2007-03

Intrusion deception in defense of computer systems

Goh, Han Chong

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

INTRUSION DECEPTION IN DEFENSE OF COMPUTER SYSTEMS

by

Han Chong Goh

March 2007

Thesis Advisor:	Neil C. Rowe
Second Reader:	Daniel F. Warren

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** March 2007	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE Intrusion Deception in Defense of Computer Systems		**5. FUNDING NUMBERS**	
6. AUTHOR(S) Han Chong Goh			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		**12b. DISTRIBUTION CODE**	

13. ABSTRACT (maximum 200 words)

We investigate deception in response to cyber-intrusion or trespassing on computer systems. We present a Response Framework that categorizes the types of response we can employ against intruders and show how "intrusion deception" has its place in this framework. To experiment, we put together tools and technologies such as Snort, VMware, and honeynets in a testbed open to attacks from the Internet. We wrote some Snort rules and ran Snort in inline mode to deceptively manipulate packets of attackers. Our results showed that attackers did react to our deceptions in some interesting ways, suggesting that intrusion deception is a viable response to intrusion.

14. SUBJECT TERMS Deception, Intrusion Detection Systems, Intrusion Prevention Systems, Active Response Systems, Honeynet, Snort, VMware			**15. NUMBER OF PAGES** 59
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

INTRUSION DECEPTION IN DEFENSE OF COMPUTER SYSTEMS

Han Chong Goh
Civilian, Singapore Defence Science & Technology Agency
B.S., National University of Singapore, 1997

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2007

Author: Han Chong Goh

Approved by: Neil C. Rowe
Thesis Advisor

Daniel F. Warren
Second Reader

Peter J. Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

We investigate deception in response to cyber-intrusion or trespassing on computer systems. We present a Response Framework that categorizes the types of response we can employ against intruders and show how “intrusion deception” has its place in this framework. To experiment, we put together tools and technologies such as Snort, VMware, and honeynets in a testbed open to attacks from the Internet. We wrote some Snort rules and ran Snort in inline mode to deceptively manipulate packets of attackers. Our results showed that attackers did react to our deceptions in some interesting ways, suggesting that intrusion deception is a viable response to intrusion.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I am grateful to my thesis advisor, Professor Neil Rowe, for his constant mentorship and unyielding passion in this field of research. I also thank Chris Eagle and Richard Harkins for extending the use of the Internet leased line and the physical lab space respectively for the purpose of this research. Without your kind support, I would not have completed this thesis.

In addition, I would like to recognize the Master's work done by those who have gone before me, Binh Duong and Harry Lim, who laid the foundations of this thesis. I also acknowledge the staff of Naval Postgraduate School for contributing to the success of this thesis. They range from the ITACS to the Thesis Processing Office and down to our very own Department Education Coordinator, Ms. Jean Brennan.

Lastly, all these would not have been possible if not for the tremendous support that I have from my wife, Aileen, and my toddler daughter, Anne Catherine, who continually inspire and motivate me.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. CYBER-INTRUSION

Malicious hacking has been a perennial problem from the days of wardialing and phone phreaking to the present day of “script kiddies” and underground “blackhats.” In the early days, very little was known about these cyber-intruders, let alone tools to detect and prevent their attacks. Today we depend primarily on our access control and other security policies to prevent unauthorized access. Hardening and patching our systems with the latest service packs and “hot fixes” can further deter intruders. In addition, we also have fairly mature technologies in intrusion detection systems and intrusion prevention systems that react accordingly as a second line of defense should an intrusion occur.

According to the 2006 CSI/FBI Computer Crime and Security Survey [1], the use of intrusion detection systems is at 69% and this has maintained fairly stable over the years. Usage is also measured as fairly stable for other mature technologies such as the use of firewalls, anti-virus software and server-based access control lists. Yet the top two sources of financial losses continue to be attributed to virus attacks and unauthorized access to information. On the other hand, the use of intrusion prevention systems has increased to 43% from 35% in the previous year. We argue that more advances need to take place as the intrusion prevention technology grows and matures so that it can be more effective in responding to intrusions.

B. LIMITATIONS OF CURRENT RESPONSES

The current efforts of intrusion detection and intrusion prevention against cyber-intrusion can be limited. Intrusion detection systems are passive in nature and exist only to alert the administrator of possible intrusions. Further analyses are required to ascertain the validity of the alert and very often we are bogged

down by the sheer number of such alerts. In general, intrusion detection systems do not respond directly to the intruder.

Intrusion prevention systems essentially work the same way as intrusion detection systems except that they have the additional capability to block potentially malicious traffic. Some intrusion prevention systems can dynamically reconfigure the firewall or specific host systems to tighten their security in response to the intrusion [2]. Although these responses are active, they are restricted to entities within the organization and are defensive in nature.

We do not dispute the respective roles of hardening, patching, intrusion detection, and intrusion prevention in our suite of responses to malicious intruders. Hardening and patching our systems are like the helmets and protective gear used in the military for basic protection against the adversary. There is also a need for outposts and blockades to detect and deter potential intruders. This is where intrusion detection and intrusion prevention fit in the framework of intrusion responses.

C. OTHER POSSIBLE RESPONSES

Consider two other possibilities in response to intruders, namely, deception and counterattack. Responding to an intrusion with a counterattack, however, may not be legally justified and it may also cause unintended collateral damage to innocent bystanders [3]. It is similar to the use of disproportionate and unnecessary force to bombard an intruder with artillery shells while causing inadvertent damage to civilians in the vicinity. Due to these ethical and legal issues, we will not consider counterattack responses even though there may be situations that render such courses of actions viable.

Deception serves to confuse, delay or even just to frustrate the adversary. Following the same analogy to conventional military operations, we view deception akin to military employment of ruses and wit to deter or even entrap

intruders. This approach gives us a rich set of alternative techniques for intrusion deception which we can add to our framework of responses.

To develop such deceptive techniques, we need to study and understand how our potential adversaries conduct their business. The Honeynet Project was formed to develop tools and technologies to capture and study intruders who break into a network of decoy machines known as honeypots. It is worthy to note that a honeypot by itself is also a deception. Through the use of this deception, we study the behavior of intruders and develop deceptive techniques to foil their future intrusions.

D. THESIS ORGANIZATION

This thesis is organized into six chapters. Chapter II will survey the previous work done on topics related to intrusion deception. We discuss the possible applications of it and its place in the framework of intrusion responses in Chapter III. In Chapter IV, we present the methodology behind our work with intrusion deception. We analyze the results of our experiments in Chapter V before concluding in Chapter VI.

THIS PAGE INTENTIONALLY LEFT BLANK

II. PREVIOUS WORK

Sun Tzu, the military strategist in medieval China, endorsed the use of deception during battle. His reasoning is twofold: to obtain a swift victory and to achieve it at a minimal loss of lives or casualties from his army. Accordingly, it makes sense to apply deception-related concepts in our battles against cyber-intrusion. We shall discuss a few works by our predecessors in this chapter.

A. DECEPTION TOOLKIT

Cohen published a seminal paper introducing the use of deception in information protection [3]. In the paper, he draws many parallels to the use of deception tactics in a military context. These tactics can be broadly classified as concealment, camouflage, false and planted information, ruses, displays, demonstrations, feints, lies, and insight [4].

At the same time, he made freely available a Deception Toolkit (DTK) which was a software designed to enable intrusion deception. The host server running DTK appears to intruders to have a large number of widely known vulnerabilities. DTK listens for inputs from intruders and provides seemingly normal responses that further confuse or delay them. For example, a fake password file can be provided to the intruder who will have to spend CPU cycles cracking the passwords before finding out that they are useless anyway. In this way, DTK increases the workload of the intruder and serves to frustrate and eventually deter them.

The output responses issued by DTK to the intruder are also programmable, but need an in-depth understanding of the protocol that the intruder chooses to exploit. The last update to DTK was on August 1999.

B. LABREA TARPIT

Liston had the idea of a “sticky honeypot” in response to the CodeRed worm [5]. The resulting software, the LaBrea Tarpit, creates virtual servers that seem to reside on the unused IP (Internet Protocol) addresses of an organization’s allotted IP space. These servers delay intruders by answering connection attempts so that the intruders get "stuck," sometimes for a very long time. Thus the software acts like a tar pit.

This form of deception is very similar to those of DTK except that LaBrea Tarpit specializes only on the TCP connections that an intruder makes while probing a victim network. Since then, other “tarpit” technologies have surfaced. These include SMTP tarpits and IP-level tarpits as implemented by Iptables in Linux kernels.

Developments on LaBrea Tarpit came to a halt in 2003 because of concerns about the DMCA (Digital Millennium Copyright Act) which indirectly made the software an unlawful communication device. This highlights the very real ethical and legal issues that come with using deception even for the purpose of defending our networks.

C. HONEYPOTS AND ANTI-HONEYPOTS

When the Honeynet Project was initially formed it was loosely comprised of security experts who wanted to learn more about how intruders operate and to develop tools and techniques to enable that [6]. Today, the organization comprises of active members from the security community and is still led its founder and security authority, Spitzner. In addition, the Honeynet Research Alliance brings together different independent organizations from around the world that are interested in honeynet research.

A honeynet is a network of honeypot machines used to create an environment where the tools and behaviors of intruders can be captured and studied. Evolving over three generations, the Honeynet Project’s honeynet

architecture consists of three main components: the honeypots that attract the intruders, the gateway that provides data control and data capture, and the remote alert/log server. The specifics of the honeynet methodology, techniques, and tools are well documented in their book.

A honeynet by itself does not provide a response to the intruder. On the other hand, in a more sophisticated intrusion prevention scenario, an intruder can be lured using the actual production server as bait and switched or redirected to a honeypot server acting as a decoy target [7]. Our works departs from this model as we use the honeynet technology solely as a framework in which we can study the intruders and experiment with deceptive responses.

As the honeypot technology matures and becomes more pervasive, there is also increasing interests in detecting the presence of honeypots [8]. Such anti-honeypot techniques focused on examining peculiarities of honeypots such as User-mode Linux, VMware, additional defenses like chroot() and jail(), timing issues, and detecting debuggers. Another interesting development to anti-honeypot technologies is in the area of spamming [9]. As spammers increasingly face the challenge of having their bulk mail channeled to honeypots, they react by creating tools such as the Honeypot Hunter to help them avoid the bait. These developments can be classified as a kind counter-deception and our responses to them as counter counter-deception [10].

D. SNORT

In 1999, Roesch created a “simple, lightweight, and open-source” network intrusion detection system, Snort [11], for small networks. Today, Snort has evolved to become the de facto standard in intrusion detection and intrusion prevention technologies. He was a member of the initial Honeynet Project and Snort was used as one of the data capture mechanisms in its honeynet architecture.

The Snort architecture has three components: packet decoder, detection engine, and the logging and alerting subsystem. In addition, Snort depends on the libpcap library to enable its promiscuous packet sniffing and filtering capabilities. The key to successful detection is rules which define the intrusion signature and the action to take if a rule fires.

If a rule action is configured to “log” or “alert,” Snort essentially operates as an intrusion-detection system. Later versions of Snort allow it to operate in inline mode and the rule actions are extended to “drop” and “reject,” making Snort function as an intrusion prevention system. In addition, there is also a “replace” rule option that modifies packets as they traverse the network. Our interest in Snort lies in this packet-mangling capability of Snort Inline that enables intrusion deception.

E. IMMUNOLOGY

Somayaji and Forrest introduced a kind of automated response that models computer defenses based on the immune system [12]. The proposed pH (process Homeostasis) system monitors all system calls thereby emulating the immune system. It then either delays or aborts system calls if they were deemed to be anomalous, very much like the way antibodies neutralizes the harmful virus.

In order to determine whether an anomaly has occurred, there is a training phase where the pH system learns what constitutes normal behavior. The amount of delay is governed by the locality frame count or the records of the recent anomalous system calls. If the locality frame count passes a certain threshold, then all anomalous system calls are aborted.

This form of immunological responses is essentially contained within the system. It does not interact with the intruder directly and hence can be viewed as a defensive approach. Yet at the same time, it also employs some form of deception to delay the intruder.

F. COMPARISON TO OUR APPROACH

We view our work as continuing these efforts with some differences. We use a patched server without any additions, allowing the attacker to see and interact with the system as is. We also use only one IP address for each honeypot server that we put online leaving the unused IP addresses unoccupied. We do not apply any countermeasures against anti-honeypot technologies although this can be an area of refinement that we can explore in the future.

In addition, we prefer to leverage existing software that has the ability to achieve some level of deception. In particular, we use Snort as an intrusion-detection system and experiment with its intrusion-prevention capabilities. The idea is to modify existing Snort detection rules and change them into deception rules. In this way, we achieve the programmability of DTK without going into the details of the protocol being exploited by the intruder.

Lastly, we want to experiment with deception that interacts with the intruder in a somewhat offensive manner. We do not want to restrict ourselves to defensive deceptions such as those used in the pH system. Our Response Framework in the next chapter categorizes the various responses to intrusion and shows how intrusion deception can be a viable alternative to existing responses.

THIS PAGE INTENTIONALLY LEFT BLANK

III. APPLICATIONS

A. RESPONSE FRAMEWORK

We present a framework for intrusion responses that is currently employed today and also that we propose in this thesis (Figure 1). The matrix in the framework categorizes the responses based on whether they are active or passive, and whether they are defensive or offensive in nature. This classifies the responses such as detect, block, tighten, deceive, and counterattack into four possible categories of responses:

Nature of Response	**Defensive**	**Offensive**
Passive	Detect	Deceive
Active	Block Tighten Deceive	Counterattack

Figure 1. Response Framework

Intrusion-detection systems serve only to detect intrusions and hence are truly passive-defensive in nature. Intrusion-prevention systems essentially use blocking mechanisms to prevent intrusions once they are detected. Some intrusion-prevention systems can even tighten the firewall or host systems depending on the vulnerability the detected intrusion aims to exploit. Yet other intrusion-prevention systems mangle with the malicious network packets to render them harmless once they enter the internal network. Interestingly, this is a form of deception of the internal servers into thinking that all is well. Although

all these dynamic measures are active, they are all restricted within the internal boundary and hence are defensive. Thus, intrusion-preventive responses are mainly active-defensive in nature.

The upper right cell in the matrix represents a class of responses that are passive yet offensive at the same time. This may seem contradictory but we view passive-offensive as aptly describing the very nature of deception. Contrast these with active-defensive ones, such deceptions are not restricted within the internal boundary. They act on outgoing traffic to deceive the intruder, usually in response to malicious requests made by the intruder.

Finally, the last cell can be viewed as a true act of aggression representing responses that are actively on the offensive. These responses are often left as the last resort and are activated only under exigent circumstances. Such scenarios are plentiful in the conventional military context but we must tread carefully when taking the course of counterattack in response to intrusion in cyberspace.

B. RESPONSE LEVELS

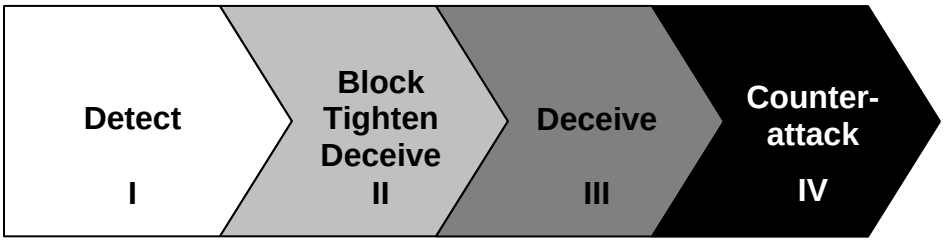

Figure 2. Escalating Response Levels

We can also view our Response Framework as a series of escalating levels (Figure 2). Our Level I response to intrusion is to detect it. When we find out more about the nature of the intrusion, we may then proceed to Level II responses. With the ability to identify previous intrusions, we can block similar future occurrences. We may also tighten our own defenses to remove known

vulnerabilities. In addition, we may employ some form of deception on the incoming intruder so that they do not end up damaging our assets.

If the intrusions still persist despite all our access controls, we may then proceed to Level III deceptions. The intent of these responses is to confuse, delay, and frustrate, but not directly attack the intruder. We do this in the hope of attaining a swift victory at a minimal cost. If this does not deter or divert the intruder, we will then have to consider our Level IV options. Our counterattack must be minimal and proportionate to the force in which the violating intrusion had applied. Prior to the declaration of an all-out war, we must only be seen as using a reasonable and restrained level of force in defense of our assets. As the aggression continues to escalate and hostile intent has been demonstrated, we may then be on the full offensive. Our response could be to launch malicious counterattacks on the system originating the intrusions or even its neighbors depending on the threat level.

The Response Levels can indicate different levels of an organization's ROE (Rules of Engagement) in response to intrusion threats. Under normal non-hostile conditions, an organization's ROE level would be Level I. With the occurrences of confirmed intrusions, the ROE level can then be upgraded accordingly.

Intrusion-detection and intrusion-prevention vendor products can also be similarly rated. For example, intrusion-detection systems are typically Level I products while current intrusion-prevention systems are Level II products. An organization's ability to react to the various ROE levels depends largely on the availability of suitable enabling tools. Intrusion deception plays a crucial role in pushing the current boundaries of intrusion-prevention technology and taking intrusion responses to the next levels.

C. ASSUMPTIONS

Besides technical issues, an important barrier to the advancement of intrusion-prevention technologies to Levels III or IV is the legal and ethical issues with deception and counterattack. In addition, the Internet space cut across international boundaries that span a variety of laws and constitutions that further complicate the employment of these measures. If some common code of conduct such as the Geneva Conventions can be adopted, then we can achieve a certain level of understanding when employing Level III and IV technologies. However, this does not resolve all of the legal and ethical issues any more than the Geneva Conventions did to the conduct of war.

Leaving legality and ethics aside, Level III deceptions can themselves create problems on real production servers. This is because a production server is supposed to be truthful and provide services in accordance to the standards and protocols that govern its services. Fortunately, there are technologies such as the "bait and switch" honeypot that redirects the intruder to a decoy server which can handle the deceptive responses.

The earlier discussion on the escalating Response Levels does not restrict us from employing different technologies in parallel in a defense-in-depth manner. We still need helmets and protective gear for basic protection, and also outposts and blockades as a second line of defense. In addition, we now have a new set of defenses to respond to intruders at Levels III and IV.

IV. METHODOLOGY

We break down our work on intrusion deception into three distinct phases, namely, setup, rule development, and experimentation. The setup phase involved studying the intruders and their exploit methods. We targeted the common intrusion exploits that we collected in the setup phase and developed corresponding deceptive rules to test in the experimentation phase. This process is cyclical in nature as the results of the experimentation provide feedback to fine tune the rules.

A. SETUP

We set up our testbed (Figure 3) in the first phase. Most of the setup was done in the thesis of Binh Duong [13]. The three components of the setup correspond to the honeynet architecture discussed in Chapter 2. The testbed makes use of virtualization software to run the different honeypots. We take a hybrid approach in which the virtual honeypots are hosted in one physical machine while the gateway and remote alert/log servers are hosted in two other machines.

The honeynet machine runs VMware that hosts two virtual machines a Windows 2000 Server and a Windows XP machine. The router machine serves as the gateway and channels traffic from the Internet to the honeynet and vice versa. Data capture is also done on the router machine by running Snort in intrusion-detection mode. The alerts are logged locally and also sent to a remote Postgres database that is inaccessible from the Internet. As Snort listens to the network in promiscuous mode, we encountered alerts that were meant for servers on the network other than our setup. One way to eliminate these alerts is to configure Snort to restrict its processing only to the IP address of the router machine instead of a subnet of addresses. However, we chose to keep these secondary alerts for better situational awareness and filter them out through structured queries on the database when necessary.

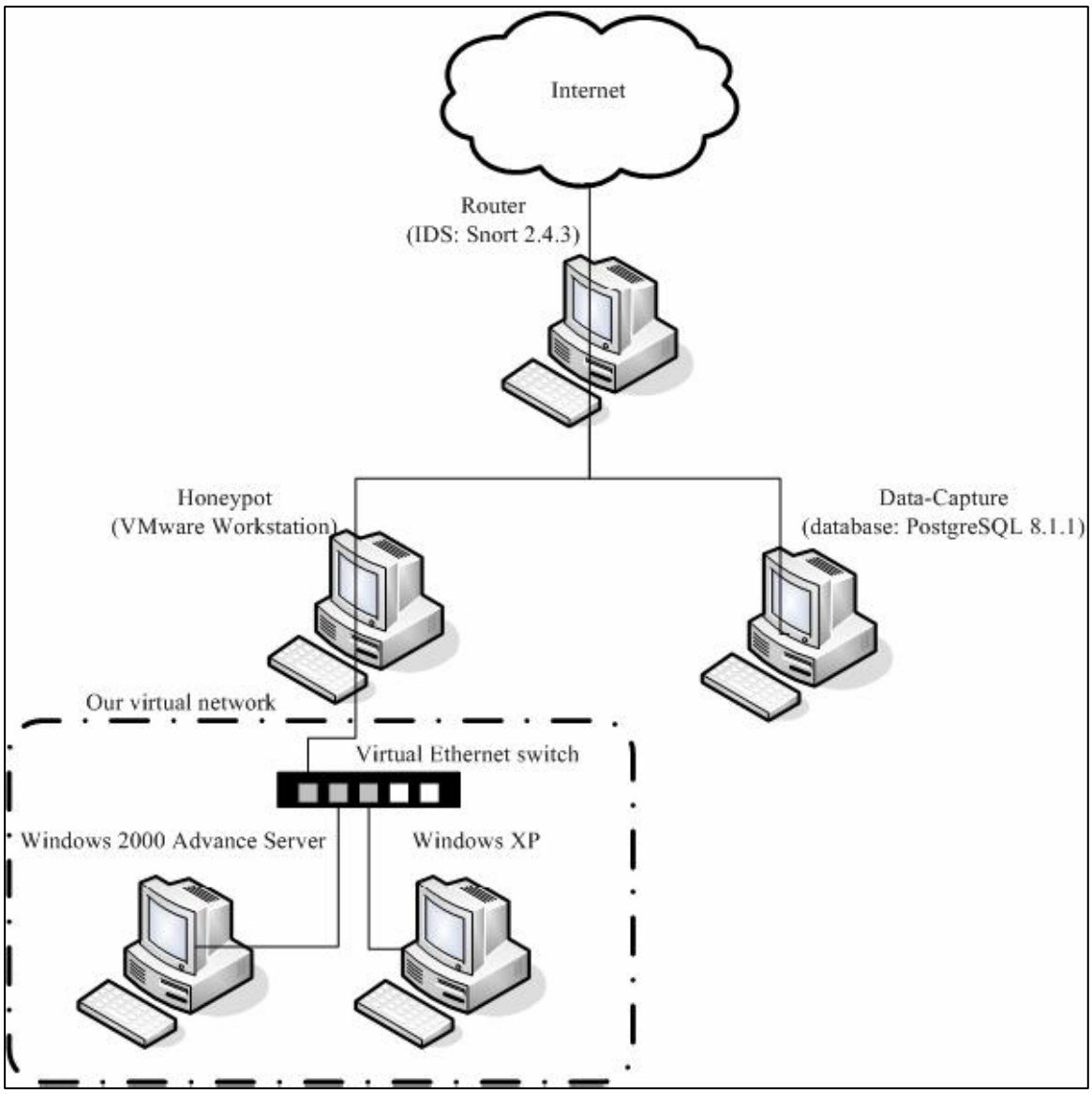

Figure 3. Learning Testbed

We ran this testbed for about a year and observed the number and types of Snort alerts that took place. The default Snort configuration had did not differentiate between the external network and the home network. This resulted in alerts that were false positives such as the MS-SQL outbound alerts that were triggered even though the intrusion was clearly inbound. We rectified the configuration by setting the home and external network accordingly. From these

numerous observations, we created a list of exploits used frequently by intruders. This list forms a base set of exploits for experiments with intrusion deception.

We took precautions such that should our honeypots become compromised, they will not be used by the intruder as a launch pad to spread further attacks. This is achieved by regularly monitoring the states and processes on the honeypots. We also blocked traffic generated by our honeypots to external IP addresses that seem to be probing other networks. These were systematic sets of network messages destined for external addresses that varied only in the last octet and occurred within a second. In the event that our honeypots are compromised, it is easy to restore to a clean state with the use of VMware but we only did this a few times in the year we ran the honeynet.

B. RULE DEVELOPMENT

As noted earlier, we want to experiment with the deception capabilities of Snort running in “inline” mode. This runs Snort as a simple intrusion-prevention system instead of the default intrusion-detection mode. Inline mode requires two additional packages, Iptables and Libnet. Iptables is used to intercept network packets from the network interface card and then pass them along from the kernel space to the user space for Snort to process. Packets not intercepted by Iptables are of no interest to Snort Inline and they proceed on to their intended destination. Libnet is a set of high level APIs that allows Snort to construct and inject network packets. We still need to write Snort rules to enable intrusion deception. The design of the Snort rules must be done in tandem with the Iptables configuration. It would be easy to configure Iptables such that it would intercept all packets and let Snort process them, but this would be very slow. So we should configure Iptables to be an effective prefilter for Snort. On the other hand, Snort rules should only affect network packets captured by the Iptables configuration, so we need to verify the Iptables configuration whenever make changes to the rules.

We will present a few examples to illustrate the subtle relationship between Iptables and Snort. Say we want to intercept all inbound ICMP traffic through Iptables so that we can direct Snort Inline to drop these packets. We would need a rule in the default Filter table in Iptables as:

```
iptables -I INPUT -p icmp -j QUEUE
```

This inserts the rule at the top of the rule set in the INPUT rule chain, giving it the highest priority on that chain. The rule intercepts incoming network packets of the ICMP protocol and passes them on or jumps to a Snort accessible object called the QUEUE. The corresponding Snort rule would be:

```
drop icmp $EXTERNAL_NET any -> $HOME_NET any (…)
```

The drop rule action will not pass any incoming ICMP packets from external origins on to the operating system thus blocking inbound ICMP traffic.

To set up the configuration to block outgoing ICMP packets, the rules for Iptables and Snort would be:

```
iptables -I OUTPUT -p icmp -j QUEUE

drop icmp $HOME_NET any -> $EXTERNAL_NET any (…)
```

Note that we now apply the rule on the OUTPUT rule chain in Iptables and the directionality is reversed in the Snort rule.

As packets traverse the different chains under the filter table in Iptables, they take different packet flow paths (Figure 4). There are two other tables, Nat and Mangle, with their respective chains under Iptables. The Nat table is meant for network address translation while the Mange table does low-level packet mangling such as that of the TOS (Type of Service) field. For our purposes, we only needed to configure the Filter table which is meant for traffic filtering.

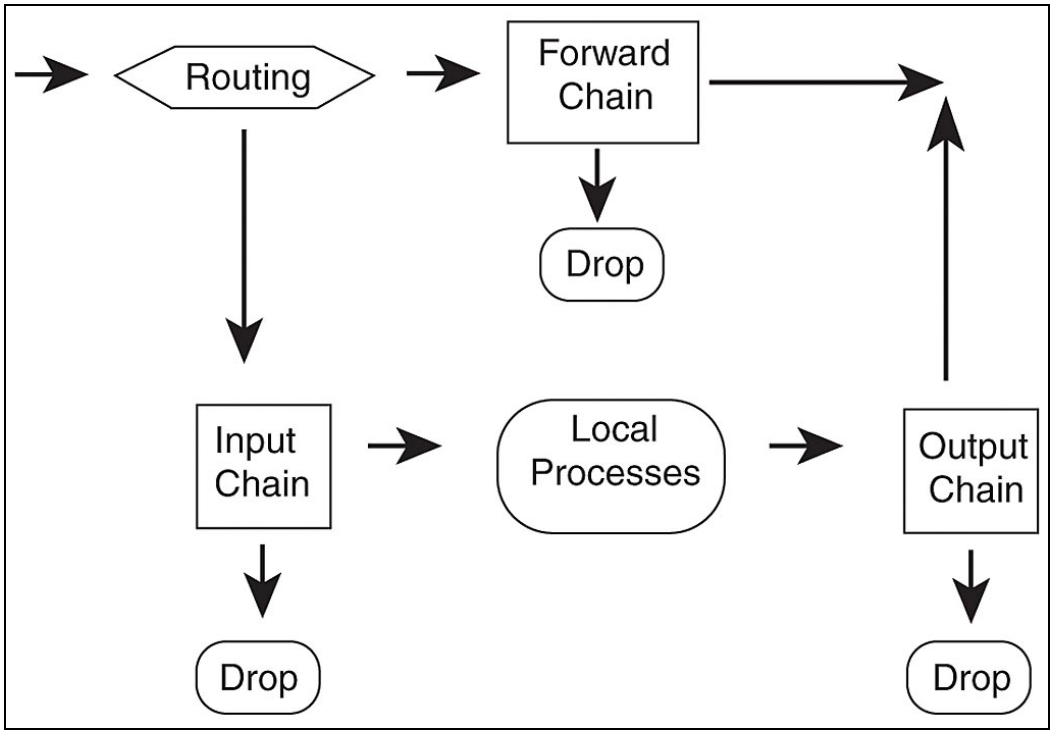

Figure 4. Packet flows and chains under Iptables

The “Local Processes” node represents the processes that run on our router machine. The router machine is assigned the public IP address that is accessible on the Internet. Therefore, when an incoming ICMP packet reaches the network interface card, Iptables checks if the packet requires routing. In this case the router machine itself will answer to the ICMP request and hence it does not require further routing. The packet is passed on to the INPUT chain and Iptables will run the packet through the filtering rules in that chain. The filtering rules then decide to pass or drop the packets. The flow is the same if the router machine is sending outgoing ICMP packets except that now they pass through the OUTPUT chain.

Due to the peculiarity of our testbed, traffic destined for our honeynet will be routed accordingly by the router machine. In this case, the final destination is not the router machine and hence the traffic will not pass through the INPUT or

OUTPUT chains but through the FORWARD chain instead. To illustrate, if we want to drop outbound probes for port 135 on external IP addresses, the respective rules for Iptables and Snort would be:

```
iptables -I FORWARD -p tcp –-dport 135 –s X.X.X.X/24 -j QUEUE

drop tcp $HOME_NET any -> $EXTERNAL_NET 135 (…)
```

The Iptables rule captures routed traffic TCP that comes from our honeynet and is destined for port 135 of any IP address. The Snort rule acts on the same traffic and drops it accordingly.

Our Snort rules so far have only used the action “drop” making Snort just an intrusion prevention system. We show an example of using the “replace” option in the rule to enable intrusion-deception.

```
alert tcp $HOME_NET 21 -> $EXTERNAL_NET any (content:"530 ";
replace:"331 "; isdataat:1, relative; content:"cannot log in";
replace:"logged in.   ";)
```

This Snort rule acts on FTP (File-Transfer Protocol) responses from our honeynet to the any FTP client on the Internet. The “alert” rule action seems to be running Snort in intrusion-detection mode. However, the “replace” rule option is used (twice in this case) to make Snort run as a Level III system. The first “replace” substitutes the FTP return code “530” with “331”. The FTP client will be deceived into interpreting the response “331” as “User name okay, need password” instead of “530” which is “User X cannot log in”. The second “replace” substitutes the message associated with the return code “530” with that associated with the code “230 – User X logged in”. Note that there are trailing spaces in the second “replace” because the implementation in Snort requires an exact number of characters to be substituted. The option “isdataat” takes care of the “User X” string that appears after the return code but before the string “cannot log in”. In essence, there are two deceptions that take place here. One is with the FTP return code, while the other is with the return message. It is not necessary to synchronize both deceptions and we do this in the hope of confusing the intruder.

Although Snort in intrusion-detection mode was already in place during the setup phase, we ran another instance of Snort on the router machine in intrusion-deception mode to accomplish the packet-manipulation rules. We also set up a corresponding database to receive alerts from the second instance of Snort. In addition, it has become apparent over time that we need a different set of clues other than the Snort alerts as a cross reference in aid of a more complete forensic analysis [14]. As a result, we also ran a full packet capture using Tcpdump on all the network traffic arriving at the router machine. The details of configuring Iptables and the various command line options to run Snort Inline and Tcpdump can be found in Appendix A.

C. EXPERIMENTATION

Having set up Snort Inline, we developed several types of deceptive rules we could use. In particular, we wanted to compare across the Response Levels. This led us to group experiments in terms of Level I, Level II, and, Level III responses. In addition, we also experimented with the effects of operating system patching which is a Level 0 response.

As we had a single honeynet, we successively reused the same setup for each experiment but made modifications to reflect the correct Response Level. The nature of intrusion attempts generated by each of these Response Levels provides us with a comparison across levels. Specifically, we looked at the counts and variety of intrusion types over the period of time when a particular Response Level was active. The counts are the number of intrusions while the variety is the number of distinct intrusion types. For a more detailed analysis, we break down the intrusion counts into five major intrusion groups and study how the counts in these groups are affected by the different Response Levels.

For the Level 0 experiment, we set up a honeypot with Windows 2000 Advanced Server without any service packs or patches. The honeypot was then put online allowing intruders to interact with it. We collected a separate 36 hours of intrusion data for each particular experiment. We ran each experiment several

times to collect the mean and standard deviation of its statistics. As a control experiment, we also perform the same experiments but updated the operating system with the latest service pack and patches.

The Level 0 experiment examined the effects of introducing a newly installed system on the Internet. In the Level I experiment, we let the setup stabilize over a period of one week after it was first put online. This setup was no longer novel to intruders and we used this to represent the typical system that runs a Level I intrusion-detection system.

For the Level II experiment, we modified the Snort rules to drop ICMP traffic going to and coming out of our honeynet. The purpose was to study the effects of blocking reconnaissance activity on the subsequent actions of the intruder. This setup was a kind of intrusion-prevention system.

Finally, we focused on intrusion deception for our Level III experiment. We kept the rules from the Level II experiment but in addition, we changed the rules to replace certain keywords in protocol messages with deceptive ones. Such keywords can include return codes and substrings in return messages as discussed in our file-transfer example earlier. Our strategy here was to delay the intruder by faking protocol messages and codes. Another instance of our deception was to change the protocol version or command number to an invalid one. Here, we were hoping to confuse and discourage the intruder through the breakdown in the protocol. The details of these deceptive rules are listed in Appendix B. As a subexperiment, we also examined the effects of deceptive responses on file-transfer intrusions.

V. RESULTS ANALYSIS

Our initial findings from the alerts collected for the past year indicated that the intruder behavior is affected significantly by our honeynet downtime [15]. We clustered the data in three different ways and found that they all pointed to the same conclusion; that intrusions spiked whenever our honeynet recovered from a period of downtime (Figure 5, taken from [15]). The spikes occurred after weeks twelve and twenty-six when we had just brought the honeynet online after a period of maintenance. This led us to further investigate this finding through the series of experiments that follow.

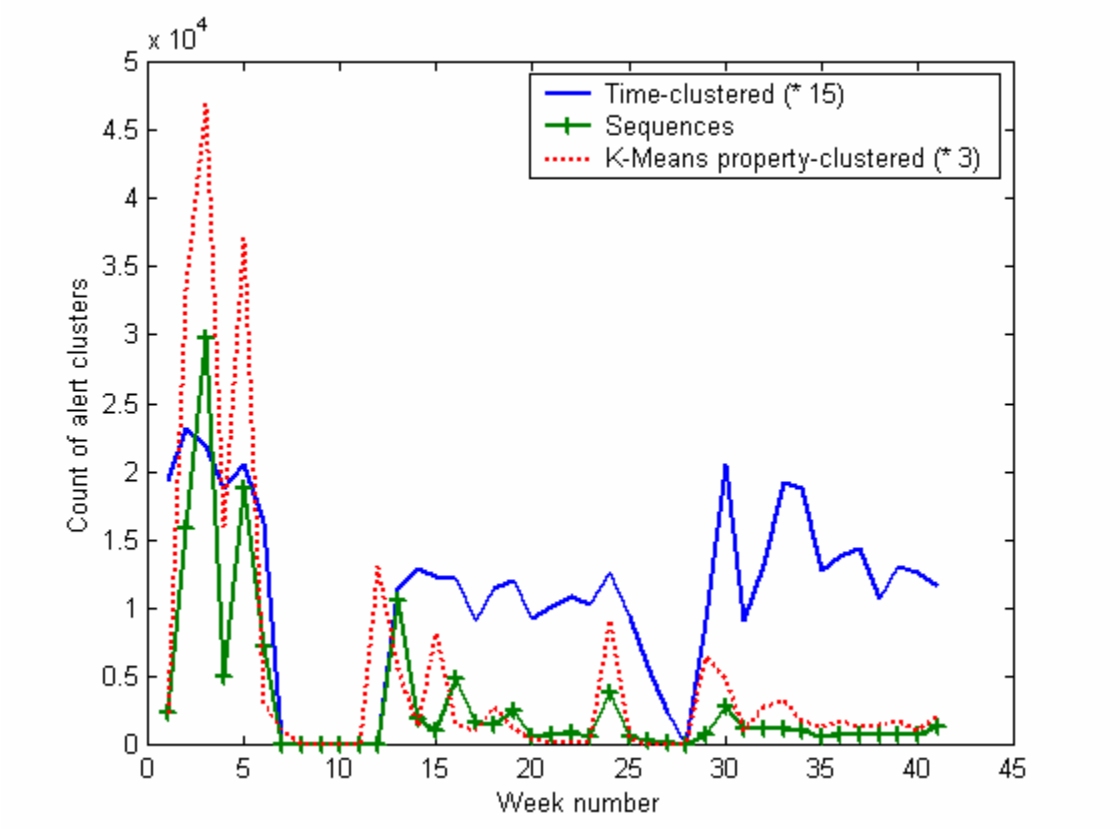

Figure 5. Alerts clustered by time, sequences, and K-Means properties

In tabulating the results we obtain from the various experiments, it was helpful to remove some intrusion types that gave anomalous counts. This was also done with the ICMP Unreachable backscatter [16] because their large numbers would skew the data and prevent proper comparison. We list such intrusion types (Table 1) and explain why they are excluded from the normalized data. These intrusion types accounted for 98.7% of the raw counts for one run in the Level 0 experiment where the honeypot was compromised and used by the successful intruder to probe other networks. Counts were greatly inflated thus indicating an anomaly.

S/No	Intrusion Type	Reason for exclusion
1.	ICMP Destination Unreachable Host Unreachable	These intrusion types reflect our honeypot being used as a bot to probe other networks and do not bear on the actual successful intrusion.
2.	ICMP Destination Unreachable Port Unreachable	
3.	ICMP Destination Unreachable Network Unreachable	
4.	ICMP redirect host	These intrusion types were side effects of the intrusion types above.
5.	ICMP redirect net	
6.	INFO FTP Bad login	These intrusion types were sporadic and the counts varied widely depending on whether the intruder was using an automated tool and what kind if so.
7.	NETBIOS SMB repeated logon failure	

Table 1. List of excluded intrusion types

A. LEVEL 0: PATCHED VS. UNPATCHED

Run	Count	Variety
1	1219	20
2	937	18
3	1043	12
Mean	**1066.33**	**16.**
S.D	**142.44**	**4.16**

Table 2. Normalized results of an unpatched system

Run	Count	Variety
1	609	9
2	244	10
3	296	22
4	748	18
5	1150	19
Mean	**609.40**	**15.60**
S.D	**368.51**	**5.77**

Table 3. Normalized results of a patched system

The Level 0 experiment compared the effects of an outdated operating system (Table 2) with an updated one (Table 3) after the system was brought online. Comparing the data, we can see that the unpatched systems had higher counts and variety (as defined in Chapter 4) with lower standard deviations. This suggests that outdated operating systems are more likely to attract intruders than updated ones. In addition, since there are more known vulnerabilities in outdated operating systems, intruders have more leeway in choosing the type of attacks

against it. These findings confirm the effectiveness of operating system patching, a basic Level 0 response, against intrusion.

Run	ICMP Ping	MS-SQL Overflow	NETBIOS Overflow	SHELLCODE NOOP	Others
1	997	100	13	32	77
2	786	64	22	59	6
3	941	70	12	29	2
Mean	**908.00**	**78.00**	**15.67**	**40.00**	**28.33**

Table 4. Results of an unpatched system by intrusion groups

Run	ICMP Ping	MS-SQL Overflow	NETBIOS Overflow	SHELLCODE NOOP	Others
1	365	58	0	0	186
2	128	40	0	0	76
3	53	54	71	80	38
4	246	56	198	243	5
5	588	62	262	235	3
Mean	**276.00**	**54.00**	**106.20**	**111.60**	**61.60**

Table 5. Results of a patched system by intrusion groups

Next, we compared the intrusion counts for each of the five major intrusion groups between the unpatched (Table 4) and the patched (Table 5) systems. The ICMP Ping and MS-SQL overflow intrusion groups had a higher count for the unpatched system. The reverse was true for the NETBIOS Overflow, SHELLCODE NOOP, and Others intrusion groups. This suggests that the

intruder typically conduct more ICMP type of reconnaissance on an unpatched system. In addition, the preferred intrusion type for an unpatched system seems to be from the MS-SQL group. On a patched system, however, the intruder opted for more sophisticated and varied types of intrusion as seen in the significant increase in the NETBIOS Overflow, SHELLCODE NOOP and Others groups.

B. LEVEL I: STEADY-STATE SYSTEM

Run	Count	Variety
1	767	9
2	679	25
3	439	16
4	601	16
5	796	16
Mean	**656.40**	**16.40**
S.D	**143.67**	**5.68**

Table 6. Normalized results of a steady-state system

In the Level I experiment, we studied how intrusions on a system running an intrusion-detection system has stabilized since it was first brought online (Table 6). Comparing the data with a Level 0 patched system, we see that the intrusions for the stabilized system increased slightly but had a significantly lower standard deviation. The same applies to the variety of intrusion types. This suggests that even if the system is patched, it is still subjected to attacks. Moreover, the fact that it is still uncompromised over a week seems to encourage more and varied intrusion attempts.

Run	ICMP Ping	MS-SQL Overflow	NETBIOS Overflow	SHELLCODE NOOP	Others
1	521	58	0	0	188
2	148	58	172	151	150
3	144	64	129	88	14
4	277	58	137	128	1
5	640	52	37	54	13
Mean	**346.00**	**58.00**	**95.00**	**84.20**	**73.20**

Table 7. Results of a steady-state system by intrusion groups

We made a similar comparison between the patched and the steady-state systems but this time on the intrusion groups (Table 7). There was a slight increase in the ICMP Ping, MS-SQL Overflow, and Others intrusion groups. This concurred with our earlier conclusion that a steady-state system is still subjected to attacks. The focus now seems to be on ICMP reconnaissance and more varied attacks under the Others group.

C. LEVEL II: BLOCKING SYSTEM

We observe a system running an intrusion-prevention system in our Level II experiment (Table 8). Comparing it with the Level I system, we see that the intrusion count increased significantly with a low standard deviation, but the variety of intrusion types decreased with a lower standard deviation. This suggests that blocking ICMP reconnaissance attempts from intruders only make them more curious about compromising the system. However, the blocking strategy seems to be effective in restricting the types of intrusions that occur.

Run	Count	Variety
1	892	14
2	1016	10
3	1027	18
4	881	20
5	968	14
6	1244	17
7	933	17
8	917	11
Mean	**984.75**	**15.13**
S.D	**117.65**	**3.48**

Table 8. Normalized results of a blocking system

Run	ICMP Ping	MS-SQL Overflow	NETBIOS Overflow	SHELLCODE NOOP	Others
1	579	74	95	140	4
2	881	84	17	33	1
3	807	78	58	72	12
4	700	78	25	46	32
5	675	60	70	147	16
6	932	114	64	120	14
7	762	70	52	45	4
8	785	86	20	25	1
Mean	**765.13**	**80.50**	**50.13**	**78.50**	**10.50**

Table 9. Results of a blocking system by intrusion groups

The count by intrusion groups also supported our conclusion about the Level II system (Table 9). The ICMP Ping group more than doubled while the MS-SQL Overflow group had a modest increase. The rest of the groups reported decreases with the most significant drop in the Others group. The blocking system was successful in reducing these intrusion groups but at the expense of rousing the curiosity of intruders and causing more reconnaissance.

D. LEVEL III: DECEPTIVE SYSTEM

We examined a system running intrusion deception in our Level III experiment (Table 10). Comparing it with the Level II system, we see that the intrusion count has increased with a fairly low standard deviation. On the other hand, the variety of intrusion types has decreased significantly with a low standard deviation. This suggests that our deceptions seem to encourage more intrusions possibly due to the confusion and delay that was caused by the deceptive responses. However, our deceptions were more effective in restricting the intrusion types than Level II blocking did.

Run	Count	Variety
1	987	15
2	971	14
3	776	13
4	1234	11
5	1526	15
6	1519	18
7	1356	14
Mean	**1195.57**	**14.29**
S.D	**291.83**	**2.14**

Table 10. Normalized results of a deceptive system

The comparison by intrusion groups agreed with our discussion about the Level III system (Table 11). Our deception targeted the NETBIOS Overflow thus explaining its significant decrease. This also led to a consequential decrease in the SHELLCODE NOOP group because these intrusions usually follow after successful NETBIOS Overflow attempts. The intruder was thus forced to try other means as shown by the increase in the ICMP reconnaissance, MS-SQL Overflow and Others groups.

Run	ICMP Ping	MS-SQL Overflow	NETBIOS Overflow	SHELLCODE NOOP	Others
1	854	90	23	15	5
2	875	64	9	19	4
3	693	64	5	12	2
4	1081	132	6	8	7
5	1304	166	9	9	38
6	1123	154	55	124	63
7	1098	150	19	77	12
Mean	**1004.00**	**117.14**	**18.00**	**37.71**	**18.71**

Table 11. Results of a deceptive system by intrusion groups

We also examined the effects of our FTP deception on the duration of logon attacks attempts. We charted the durations of FTP intrusions by eight intruder IP addresses (Figure 5). This first shown with the dotted pattern fill indicates a typical FTP intrusion on a Level II system; the remaining seven occurred on a Level III system. Six out of these seven intrusions ended within two hours; one where our deception was not successful in dissuading the intrusion ended just less than five hours. The latter attack suggests an intruder who uses automated tools to scan and conduct attacks. They probably let the

tool run without much monitoring and return some five hours later to check on their yield. In all seven attempts on the Level III system, the intruder did not learn anything about the password from the numerous logon attempts. The first FTP intrusion on the Level II system, however, the intruder had learnt that the three hours worth of password combinations tried so far do not work.

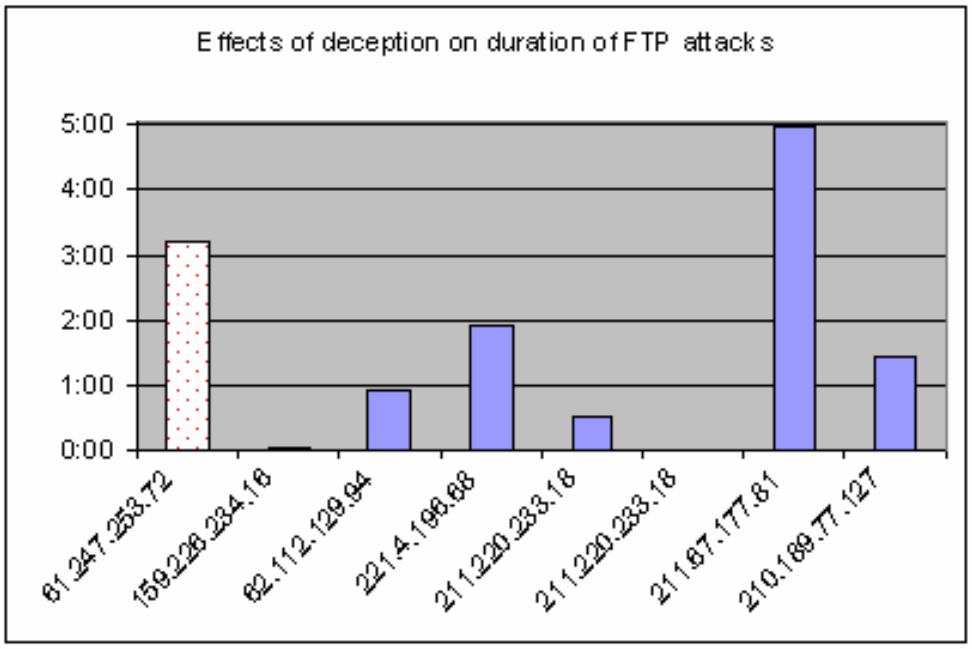

Figure 6. Effects of deception on duration of FTP attacks

VI. CONCLUSIONS

A. ACHIEVEMENTS

We have shown that intruders are affected by the different responses to their intrusions. We summarize our previous results in Figure 6 by plotting the intrusion counts (left y-axis) and variety (right y-axis) across the different Response Levels. The Level I system performs best at keeping the intrusion counts at a stable low. As we go to Levels II and III, the intrusion count rises although the Level III rise is not as much due to a higher standard deviation. At the same time, the intrusion variety decreases at a slower rate. This suggests the potential of Level II and III systems in dealing with intrusion types that Level I systems cannot handle.

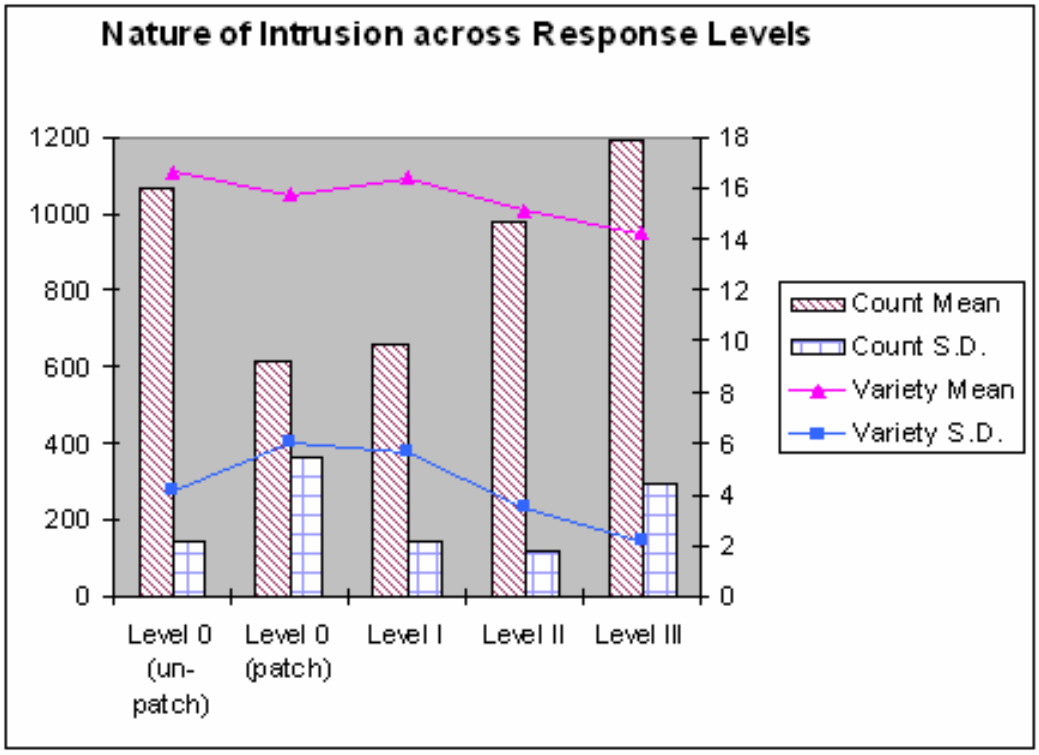

Figure 7. Statistics on Intrusions across Response Levels

We also showed that the intruders' choice of attack was affected by the different Response Levels in use (Figure 8). We were successful in decreasing the intrusions belonging to the NETBIOS Overflow, SHELLCODE NOOP, and Others group as we moved from a patched Level 0 system to a Level III system. The side effect of progressing up the levels is the increase in the intrusions belonging to the ICMP PING and MS-SQL Overflow groups. We also note an increase in the Others group with our Level III deceptive system as intruders are forced to find other means of attack.

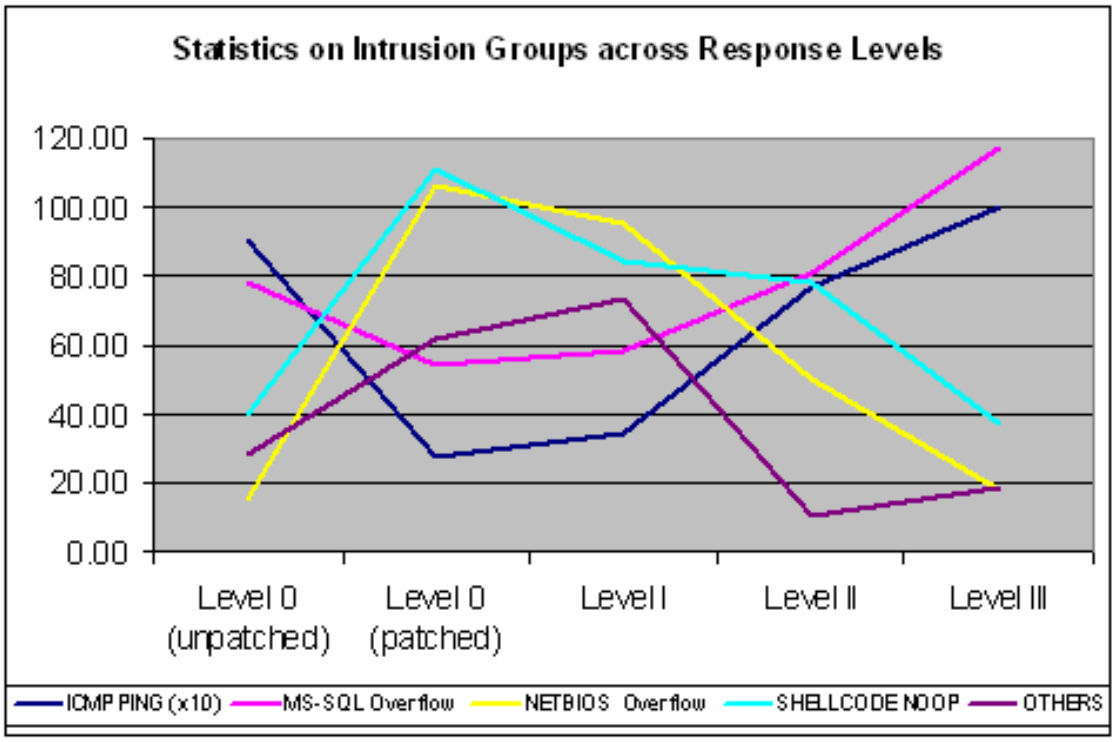

Figure 8. Statistics on Intrusions Groups across Response Levels

In summary, Level 0 experiments showed that patching our systems is effective as a basic form of protection against intruders. On top of that, a Level I response using intrusion-detection systems provides an effective layer of secondary defense against intrusions. To respond to common intrusions that

take place even with Level I defenses in place, we can take advantage of Level II and III responses.

As a subexperiment in Level III responses to FTP intrusions, we have also shown that deception can attain a swift victory at a minimal cost against many intruders. However, similar successes were not observed for deceptions for other intrusion types as indicated by the high intrusion counts in our Level III experiments.

B. WEAKNESSES

An important weakness of our Level III experiment was the limitations of the "replace" capability in Snort: we could only substitute words with other words of the same length. We also did not have time to try the pattern matching capability with regular expressions that is in Snort. In addition, the "replace" capability of Snort Inline is restricted to the application payload of the network packet. This prevents us from changing header information such as the IP address or the port number for some additional useful deceptions. This limits the level of sophistication of our deceptions, and might explain the increase in intrusion counts for our Level III experiment.

A weakness in our Level II experiment is that we only blocked ICMP messages. The experiment would be more representative of a Level II system if we blocked other intrusions as well. Lastly, our experiments were conducted using the same IP address over a year. Over this time, seasoned intruders would have learned and passed along information about the system on this IP address. This may have affected the nature of intrusions that occurred on our system even though it evolved from Level 0 to Level III over that time. It would be a fairer comparison if we were to run parallel setups for Level 0 through Level III on different IP addresses over the same time period.

C. FUTURE WORK

Possible future work on intrusion deception could use more sophisticated deceptions against the intruder. This could involve other tools with deception capabilities instead of the “replace” feature in Snort. One possibility is to use the packet mangling capabilities in Iptables which would allow us to modify packet header information along with the payload. However, this would require designing our deceptions at a fairly detailed level and an in-depth understanding of the exploited protocol.

Other aspects of deception can also be explored. These include the tarpit capability which is also inherent in Iptables. When experimenting with tarpit deception, it is worthy to note that the objective is no longer to achieve a swift victory but rather to delay the intruder. As such, a different set of metrics need to be derived in order to compare its effectiveness across the Response Levels.

Finally, our deceptions can be detected and the intruder may take measures to counter our deception. As a consequence, we also need to explore what we can do to counter the intruder’s counter-deception. One possibility of such counter counter-deception is the use of fake honeypots which supports the notion of achieving victory at a minimal cost.

APPENDIX A: IPTABLES AND SNORT CONFIGURATION

This set of configuration commands assumes that the current system is preinstalled with the required modules to run Iptables and Snort Inline. By running this set of commands, we configured the system into a Level II or III system.

1. By default, all traffic flowing to the kernel and back to user space must be intercepted by Iptables and passed to Snort Inline for processing. Iptables accomplishes this by pushing the data into a queue using the ip_queue module. You can load ip_queue and verify its presence as follows

```
modprobe ip_queue
lsmod | grep ip_queue
```

2. For a Level II system, set Iptables to intercept all inbound and outbound ICMP traffic and place them in the QUEUE object.

```
iptables -I INPUT -p icmp -j QUEUE
iptables -I OUTPUT -p icmp -j QUEUE
```

   For a Level III system, set Iptables to intercept all forwarding traffic (both inbound and outbound) and place them in the QUEUE object.

```
iptables -I FORWARD -j QUEUE
```

   To delete the topmost rule,

```
iptables -D INPUT 1
```

   Note that upon reboot, all settings above are lost!

3. List Iptables to confirm insertion of the rule above

```
iptables -L INPUT
```

4. The corresponding configuration to enable a Level II or III system is to configure the Snort rules in Appendix B. Note that every time you change the rules, you must restart Snort Inline for changes to take effect.

5. Once the configuration above has been done, start Snort Inline as follows.

```
snort_inline -Q -i eth0 -v -c /etc/snort_inline/snort_inline.conf
-l /var/log/snort_inline/ -D
```

-Q: receive packets from Iptables-QUEUE,
-i: network interface to sniff,
-v: verbose mode,
-c: configuration file,
-l: log file,
-D: daemon mode

6. For additional network data capture, we can run Tcpdump as follows.

```
tcpdump -i 1 -n -N -s 0 -v -U -C 15 -w /home/tcpdump/0201-1610- &
```

-i 1: listen to interface 1; use "tcpdump -D" to determine what is interface 1.
-n: do not do name resolution
-N: do not use fqdn
-s 0: capture full packet
-v: verbose mode
-U: write to file directly instead of the buffer
-C 15: limit each output file to (approx) 15MB for quick loading in WireShark
-w /home/tcpdump/0201-1610-: output file path + name
&: run in background mode

Type "exit" the next line to close window without killing the Tcpdump process

APPENDIX B: SNORT RULES

For the Level I experiment, we used the rules that come with the default Snort installation. These rules are used in combination with the following ones for our Level II and III experiments.

The rules below enable a Level II system by blocking ICMP traffic.

```
drop icmp $EXTERNAL_NET any -> $HOME_NET any
(classtype:attempted-recon; msg:"Drop ICMP Request"; itype:8;)

drop icmp $HOME_NET any -> $EXTERNAL_NET any(classtype:attempted
recon; msg:"Drop ICMP outbound";)
```

The four rules below enable intrusion deception on a Level III system. The one immediately below replaces the FTP return code and words in the return message as discussed in Chapter 4.

```
alert tcp $HOME_NET 21 -> $EXTERNAL_NET any (classtype:bad-
unknown; msg:"Replace FTP Bad login";
flow:from_server,established; content:"530 "; replace:"331 ";
isdataat:1, relative; content:"cannot log in."; replace:"logged
in.    "; pcre:"/^530\s+(Login|User)/smi"; sid:491; rev:8;)
```

The rule below replaces the SMB error code of “C0” to no error which is “00”.

```
alert tcp $HOME_NET 445 -> $EXTERNAL_NET any (msg:"Replace
NETBIOS SMB-DS repeated logon failure";
flow:from_server,established; content:"|FF|SMB"; depth:4;
offset:4; content:"s"; within:1; content:"m|00 00 C0|";
replace:"m|00 00 00|"; within:4; classtype:unsuccessful-user;
sid:2924; rev:3;)
```

The rule below replaces the DCERPC protocol version number of “5.0.0” to an invalid one represented by the hexadecimal “88 88 88”.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 135 (msg:"Replace
NETBIOS DCERPC inbound"; flow:established,to_server; content:"|05
00 00|"; replace: "|88 88 88|"; classtype:protocol-command-
decode; sid:9601; rev:2;)
```

The rule below replaces the DCERPC command represented by the hexadecimal “0B 00 01 1C” to an invalid one.

```
alert tcp $HOME_NET 135 -> $EXTERNAL_NET any (msg:"Replace
NETBIOS DCERPC outbound"; content:"|0B 00 01 1C|"; replace:"|88
88 88 88|"; classtype:protocol-command-decode; sid:9601;)
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Gordon, L. A. and others. (2006). 2006 CSI/FBI Computer Crime and Security Survey. Annual Survey, Computer Security Institute, San Francisco, California.

2. Rash, M., and others. (2005). Intrusion Prevention and Active Response. Rockland, Massachusetts: Syngress.

3. Himma K. E. (2004). The Ethics of Tracing Hacker Attacks through the Machines of Innocent Persons. International Review of Information Ethics, Vol II, Stuttgart, Germany.

3. Cohen, F. (1998). A Note on the Role of Deception in Information Protection. White Paper, http://all.net/journal/deception/deception.html, accessed Jan 2007.

4. Dunnigan, J. F., and Nofi, A. A. (2001). Victory and Deceit: Deception and Trickery in War, Second Edition. San Jose, California: Writers Club Press.

5. Haig, L. (2003). LaBrea – A New Approach to Securing our Networks. White Paper, SANS Institute, Bethesda, Maryland.

6. The Honeynet Project. (2004). Know Your Enemy: Learning about Security Threats, Second Edition. Boston, Massachusetts: Addison-Wesley.

7. Northcutt, S., and Novak, J. (2002). Network Intrusion Detection, Third Edition. Indianapolis, Indiana: New Riders.

8. Holtz, T., and Raynal, F. (2005). Detecting Honeypots and other suspicious environments. Proc of the 2005 IEEE Workshop on Information Assurance and Security, United States Military Academy, West Point, New York.

9. Krawetz, N. (2004). Anti-Honeypot Technology. The Honeynet Files, Vol. 2 Issue 1, IEEE Security & Privacy, Washington, District of Columbia.

10. Rowe, N. (2006). Measuring the effectiveness of honeypot counter-counterdeception. Proc. 39th Hawaii International Conference on Systems Sciences, Poipu, Hawaii.

11. Roesch, M. (1999). Snort - Lightweight Intrusion Detection for Networks. Proc. of the 13th USENIX Conference on System Administration, Seattle, Washington.

12. Somayaji, A., and Forrest, S. (2000). Automated Response Using System-Call Delays. Proceedings of the 9th USENIX Security Symposium, Denver, Colorado.

13. Rowe, N., Duong, B., and Custy, E. (2006). Fake honeypots: a defensive tactic for cyberspace. 7th IEEE Workshop on Information Assurance, West Point, New York.

14. Jones, K. J., Bejtlich, R., and Rose, C. W. (2006). Real Digital Forensics: Computer Security and Incident Response. Upper Saddle River, New Jersey: Addison-Wesley.

15. Rowe, N., and others. (2007). Experiments with a testbed for automated defensive deception planning for cyber-attacks. Proceedings of Second International Conference on I-Warfare and Security, Monterey, California.

16. Pang, R., and others. (2004). Characteristics of Internet Background Radiation. Proceedings of Internet Measurement Conference, Taormina, Sicily, Italy.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Dr Neil Rowe
   Naval Postgraduate School
   Monterey, California

4. Mr. Daniel Warren
   Naval Postgraduate School
   Monterey, California

5. Han Chong, Goh
   Naval Postgraduate School
   Monterey, California