tested, by allowing applying the rules against live traffic without risking blocking customer traffic.

A final avenue that I looked at briefly but decided was not necessary for the purposes of the research was to have two databases, one in the production network and one in the shadow network. The shadow network database would only contain public data and would not pose a risk if it were compromised. This poses several difficulties, especially in the synchronizing of data without exposing private data to the shadow network. There would have to be a way for users in the shadow network to interact with the shadow network without knowing they were working against a separate set of data. In addition to further testing and modification of the proposed system, there are several ways to expand its capabilities that could make it much more effective.

5.2.2 Anomaly Based IPS

As discussed earlier, an anomaly based IPS works by analyzing traffic against a baseline of network activity. The drawback of a system such as this is that, over time, typical network activity changes and it can be difficult to keep the baseline on level with what should be expected, leading to an increase in both false-positives and false-negatives. While this research demonstrated the possibility of significantly reducing false-positives in a rule based IPS, the same capabilities could be used to create a constantly evolving baseline in an anomaly based IPS. The baseline would initially be created in the same manner as is seen in a typical IPS. Any activity that is deemed anomalous would be rerouted to the shadow DMZ. This traffic would be further analyzed over a period of time to determine the nature of the traffic based on a certain set of criteria set forth by the security administrators. If after a designated period of time no attack was discovered, the baseline could be modified to take into account the anomalous traffic so that in the future it would no longer be considered anomalous. At the same time any attack traffic that is aimed at the network would be sent to the shadow network and would be handled in the same manner as in the proposed system. While it is feasible that an attacker could still penetrate this system, it would require a much more coordinated and sophisticated attack over a long enough period of time to make the attack traffic appear legitimate. This extended attack would leave a significant footprint that could be used for the purposes of tracking down the attackers.

5.2.3 A True Learning IPS

A rule-based IPS is only as effective as the rules that are built into the system. These rules are either static or slow to evolve due to the requirement of human involvement. Most rule-based IPSs can be bypassed by altering the code enough or passing different parameters so that the traffic does not match any of the rules in the IPS. This sort of attack becomes more likely when rules are constructed to reduce false-positives.

The proposed shadow honeynet could be extended to 'learn' from malicious traffic. As before, legitimate traffic would be treated the same as in the research. The 'learning' would occur in the shadow DMZ when an attacker successfully exploits a vulnerability. Similar to the tool Honeycomb, signatures would be added to Snort-inline based on results discovered after an attack. Honeycomb works specifically with honeyd to create signatures based on logged activity. Using the shadow honeynet, the data available for creating new rules would be much more extensive.

The process, which would work similar to the current system when an attack occurred, would use a central log correlation system. Following a successful attack, the script monitoring traffic exiting the network would query the central log system for all data relating to the attack. This data would be used to create a rule on the IPS which would monitor future traffic for similar patterns to those found in the logs. These rules could be applied to both the network intrusion prevention system that monitors traffic entering the network, but could also be used to create new rules on host based intrusion prevention/detection systems. By learning from attacks rather than just blocking the attacker, it is possible to prevent access to vulnerabilities in applications without preventing access to the application itself.

6 Conclusion

The research presented here demonstrates a proof of concept for a shadow honeynet that is designed to reduce false-positives while having a minimal, if any, impact on the false-negative rate. While a pre-chosen set of network traffic had a near 50% false-positive rate in the first test phase, the same network traffic sent to the network with the shadow honeynet had zero false-positives. In addition, I demonstrated the capability of the system to gracefully and automatically recover from a successful attack. A graceful recovery was defined as having a copy of the compromised system that could be used for a forensic analysis as well as returning the compromised system to a safe state so that it could be used by legitimate users. Being automated implies that the system was able to recover from the attack without any human interaction. The proposed system goes a long way in reducing false-positives without reducing false-negatives and opens up several avenues of expansion that could further improve the current systems of intrusion prevention.

References

[1] Agnostikas, K. G., Sidroglou, S., Akritidis, P., Xinidis, K., Markatos, E., & Keromytis, A. D. (n.d.). Detecting targeted attacks using shadow honeypots. Retrieved May 20, 2009, from Columbia University, Dept. of Science Web site: http://ics.forth.gr/dcs/Activities/papers/replay.pdf

[2] Desai, N. (2003, February 23). Intrusion prevention systems: The next step in the evolution of IDS. Retrieved May 20, 2009, from http://www.securityfocus.com/infocus/1670

[3] Honeynet Projct & Research Alliance. (2005, August 17). Know your enemy: Honeywall cdrom roo. Retrieved May 20, 2009, from Honeynet Project Web site: http://old.honeynet.org/papers/cdrom/roo/index.html

[4] Honeynet Project. (2000, May 23). Know your enemy: A forensic analysis the study of an attack. Retrieved May 20, 2009, from Honeynet Project Web site: http://old.honeynet.org/papers/forensics/index.html

[5] Honeynet Project. (2003, November 17). Know your enemy: Sebek a kernel based data capture tool. Retrieved May 20, 2009, from Honeynet Project Web site: http://old.honeynet.org/papers/sebek.pdf

[6] Honeynet Project. (2006, May 23). Know your enemy: Honeynets. Retrieved May 20, 2009, from Honeynet Project Web site: http://old.honeynet.org/papers/honeynet/index.html

[7] Kreibich, C. (n.d.). Honeycomb automated signature creation using honeypots. Retrieved May 20, 2009, from http://www.icir.org/christian/honeycomb/

[8] Mamniya, Y. (n.d.). Evaluation of snort inline as an intrusion prevention system [Data file]. Rochester Institute of Technology College of Computing and Information Sciences.

[9] Nepenthes readme. (n.d.). Retrieved May 20, 2009, from http://nepenthes.carnivore.it/documentation:readme

[10] Savage, P. (2005, August). Snort inline part I. Retrieved May 20, 2009, from http://linuxgazette.net/117/savage.html

[11] Savage, P. (2005, September). Snort inline part II. Retrieved May 20, 2009, from http://linuxgazette.net/118/savage.html

[12] Spitzner, L. (2003, May 29). Honeypots Definitions and Value of Honeypots. Retrieved May 20, 2008, from http://www.spitzner.net/honeypots.html

[13] The Snort Project. (March 5, 2009). Snort user manual (Version 2.8.4) [Data file]. Retrieved May 20, 2009, from Sourcefire Web site: http://www.snort.org/docs/snort_manual/2.8.4/snort_manual.pdf

A Snort-inline Rules Phase 1 - No bait-and-switch

```
# Custom made rules to generate false-positives.
    drop tcp $EXTERNAL_NET any -> $HOME_NET any (classtype:misc-
attack;
    msg:"False positive 1.";
    content:"GET"; content:"/phpBB3/styles/prosilver/template/styleswitcher.js";
    sid:100001;)
    drop tcp $EXTERNAL_NET any -> $HOME_NET any (classtype:misc-
attack;
    msg:"False positive 2."; content:"GET";
    content:"/phpBB3/styles/prosilver/imageset/site_logo.gif";
    sid:100002;)
    drop tcp $EXTERNAL_NET any -> $HOME_NET any (classtype:misc-
attack;
    msg:"False positive 3."; content:"GET";
    content:"/phpBB3/styles/prosilver/imageset/icon_topic_latest.gif"; sid:100003;)
    drop tcp $EXTERNAL_NET any -> $HOME_NET any (classtype:misc-
attack;
    msg:"False positive 4."; content:"GET";
    content:"/phpBB3/styles/prosilver/theme/images/bg_header.gif";
    sid:100004;)
    drop tcp $EXTERNAL_NET any -> $HOME_NET any (classtype:attempted-
recon;
    msg:"Directory traversal detected."; content:"GET";
    content:"../etc/passwd"; sid:100005;)
```

B Snort-inline Rules Phase 2 - Bait-and-switch enabled

```
# Custom made rules to generate false-positives.
    drop tcp $EXTERNAL_NET any -> $HOME_NET any (classtype:misc-
attack;
    msg:"False positive 1."; content:"GET";
    content:"/phpBB3/styles/prosilver/template/styleswitcher.js";
    bait-and-switch:600,src,192.168.10.6; sid:100001;)
    drop tcp $EXTERNAL_NET any -> $HOME_NET any (classtype:misc-
attack;
    msg:"False positive 2."; content:"GET";
    content:"/phpBB3/styles/prosilver/imageset/site_logo.gif";
    bait-and-switch:600,src,192.168.10.6; sid:100002;)
    drop tcp $EXTERNAL_NET any -> $HOME_NET any (classtype:misc-
attack;
    msg:"False positive 3."; content:"GET";
    content:"/phpBB3/styles/prosilver/imageset/icon_topic_latest.gif";
```

```
    bait-and-switch:600,src,192.168.10.6; sid:100003;)
    drop tcp $EXTERNAL_NET any -> $HOME_NET any (classtype:misc-
attack;
    msg:"False positive 4."; content:"GET";
    content:"/phpBB3/styles/prosilver/theme/images/bg_header.gif";
    bait-and-switch:600,src,192.168.10.6;sid:100004;)
    drop tcp $EXTERNAL_NET any -> $HOME_NET any (classtype:attempted-
recon;
    msg:"Directory traversal detected."; content:"GET"; content:"../etc/passwd";
    bait-and-switch:600,src,192.168.10.6;sid:100005;)
```

C Bait-and-switch configuration

bait-and-switch: Attempt to do stealthy reroutes of an attacker to a honeypot for x number of seconds

——————————————————————————

For use in rule language

reroute packets from attackers for x number of seconds because we don't like them messing with

our stuff.

#

In the example below the first line tells bait-and-switch a max amount of entries for memory allocation

In addition the first line tells bait-and-switch to log droped packets to the snort log dir bands.log,

and to insert reroute rules before anything else that may be in your postrouting/prerouting chains via insert_before

#

#

The second line tells which sources to never reroute it is very, very important to add your home net

and you dns servers to this list.

#

#example:

preprocessor bait-and-switch: max_entries 200,log,insert_before

preprocessor bait-and-switch-ignorehosts: 10.0.0.0/8

D Shadow Gateway Snort rules

$Id: local.rules,v 1.11 2004/07/23 20:15:44 bmc Exp $

————————

LOCAL RULES

————————

This file intentionally does not come with signatures. Put your local

additions here.

alert tcp 192.168.10.6 80 -> any any (msg:"Successful directory traversal"; content:"root:x:0:0:root:/root"; sid:100050;)

E Snortwatch.pl

#!/usr/bin/perl -w #script to monitor snort logs for specific alerts and react accordingly

use File::Tail;

use Net::SSH2;

use Term::ReadKey;

my $sid;

```
my $src_ip;
my $dst_ip;
$file=File::Tail->new(name=>"./alert",maxinterval=>5);
while (defined($line=$file->read))
{
print "Line: " . $line;
if ($line =~ /^(?s).*\[\*\*\](\s)*\[(\d+):(\d+):(\d+)\]/){
$sid=$3;
for($count=0; $count<4; $count++){
$line=$file->read;
if ($line =~ /((\d+)\.(\d+)\.(\d+)\.(\d+)):(\d+) -> ((\d+)\.(\d+)\.(\d+)\.(\d+)):(\d+)/){
$src_ip = $1;
$dst_ip = $7;
}
}
if ($sid==100050){
print "SID: " . $sid . "\n";
my $vm_name;
my $ssh_con = Net::SSH2->new();
$ssh_con->connect('192.168.10.1') or die;
if ($ssh_con->auth_password('root', 'p@ssw0rd')){
print "SSH Connection Made";
my $chan = $ssh_con->channel();
$chan->exec("iptables -I INPUT -s $dst_ip -j DROP");
$chan=$ssh_con->channel();
$chan->exec("iptables -I FORWARD -s $dst_ip -j DROP");
$vm_name="ShadowWeb";
$cmd_snap = 'perl /root/vmware-viperl-distrib/apps/vm/vmsnapshot.pl –
server 192.168.10.201 –username root –password p@ssw0rd';
$cmd_snap = $cmd_snap . " –vmname $vm_name –sname compromised";
$cmd_revert = 'perl /root/vmware-viperl-distrib/apps/vm/snapshotmanager.pl
–operation goto –server 192.168.10.201 –username root –password p@ssw0rd';
$cmd_revert = $cmd_revert . " –vmname $vm_name –snapshotname fi-
nalstate";
$cmd_power = 'perl /root/vmware-viperl-distrib/apps/vm/vmcontrol.pl –
server 192.168.10.201 –username root –password p@ssw0rd –operation poweron';
$cmd_power = $cmd_power . " –vmname $vm_name";
system($cmd_snap);
system($cmd_revert);
system($cmd_power);
}
}
}
}
```