2013-03

A Technique for Presenting a Deceptive Dynamic Network Topology

Trassare, Samuel T.

Monterey, California: Naval Postgraduate School

http://hdl.handle.net/10945/32911

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

A TECHNIQUE FOR PRESENTING A DECEPTIVE DYNAMIC NETWORK TOPOLOGY

by

Samuel T. Trassare

March 2013

Thesis Advisor:	Robert Beverly
Second Reader:	David Alderson

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)* 26–3–2013	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From — To)* 2102-06-06—2013-03-29

4. TITLE AND SUBTITLE

A Technique for Presenting a Deceptive Dynamic Network Topology

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Samuel T. Trassare

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Department of the Navy

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

13. 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.IRB Protocol Number: N/A

14. ABSTRACT

Adversaries scan Department of Defense networks looking for vulnerabilities that allow surveillance or the embedding of destructive malware weapons. In cyberspace, adversaries either actively probe or passively observe defended computer networks in attempts to determine, among other attributes, the topology of the network. We develop a novel strategic deceptive methodology, based on principles of military deception, for deceiving a malicious traceroute probe in defense of a physical data communications network. We construct a proof-of-concept network to show that a remote adversary who uses traceroute to map the defended network's topology can be presented with a false route of the defender's choosing. Akin to military deception operations in the field and at sea, a network that employs a deception scheme implemented on an intelligent border router can present a deceptive topology to an adversary. Our experiments show that a defender using our technique can successfully deceive a traceroute probe, the first in a sequence of steps to mount a credible deception scheme against an adversary.

15. SUBJECT TERMS

Topological deception, military deception, traceroute, network defense

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(include area code)*
Unclassified	Unclassified	Unclassified	UU	69	

THIS PAGE INTENTIONALLY LEFT BLANK

A TECHNIQUE FOR PRESENTING A DECEPTIVE DYNAMIC NETWORK TOPOLOGY

Samuel T. Trassare
Lieutenant, United States Navy
B.S., Computer Science, San José State University, 2005

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN CYBER SYSTEMS AND OPERATIONS

from the

NAVAL POSTGRADUATE SCHOOL
March 2013

Author:	Samuel T. Trassare
Approved by:	Robert Beverly Thesis Advisor
	David Alderson Second Reader
	Cynthia Irvine Chair, Cyber Academic Group

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Adversaries scan Department of Defense networks looking for vulnerabilities that allow surveillance or the embedding of destructive malware weapons. In cyberspace, adversaries either actively probe or passively observe defended computer networks in attempts to determine, among other attributes, the topology of the network. We develop a novel strategic deceptive methodology, based on principles of military deception, for deceiving a malicious traceroute probe in defense of a physical data communications network. We construct a proof-of-concept network to show that a remote adversary who uses traceroute to map the defended network's topology can be presented with a false route of the defender's choosing. Akin to military deception operations in the field and at sea, a network that employs a deception scheme implemented on an intelligent border router can present a deceptive topology to an adversary. Our experiments show that a defender using our technique can successfully deceive a traceroute probe, the first in a sequence of steps to mount a credible deception scheme against an adversary.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

1 INTRODUCTION 1

1.1 Application to the Department of Defense 1

1.2 Pervasive Network Probing 1

1.3 Military Deception in Cyberspace. 1

2 BACKGROUND AND REVIEW OF LITERATURE 5

2.1 Traceroute Basics 5

2.2 Tools 8

2.3 Review of Literature 10

2.4 Contributions in Context 13

3 METHODOLOGY 15

3.1 Design Overview 16

3.2 Implementation Requirements 16

3.3 Network Design. 18

3.4 Software Selection. 19

3.5 Vulnerability Analysis 20

3.6 Kernel Module Implementation Details 23

3.7 Noise 29

4 FINDINGS 31

4.1 Experiment Overview 31

4.2 Most Vulnerable Node 31

4.3 Physical Route 32

4.4 Deceptive Route. 33

4.5 Network Performance Impact 33

5 CONCLUSION AND FUTURE WORK 35

5.1 Alternative Approaches 35

5.2 Future Work 38

5.3 Adversary's Assessment 43

5.4 Concluding Remark 43

Initial Distribution List 51

List of Figures

Figure 2.1 An example of traceroute output. 7

Figure 3.1 Experiment topology implemented in virtualization. All Internet Protocol (IP) addresses are on the 192.168.000.000 subnet. 19

Figure 3.2 Example of betweenness centrality. Node 2 in (a) has the highest betweenness centrality (C_B), 1. In (b) all nodes have equal C_B, 0. 20

Figure 3.3 The kernel configuration file for the deception implementation. 22

Figure 3.4 This function returns an edge whose inclusion in graph g produces the lowest C_B. 23

Figure 3.5 This function identifies incoming traceroute probes and conditionally modifies the time to live (TTL) value. 25

Figure 3.6 This function provides a deceptive Time Exceeded or Port Unreachable message to the adversary. 26

Figure 3.7 Linux packet routing scheme. The circles provide representation of the available Netfilter hooks. After [16]. 27

Figure 3.8 Traceroute results with a “noisy” deception deployed. 28

Figure 4.1 The deceptive route implemented. The solid arcs represent the truthful route provided to the adversary in response to a traceroute probe. The dashed arcs represent the deceptive route supplied by the intelligent router. 32

Figure 4.2 Traceroute results (prior to deception). Device labels from Figure 4.1 are added for ease of reference. 33

Figure 4.3 Traceroute results (after deception). Device labels from Figure 4.1 are added for ease of reference. 34

Figure 5.1 Alternative implementations of deception. 37

Figure 5.2 Topological deceptions must be consistent across multiple ingress points. 41

List of Tables

Table 3.1 Options for defending a network. Denial compared to deception. 15

Table 3.2 Variables initialized at kernel load time. 24

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

AS	autonomous system
C_B	betweenness centrality
CJCS	Chairmain, Joint Chiefs of Staff
DNS	Domain Name System
DoD	Department of Defense
EW	electronic warfare
FTP	File Transfer Protocol
GNS3	Graphical Network Simulator
HTTP	Hypertext Transfer Protocol
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IO	information operations
JP	Joint Publication
KB	Kilobytes
MB	Megabytes
Mbps	Megabits per second
MILDEC	military deception
ms	milliseconds
NIC	network interface card
NSS	National Security Strategy
OS	operating system

OSI	Open System Interconnection
QDR	Quadrennial Defense Review
RF	radio frequency
RFC	Request For Comment
RIP	Routing Information Protocol
SNOS	Systemic Network Obfuscation System
SSH	Secure Shell
TCP	Transmission Control Protocol
TTL	time to live
UDP	User Datagram Protocol
VM	virtual machine
VPN	Virtual Private Network

Acknowledgements

I would like to thank Dr. Robert Beverly for supporting this endeavor. His insight and guidance helped me muddle through the most challenging aspects of this research. I genuinely appreciate the number of times he put my experiments back on track by correcting my code without even looking at it. I also would like to thank Dr. David Alderson for offering a different perspective on the formulation of this work and for illustrating the connections between deception in cyberspace and in the physical domain.

I owe a great deal to both family and friends for their support as I pursued my graduate studies. To my good friend Michael Rhoades: Thank you for the support you provided to me and to my family in my absence. You have gone above and beyond the call of friendship to make sure that my family was well during my deployments. To my fishing buddy and father-in-law, Raymond Waugh: Thank you for reading multiple drafts of this thesis and providing your valuable input and encouragement. To my grandmother, Santa: You are the truest example of grace and class. Thank you for the unconditional support you have provided to me and to my wife. To my daughter, Santa Cecilia: Your curiosity and playfulness are inspiring. I am especially thankful for all the perfectly legitimate reasons you provided for me to take a break from my studies. Finally, to my wife, Harmony Trassare, MSN, RN: Not only have you supported me but you've empathized with me, having already walked this road. Your kindness, patience and tolerance is never-ending.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1: INTRODUCTION

In this chapter we discuss the applicability of the research contained herein to the Department of Defense (DoD) and the pervasiveness of the threat of computer network probing as well as offer a technique based on deception for countering the threat.

1.1 Application to the Department of Defense

Cyber warfare is moving to a more central position in U.S. national security concerns. The 2010 National Security Strategy (NSS), Quadrennial Defense Review (QDR), and the 2011 DoD Strategy for Operating in Cyberspace each place greater emphasis on defending cyberspace than in the past. In a 2012 speech given to the Business Executives for National Security in New York, Defense Secretary Leon Panetta warned of an impending "cyber Pearl Harbor; an attack that would cause physical destruction and the loss of life. In fact, it would paralyze and shock the nation and create a new, profound sense of vulnerability. [1]" As the importance of cyberspace as a war-fighting domain grows, so will the need for a robust toolkit of cyber war-fighting techniques. The research presented herein offers a novel technique that may be added to the suite of tools available to defend DoD networks.

1.2 Pervasive Network Probing

In cyberspace, adversaries either actively probe or passively observe defended computer networks in attempts to determine, among other attributes, the topology of the network. Both DoD and corporate networks alike, in the U.S. and around the world, are probed thousands of times a day for vulnerabilities and points of access [2].

The intuitive response to such near-constant probing may be to *selectively deny* the adversary's probes from entering the network while allowing all other traffic to pass. Unfortunately, it is hard to distinguish legitimate traffic from malicious probes, and it is nearly impossible to know with certainty that an adversary is successfully blocked from probing a defended network.

1.3 Military Deception in Cyberspace

Instead of merely denying the adversary a vantage into a defended network, this thesis examines if there is value to be gained in employing the principle of military deception by presenting a false network. Such deception can frustrate an adversary's ability to gain usable intelligence from a target and thus prevent exploitation of the network.

The use of military deception in warfare dates back to ancient times. The most well-known example is that of the Trojan Horse, a ruse the Greeks used to infiltrate and sack the city of Troy during 12th century BC. Today, deception tactics abound, including placement of inflatable mock-ups of tanks, airplanes made of wood, warships with lighting rigged to appear as merchant vessels, and intentional disclosure of false intelligence that purports to reveal upcoming operations. In the field of electronic warfare (EW), radar systems are capable of jamming an adversary's signal or subtly modifying its return so as to alter the perceived distance to its target.

Military deception is one of the pillars of information operations (IO) and has long been used as a viable and valuable military tactic at sea, on land and in the air. Joint Publication (JP) 3-13.4 defines military deception (MILDEC) as:

> Those actions executed to deliberately mislead adversary decision makers as to friendly military capabilities, intentions, and operations, thereby causing the adversary to take specific actions (or inactions) that will contribute to the accomplishment of the friendly mission. [3]

It is reasonable, therefore, to expect value in deceiving adversaries within the confines of cyberspace. To that end, we explore the use of deception in the cyber domain. We present here a novel technique for *topological deception* in which an adversary is presented with a false network topology that purposefully conceals and disguises the underlying true topology.

This thesis focuses on adversaries that are performing active probing. Active probing tools available to the adversary include traceroute, ping sweeps, and port scanners. A discussion of these tools is offered in § 2.2. These tools produce a wide range of effects that are advantageous to the adversary. The intelligence gained from such probing allows an attacker to gain valuable insight into which nodes in the network would, if disabled by attack, yield the greatest overall impact (e.g., network disruption, or other objective). In particular, this research is concerned with the effect of attacks on "weak" nodes within the topology that might partition the network. A network is partitioned when a high-value node is disabled causing significant segments of a network to lose connection.

In order to prevent a partitioning attack, we offer a defense based on the principle of deception in which the *outward appearance of a network topology is altered* and presented to an attacker for the purpose of protecting high-value nodes or links within the network. Thus we may cause the adversary to take specific actions, such as attacking highly fault-tolerant nodes that appear weak, or avoiding weak nodes that appear highly fault-tolerant.

It is possible to present a deceptive network topology through the careful manipulation of the responses to an adversary's probes. This thesis proposes a novel deception methodology, then demonstrates the feasibility of presenting a deceptive network topology to an adversary. The deception is implemented via the manipulation of outbound traffic for the purpose of deceiving a traceroute scan.

An outer layer of defense in the form of an intelligent router, one that is capable of identifying traceroute scans and providing a deceptive route in reply, may thwart an adversary's efforts to subvert critical nodes and links in defended networks. The intelligent router presents the deceptive route while allowing all other traffic into the network. Such an approach subverts a traceroute scan while allowing other forms of traffic to flow in and out of the network unimpeded. Our research proceeds in this direction.

By developing a proof-of-concept network, we demonstrate a strategic deceptive methodology for deceiving a malicious traceroute scan in defense of a physical data communications network. Akin to military deception operations in the field and at sea, the deception will present adversaries running traceroute with a deceptive route to a protected target. The deceptive route is constructed such that it causes the adversary to incorrectly ascertain the relative position (and therefore, the value) of a node in a network. The methodology accommodates any true input topology, while the deceptive topology can be modified easily and frequently if necessary to further confound the adversary's efforts to identify vulnerabilities in the physical network.

By influencing the adversary's decision-making, he can be lured into a course of action that is favorable to the defender's objectives. The methodology proposed here could potentially be used to entrap adversary attacks, which would then permit forensic analysis and defense against similar attacks in the future. Furthermore, due to the highly flexible nature of the implementation, an adversary may be kept longer in an intelligence collection phase of their mission while attempting to pin down the exact topology of a network. From the adversary's perspective, the topology may appear substantially more complex than its true structure, or may be made to appear as though it is continually and unpredictably changing. This prolonged stay in an intelligence collection phase may prevent the adversary from transitioning to an operational phase.

This thesis demonstrates how a physical network may be disguised by an intelligent router capable of manipulating outbound traffic. The manipulated traffic may provide the perception of a network that resembles, or completely disguises, the underlying physical network while having the ability to easily vary any attribute of the network, such as node count or the redundancy of links between vital nodes. The work described herein has not been deployed on a production network, nor has the method been tested against human participants. However, we introduce a working implementation of deception in a Linux kernel module executing as an intelligent router. The execution of the kernel module is shown to work within a synthetic network topology with an adversary probing the path to a publicly available service (a web server) whereby the attacker infers the deceptive (false) route. The resilience of the deceptive network presented to the adversary may be compared, for instance using graph theoretic measures, against the true physical topology. Such comparisons are left to future work.

The remainder of this thesis is organized as follows. In Chapter 2, we explore the background of cyber deception and present a representative sample of literature on the topic. In Chapter 3, we discuss the details of the topological deception implemented herein. In Chapter 4, we share the findings of our experiments. Our conclusion in presented in Chapter 5.

CHAPTER 2:
BACKGROUND AND REVIEW OF LITERATURE

Deception in cyberspace is not a new practice and many researchers have examined the various aspects of cyber deception employed by attackers and defenders. Before reviewing a representative sample of the body of work that explores deception in the physical domain and in cyberspace, we first explore some of the tools used for cyber denial and deception. We begin this chapter with an explanation of the workings of the traceroute program.

2.1 Traceroute Basics

Before describing traceroute, we take a moment to describe network ports, User Datagram Protocol (UDP) and Internet Control Message Protocol (ICMP).

2.1.1 Ports

In computer networking, ports are used as a method of permitting multiple services (software) access to a single network interface without interfering with each other. Ports are defined by a 16-bit number and are concatenated to an Internet Protocol (IP) address thus providing a complete address to a service on a computer. A port that is bound to a running service is said to be “open.”

2.1.2 User Datagram Protocol

UDP is a component of the IP suite. UDP is designed to be fast by imposing a minimum of protocol overhead. UDP packets are prefixed with an IP header containing the source address, destination address, and a time to live (TTL) field. UDP is fast because it sacrifices elements available in Transmission Control Protocol (TCP) such as ordered delivery of packets, delivery confirmation and duplication avoidance [4]. Of note, some traceroute implementations use TCP packets since they are able to pass through firewalls which are typically configured to allow TCP.

Relevant to this research, the header of a UDP packet contains, among others, fields for the source port and destination port. The destination port of a UDP packet may be any 16-bit value. Traceroute, by historical convention, uses the port range of 33434 to 33534 when sending UDP packets to a destination. Traceroute uses a range of ports rather than a single port to allow for multiple simultaneous executions of traceroute on a single host with replies being matched to the original execution.

2.1.3 Internet Control Message Protocol

ICMP was designed to support datagram error reporting between hosts [5]. ICMP messages are described by various types with each type reporting to the recipient the type of error that occurred.

Programmers such as Muuss [6] employed ICMP to develop some of the earliest TCP/IP probing tools. One simple type of probe uses the Ping command to systematically scan for IP addresses in use. The Ping command sends out an ICMP echo request to a specified host and waits for echo reply messages. In cases where ICMP traffic is not blocked by a firewall or ignored by the destination host, Ping is able to determine if a host or IP address is up and active on a network [7].

2.1.4 Traceroute

Traceroute is a network diagnostic tool for reporting nodes in a path to a destination and the delay times associated with each node. Nodes are typically routers and may also be computers configured to act as routers. Traceroute reports only the forward path in a route. For routers, which have multiple network interfaces, the forward path means that only the interface nearest the prober is reported by a traceroute probe. UDP is the default packet transmission method used by traceroute on the Linux operating system (OS) (the tracert program for the Windows OS uses ICMP by default). The research presented in this thesis focuses on traceroute's use of UDP only.

A typical traceroute command takes a form such as:

```
# traceroute 192.168.5.2
```

The result this command generates is shown in Figure 2.1. The first column is an incrementing count of nodes discovered between the prober and the destination. The IP address for each discovered node is shown in the second column. By default, traceroute sends UDP packets in groups of three meaning that for each TTL value, three probes are sent. The next three columns show the round-trip time taken for each probe. The round-trip time is the time between sending

```
traceroute to 192.168.5.2 (192.168.5.2), 30 hops max, 60 byte packets
 1  192.168.9.1 (192.168.9.1)  2.859 ms  5.490 ms  7.707 ms
 2  192.168.0.2 (192.168.0.2)  9.673 ms  11.539 ms  14.296 ms
 3  192.168.1.2 (192.168.1.2)  16.250 ms  18.176 ms  20.255 ms
 4  192.168.2.2 (192.168.2.2)  21.723 ms  23.669 ms  25.591 ms
 5  192.168.3.2 (192.168.3.2)  31.812 ms  33.677 ms  35.675 ms
 6  192.168.4.2 (192.168.4.2)  43.832 ms  18.201 ms  18.853 ms
 7  192.168.5.2 (192.168.5.2)  22.655 ms  24.371 ms  26.258 ms
```

Figure 2.1: An example of traceroute output.

the UDP packet to receiving the matching ICMP Time Exceeded message.

To understand how traceroute finds this route we look to its control of the TTL field and its use of UDP and ICMP. The TTL field is controlled by traceroute, with the first packet in a probe having TTL=1. For each UDP packet sent during a probe, traceroute expects an ICMP Time Exceeded packet in reply (or a Port Unreachable message when the destination is reached). Traceroute prepares a UDP packet for delivery for the first group of three packets in a probe. In our example, the destination address for every packet in the probe is 192.168.5.2. The TTL of the first packet is set to one. After the probing computer sends out the packet, it arrives at the next nearest node (192.168.0.2) along the route to the destination. The receiving node decrements the TTL. For the first packet, the action sets the TTL to zero, thus causing the packet to expire. The node will not allow the expired message to continue on and generates an ICMP error message of type Time Exceeded and sends it back to the prober. Recall, that the default operation of traceroute is to send UDP packets in groups of three thus the first three packets in the probe have TTL=1 and 192.168.0.2 responds to all three in identical manner.

The second group leaves the probing host with TTL=2. The next node decrements the TTL of each node as before, however, the TTL has not yet expired so the node forwards the packet. In our example, 192.168.1.2 receives the next group of three packets. This node, like the one before it, decrements TTL and replies to the prober with the ICMP Time Exceeded error message. This process continues until a group of three packets finally arrives at the destination (192.168.5.2).

The destination computer, upon receiving the packets, attempts to deliver them to the port requested by the prober. Recall that traceroute uses ports in the range of 33434 to 33534 which is a registered range as specified by Internet Assigned Numbers Authority (IANA). Reserving the port range for traceroute implies that no other service should use this range. Assuming no other service is running in this range, when traceroute attempts an access in this range on the destination computer, the destination computer is obliged to reply with an ICMP Port Unreachable message. When the Port Unreachable message is received by the prober, the probe is ceased and traceroute exits execution.

2.2 Tools

In addition to traceroute, there are two other primary tools used by an adversary to gain intelligence on a target network. Those tools are ping sweeps, and port scanners. Here we discuss how an adversary uses these three tools and the defender's countermeasures against them.

2.2.1 Ping Sweeps

The standard Ping implementation limits its user to ping only one host at a time. For an adversary, it is very time consuming to probe only one host at a time. There are workarounds to this limitation such as wrapping the Ping program in a shell or batch script. Doing so enables many hosts, even entire subnets, to be quickly probed and thereby provide the adversary with a better understanding of the available attack surface. Better yet, improved implementations of Ping exist.

Fping [8], which stands for "fast pinger," solves the limitation of being able to ping only one host at a time. By taking a list of IP addresses, either from standard input or in a flat text file, Fping is able to scan entire networks much faster than would be possible with the standard Ping utility [7]. Using the results of an Fping probe, an adversary may develop a list of potential target hosts on a network.

The standard Ping program is limited not only by its ability to ping one host at a time but also in that it sends pings using ICMP only. Routers may be configured to block ICMP traffic, and hosts may be configured to ignore ICMP. The Hping [9] program avoids such a limitation by eliciting the same behavior from a target host that an ICMP ping would cause by using UDP packets or TCP instead. It does, however, support ICMP as well.

2.2.2 Port Scanning

Port scanning involves probing a host for open ports. Its use is described by Request For Comment (RFC) 4949, "A port scan can be used for pre-attack surveillance, with the goal of finding an active port and subsequently exploiting a known vulnerability of that port's service. [10]"

Nmap [11] is the most commonly used port scanner today. Like Hping, it bypasses Ping's limitation of using only ICMP by using UDP and TCP. Nmap is a robust program that not only determines whether a host exists at an IP address but also which ports, and therefore which services, are available on that host [7].

2.2.3 Topology Mapping via Traceroute

Traceroute is one of many tools used to infer topology, but doing so is no trivial task. Traceroute sends probes using UDP and relies on ICMP replies to gather information regarding the route between the prober and the destination. A traceroute scan receives two types of ICMP messages: Time Exceeded and Destination Unreachable. There are six sub-types of a Destination Unreachable Message. In particular, traceroute receives the Port Unreachable sub-type.

Traceroute, used alone or in concert with other tools, is not a fool-proof method of mapping a target network. Intentional and unintentional outside factors may affect the results obtained in a traceroute probe. Routers may be configured to block UDP traffic thus thwarting a traceroute probe. Routers may also be configured to prohibit sending ICMP messages which may also thwart a probe. Furthermore, network congestion, queuing, network maintenance and other conditions may affect the results provided by a traceroute probe.

There has been significant work in inferring Internet topologies. However, because of the Internet architecture, doing so is fundamentally difficult and the use of inference processes introduce errors. These errors can include false links, missing links, etc. Willinger *et al.* [12] show the ease with which such mistakes can be made. We turn this to our advantage by exploiting the ease to which an adversary attempting to map the network may be deceived.

2.2.4 Firewalls

Firewalls are the de facto standard for denial of network access to adversaries. Oppliger [13] describes the operation of firewalls best saying, "a firewall builds a blockade between an internal network that is assumed to be secure and trusted, and another network, usually an external (inter)network, such as the Internet, that is not assumed to be secure and trusted." Many popular examples of firewall software for computers exist such as ZoneAlarm [14] and Comodo [15] for Windows systems and iptables [16] for Linux systems. Firewalls exist for both personal computers and for servers.

Routers have firewall functionality included as they are a vital component of network defense. Organizations construct networks using the notion of autonomous systems (ASs), in which the organization monitors all of its own resources [17]. *Border routers*, those that connect an AS to the rest of the Internet, have integrated firewalls in order to provide defense of the network against threats from the Internet at large.

Firewalls are configured by the user to deny forwarding of network traffic based on hierarchies of rules. Traffic types, such as ICMP, may be blocked. Port ranges, such as that of traceroute (33434 to 33534), may be blocked. Other criteria such as IP addresses, flags and options may also be used to block traffic.

2.2.5 Honeypots

Honeypots are machines that appear to be vulnerable computers but are actually programs designed to lure attackers. They offer a form of deception that tricks an attacker into believing they have uncovered valuable information, or tricks them into exposing their attack methods while attempting to compromise the honeypot. Honeyd [18], HoneyClient [19], The Honeynet Project [20] and Honeymonkey [21] are examples.

LaBrea [22] is a form of honeypot that takes over unused IP addresses within a network address space and attempts to answer all connection attempts from outside sources. This method is especially useful against automated attacks as it can keep the attacking script busy for a long time by continually deceiving the script into acting as if the service for which it is probing is available. Furthermore, this slows down the attacker's probe rate by tricking the attacker into a potentially long wait period. LaBrea is most directly related to the research presented here in that it attempts to deceive an attacker of the existence of devices at unused addresses. Furthermore, it attempts to simulate the existence of running services at the deceptive address.

2.3 Review of Literature

Today, there are at least two methods of employing deception in cyberspace. The first is through *obfuscation*, in which defensive software tools mask identifying attributes of a host computer to prevent "fingerprinting. [23]" The second method, the use of *honeypots*, lures the adversary into exploiting seemingly vulnerable computers which are, in reality, programs running on a host computer expressly for the purpose of deceiving an adversary into exposing his or her tactics [24]. What follows is a discussion of representative research of MILDEC in the physical domain. We also discuss research that illustrates how deception in the physical domain has been adapted to implement obfuscation and honeypots in the cyber domain.

2.3.1 Military Deception in the Physical Domain

JP 3-13.4, released by the Chairmain, Joint Chiefs of Staff (CJCS) on 13 July 2006, provides the definition of MILDEC used by the U.S. armed forces. According to JP 3-13.4, MILDEC comprises six principles:

1. focus: the deception must target the adversary decision maker capable of taking the desired action(s);
2. objective: the deception must cause an adversary to take (or not to take) specific actions, not just to believe certain things;
3. centralized planning and control: MILDEC operations should be centrally planned and directed in order to achieve unity of effort;
4. security: friendly forces must deny knowledge of a force's intent to deceive and the execution of that intent to adversaries;
5. timeliness: a deception operation requires careful timing; and
6. integration: fully integrate each military deception with the operation that it is supporting.

JP 3-13.4 also provides guidance on determining the goals and objectives of deception operations [3].

Fowler and Nesbit [25] provide several examples of tactical MILDEC in air and land warfare and identify six "rules" a deception must follow if it is to be successful. First, "a deception must be one that causes the enemy to believe what he expects." Second, timely feedback on the enemy's reaction to a deception is essential. Third, the deception must be integrated with other operations. Fourth, preventing the disclosure of information regarding true activities is essential. Fifth, the level of realism required by the deception is a function of the adversary's capability to analyze the situation. Finally, "the most effective deception will be imaginative and creative."

Whaley [26] describes deception in two categories: hiding the real, or *dissimulation*; and showing the false, or *simulation*. According to Whaley, dissimulation is that part of a deception that is concealed from the adversary. Simulation is the overt part of a deception presented to the adversary. The research presented herein employs both categories.

2.3.2 Military Deception in the Cyber Domain

From the perspective of Whaley's taxonomy, we consider the work done by others in exploring the denial of information. What Whaley calls dissimulation is also known as obfuscation. For example, consider an implementation of TCP/IP in which the generated traffic contains artifacts that allow their host OSs to be identified [27]. Smart *et al.* [28] describe the design and implementation of a scrubber that obfuscates the behavior of the TCP/IP stack to prevent identification of the host OS. This concealment of the host identification is a form of dissimulation.

Rowe and Rothstein [29] consider Fowler and Nesbit's rules and adapt them to application in cyber war. They also consider Dunnigan and Nofi's proposed taxonomy of deception [30] and provide an assessment of how useful each type of attack is in information systems. Dunnigan and Nofi identify nine types of deception. Applicable to our discussion is *concealment* (what Whaley calls dissimulation) and *lies* (which may include simulation). In Rowe's assessment, "concealment of resources" is relatively unimportant (assessed 2 out of 10 on an internal scoring system). Using the deception methodology presented in this research, concealment of resources is quite valuable, requiring at times, that nodes be concealed from an adversary's view. Rowe and Rothstein assess "lies" to be very important (i.e., a score of 10 out of 10) which is consistent with our research. As Rowe and Rothstein point out, "The best things to lie about could be the most basic things that matter to an attacker: the existence of resources and his or her ability to use them. [29]" The research presented herein will not only conceal resources but lies to an adversary about the existence of a resource (network node) that does not exist.

In an example of dissimulation, Huber [31] evaluates the effectiveness of an original approach to network-based obfuscation known as Systemic Network Obfuscation System (SNOS). SNOS is a Windows-based program that attempts to defeat host identification based on its network traffic, and it is similar to the work done by Smart *et al.* SNOS is capable of manipulating network traffic at any layer of the Open System Interconnection (OSI) model except for the physical layer. It manipulates data fields within passing network traffic to alter certain features that would reveal the host OS of the sending device. Huber evaluates the effectiveness of SNOS against an existing obfuscation tool known as OSfuscate [32]. In addition to its effectiveness, SNOS is also evaluated for performance based on its impact to network latency.

The work presented by Smart *et al.* [28] and Huber [31] employ only the principal of dissimulation to obfuscate network hosts. The research presented herein obfuscates network hosts in that the deceptive traffic to the adversary does not reveal the host OSs of the routers within the defended network. This was not an intentional design specification but rather a by-product of our implementation. In addition to obfuscating the routers in a defended network, our implementation also attempts to deceive an adversary as to the existence of routers through simulation.

Beyond obfuscation research, the work presented by Frederick [33] explores the state of the art in honeypots by testing the low-interaction honeypot software, Honeyd [18], on an Internet-facing network without the protection of a firewall. Honeypots perform by Whaley's definition of simulation by providing the illusion of an existing network resource.

Zhang *et al.* [34] describe the use of honeypots as an active defense for networks, survey the current state of the art, and classify the typical uses of honeypots in network defense.

2.4 Contributions in Context

In summary, this thesis makes the following contributions: (1) Introduces a novel methodology for topological deception, (2) implements in a Linux kernel module the described methodology, and (3) demonstrates the workings of such in a synthetic network.

An adversary using the traceroute program may be deceived as to the physical topology of a computer network through the careful manipulation of traffic leaving the network in response to the traceroute probe, thus masking the importance of key nodes along the actual route.

The implementation provided herein is demonstrated to work in a laboratory environment however evaluation of its ability to deceive real adversaries in a production network environment is left to future work. In § 5.2 we offer a discussion of possible considerations for evaluating the usefulness of the deception implementation.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
METHODOLOGY

Network defense is an operational reality. A spectrum of options is available to protect a network from adversarial probing and scanning. Table 3.1 presents three options that we review briefly. The options are not an exhaustive list of the methods of network defense available, but rather, speak to defense in terms of what kind of traffic to permit.

One option for network defense is to deny access to all services within the network. This is a seemingly easy and straightforward approach to insulate the network from attack. Unfortunately, *complete denial* of all traffic also impacts the network's usability for legitimate users and thus can defeat the purpose of the network (e.g., to share resources or content). Services provided by the network should be available to legitimate remote users even when the network is under attack.

Another option is to deny certain services (*selective denial*) thereby allowing only legitimate traffic in as identified by particular identifying protocol fields. For example, ICMP traffic may be blocked at a border router while TCP traffic is allowed in. Such an approach allows Internet-facing services such as web servers to provide service, but denies the defender from gaining any intelligence on how the adversary, for example, uses ICMP. Furthermore, it denies the defender the opportunity to place false information. Note that, in general, selective denial is difficult to do especially when encryption is used. It is also made difficult when several types of traffic are made to ride over port 80, the standard port for Hypertext Transfer Protocol (HTTP). In such cases, identifying traffic for selective denial is a field of study in its own right.

Technique	**Pros**	**Cons**
Complete Denial	Insulates the protected network.	Isolates the protected network.
Selective Denial	Permits availability of select services.	Denies the defender insight into enemy attack.
Deception	Offers protection of network. Offers defender some insight into enemy attack.	Requires robust implementation.

Table 3.1: Options for defending a network. Denial compared to deception.

A final option is to leave the network completely open in the sense that all services are allowed. Doing so would permit forensic analysis of an adversary's attacks, but may unduly increase the risk of harm to the network. However, it is possible to use *deception* on an open network to lure an adversary to attack false targets or to dilute his or her attack. Deception may offer the ability to defend the network in additional ways. Despite the advantages deception offers the defender, it requires a robust implementation to account for the many tools an adversary may employ to gain intelligence against a target network.

3.1 Design Overview

Traceroute probes are active probes, most typically used for diagnostics and troubleshooting. Probes are sent on demand by a user or an automated task, and may be directed toward any device (IP destination) within a network. These active probes, by definition, involve injecting packets that elicit some response from the target(s) in an attempt to fingerprint, characterize, or map the network. As a result, the best place to affect the results of such mapping are at the ingress or egress points to the network. An ingress point is a border router for an AS (introduced in §2.2.4) that permits traffic into the network. Therefore, we consider a strategy whereby deceptive functionality resides on the border routers of an AS. By deploying the deception at a border router, any designated device within the AS or the whole network is protected by the deception.

In the deception employed here, we focus specifically on deceiving an adversary's traceroute probe using an implementation deployed in a border router. As discussed in § 2.1.4 a traceroute probe elicits ICMP Time Exceeded messages. Crucially, the path reported to the prober is a function of the source addresses of the ICMP Time Exceeded messages. These ICMP messages are encapsulated in an IP packet, the source IP address of which may be from almost any IP address. Through experimentation we found that traceroute does not receive ICMP replies sent from Class D (multicast 224.0.0.0 to 239.255.255.255) or local host (127.0.0.0 to 127.255.255.255) addresses. As a result of this flexibility, there are many possible strategies to implement deception.

3.2 Implementation Requirements

In order for topological deception to be successful, it must satisfy several criteria. The criteria is divided into *constraints* (the deception must do something) and *restraints* (the deception must not do something).

Four operational constraints were imposed. The first constraint imposed, following from Fowler and Nesbit's [25] first rule, is that the deception must be "believable." Anyone probing the network must see something that appears to be the actual network. The idea is that the deception should present an *external consistency* in which the deception does not betray the truthful underlying network. For example, if a physical network provides a web server, the deception should not try to convince the adversary that no web server exists on the network. Conversely is the idea of *internal consistency*, that the deceptive network will look consistent from different vantage points outside the network. For example, for ASs that have multiple border routers, different deception schemes should be employed at each router but each deception scheme should reinforce, not contradict, another. For the experiment presented herein, this constraint is satisfied by the use of a single border router.

Second, the deception must account for two key metrics. The TTL of a packet and the time it takes for a packet to reach its destination are loosely correlated. Due to the loose correlation it may be possible that an inconsistency could allow for an adversary to ascertain that deception is in use. However, ensuring that these metrics are consistent is vital in upholding the plausibility of the deception. Because a single node is deploying the deception, it must send the packets with TTL values and delay times that enforce the illusion that packets are originating from nodes that are increasingly more distant from the adversary than the deceiving node. Should a deceptive packet be returned to the adversary too fast or in a time that is equal to the time taken for every other reply in the probe to be sent, the deception is exposed.

Third, the deception must be easily deployed or withheld. This feature facilitates making modifications to the deceptive topology. Additionally, the deception might need to be disabled during periods of network maintenance. Otherwise, leaving the deception running while the protected devices are legitimately offline would expose the deception. The issue of managing the deployment of the deception is an important one and may be left for a commander to decide. However, the operation of the deception may be left to network managers who must administer the deception. The issues of such management are left to future work.

Finally, the deceptive route must be easily configurable and flexible. Routine changes in the physical topology or changes in the deception method employed, will necessitate changes in the deceptive route. Furthermore, because an AS may have multiple border routers, the deceptive route must be tailored to be consistent from the perspective of each border router. Such flexibility of configuration enables us to meet the requirement of providing the internal consistency introduced with the first constraint.

In addition to the operational constraints, two restraints were imposed. First, the deception must not interfere with other mission-critical services offered by the defended network. The purpose of the deception is to foil attempts to accurately determine network topology. The purpose is not to prevent access to legitimate and necessary services offered by the network. Services such as File Transfer Protocol (FTP), Virtual Private Network (VPN), Secure Shell (SSH), web browsing, and others should not be impeded by the deployment of the deception unless such services were the specific targets of the deception. In the general case, the services provided by the network, such as those enumerated above, must be available to legitimate remote users.

The second restraint imposed is that the deception must only be employed on a per-destination basis, meaning that the deception is deployed only for designated IP addresses or prefixes. For an example of the necessity of this restraint, depending on the real topology, the deception would be exposed if for any destination IP address the same (deceptive) route was always returned. Consider the network topology shown in Figure 3.1. If the same route were to be given as a response to traceroute probes directed at every node in the network the adversary would surely surmise that some form of deception is being employed. For the experiment performed in this research, the destination IP address was that of a web server. A more robust deception would employ a slightly different deceptive route for every node on the network yet every deceptive route would be consistent with all others. Such is left to future research. It should be noted that a limitation of our implementation is that for a traceroute probe to any node other than the web server the truthful route is provided to the adversary. A more robust approach would be to employ variations on the deceptive route for each node in the network or for designated high-value nodes.

3.3 Network Design

We conducted an experiment using a simple three-node network comprised of the adversary's workstation, a target web server and an "intelligent" router separating the two. The intelligent router is the device charged with identifying incoming traceroute probes sent using UDP that are destined for the web server and, in return, providing deceptive responses to the adversary.

We also used more complicated network topologies in our experiments. We used a Watts-Strogatz [35] model to generate a synthetic topology for experimentation due to its random nature and its tendency to cause high clustering of nodes. We realize that this is not representa-

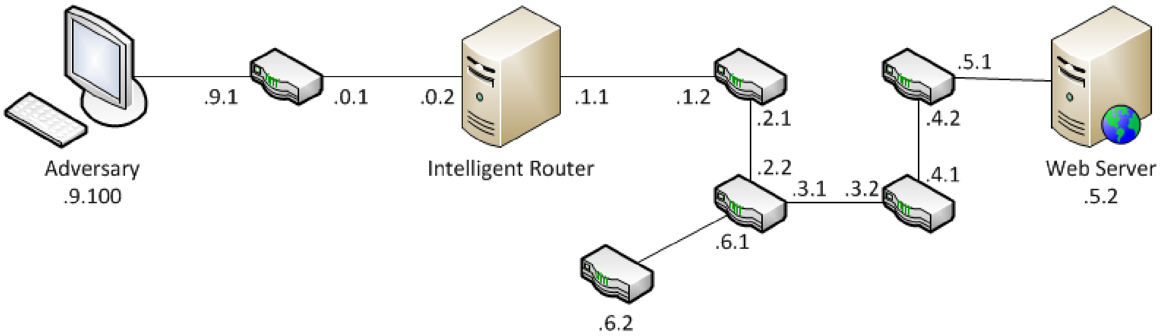


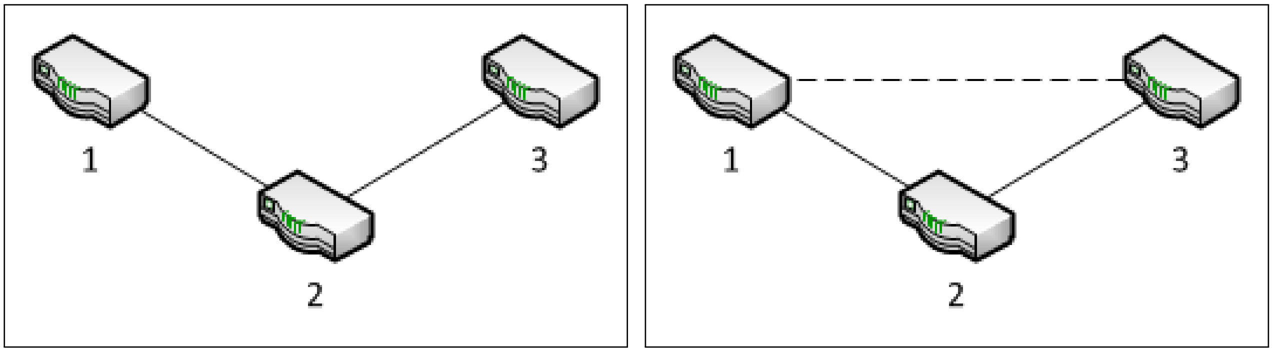
Figure 3.1: Experiment topology implemented in virtualization. All IP addresses are on the 192.168.000.000 subnet.

tive of any real network but we use it for our experiments due to the model's bias toward creating graphs with high clustering and short path lengths between pairs of nodes. Our implementation of topological deception is generalized to take as input any true network topology. Thus, for our experiments it is acceptable to use the a graph generational algorithm. The generated graph met two requirements for later experiments. The protected network had to contain at least four nodes and one of the nodes added to the original three-node model had to be included in the deceptive route being reported to the adversary's traceroute probe. This was necessary as adding only one new node to the protected graph would add no substantial value to the experiment. Furthermore, at least one of the new nodes had to be in the deceptive route to verify that its representation in the deceptive route would not conflict with its existence in the actual network. This supported our restraint of not interfering with mission-critical services. The resulting topology is shown in Figure 3.1.

3.4 Software Selection

A single workstation running Ubuntu 12.04.1 was used for all experimentation. We used Graphical Network Simulator (GNS3) [36] as the operating environment for all experiments as it allows virtual machines, routers, and the connecting links to be assembled and changed quickly without the overhead of assembling and reconfiguring candidate topologies in expensive hardware. Henceforth we assume that all hardware and software are components of the virtual environment provided by GNS3.

A virtual Cisco 3725 router, typical of small to medium sized offices, was used for each of the nodes described by the generated Watts-Strogatz model except for the intelligent router and the web server. The actual router used does not impact the results of our experiment. These



(a) Node 2 has C_B=1

(b) Each node has C_B=0

Figure 3.2: Example of betweenness centrality. Node 2 in (a) has the highest C_B, 1. In (b) all nodes have equal C_B, 0.

routers simply provided the node count necessary to implement a viable deception. Each router ran Cisco's IOS version 12.4(15)T14 and had Routing Information Protocol (RIP) enabled. We chose to use RIP for expediency of configuration, however we could have used static routing instead.

Both the adversary's workstation and the web server ran Lubuntu 12.04.1 [37] with kernel version 3.20. The adversary's workstation had traceroute installed. The web server had Apache installed. With these exceptions, both machines had an otherwise default installation. All three computers had Wireshark [38] installed to aid in debugging the software developed for the intelligent router.

The intelligent router used a lightweight Linux distribution known as SliTaz 4.0 [39] using kernel version 2.6.37. We selected this OS because the simplicity of its design facilitated fast kernel compile times, kernel module development and packaging of custom software as needed. In addition to the custom kernel module that provided the core functionality of this research, the intelligent router also ran Quagga [40], a networking protocol suite that provides an implementation of RIP for Linux systems thus allowing the intelligent router to communicate with the Cisco routers.

3.5 Vulnerability Analysis

An important consideration was identifying what constitutes a "high-value" node. A high-value node could be a device that is running a particular service or set of services, a router with known vulnerabilities or a workstation. Essentially, a high-value node is one whose exploitation would cause the biggest impact to the network. There are many metrics for evaluating the high-value targets of a network or the network's infrastructure. While we do not advocate any particular notion of value, we use the well-established notion of betweenness centrality in determining a node's value.

Betweenness centrality, often abbreviated C_B, is a measurement taken at either a node or a link that provides representation of how important the node or link is to the rest of the network. Freeman [41] describes the notion of point centrality as a structural property of betweenness, explaining, "a point in a communications network is central to the extent that it falls on the shortest path between pairs of other points." A node with high C_B has greater ability to relay or withhold communication between the pairs of nodes it connects compared to other nodes on the path between the same pair. It follows then that, should a node with high C_B be disabled by attack or other action, its inability to relay communication greatly impacts the pairs of nodes it connects.

Given a strategy for the deception type, and which nodes to protect, we ask the question, how can deception protect vulnerable nodes? Specifically, given the ability to add a single link (or, more appropriately, the illusion of such a link using deceptive techniques) to the graph in order to protect a designated "most vulnerable node," where should that link appear? By asking these questions we expect to find answers that allow us to minimize the maximum C_B in the network.

We tested two approaches to this question. The first approach was simple: find the two nodes in the network with the lowest C_B and connect them. The second approach, against which the first was compared, was to exhaustively enumerate each possible network, adding one link between every pair of nodes in the network, and returning the model of the network with the lowest overall C_B. The first approach produced results that were equal to that of the second approach 50% of the time for randomly generated input networks. However, the second approach proved to generate graphs that frequently showed lower C_B than the first. The results generated by the selected approach formed the basis of the deceptive route presented to an adversary's traceroute probe.

It is important to note that the second approach is inefficient in that the input network is scanned once for each combination of node pairs in the network. For n nodes, the network is scanned $\binom{n}{2}$ times. For example, a network with 8 nodes is scanned 28 times. Finding a more efficient method is left to future research.

To illustrate the point that a vulnerable node is one that has a high C_B, consider the three-

```
options deceptiveroute hop_addr=192.168.0.2,192.168.4.2,192.168.5.2
```

Figure 3.3: The kernel configuration file for the deception implementation.

node network shown in Figure 3.2. In Figure 3.2a node 2 has C_B=1. It is a member of the shortest path between nodes 1 and 3 and a successful attack on it would prevent those nodes from communicating. Comparatively, an attack on nodes 1 or 3 would not prohibit nodes 1 and 2 or 2 and 3 from communicating. Now consider Figure 3.2a in which a deceptive link has been added between nodes 1 and 3. Now, all nodes have a perceived C_B=0 and the vulnerability of node 2 is masked. This action does not prevent node 2 from being attacked. Rather, node 2 is less attractive as a target from the adversary's perspective.

Having determined that a vulnerable node is one with high C_B, a method was needed to find that node and minimize its C_B. For this we turned to the NetworkX [42] library for the Python [43] programming language. NetworkX provides two features useful for this experiment. It was used to generate a Watts-Strogatz model of the protected network that had a minimum of four nodes, yet were not connected strictly end-to-end. In addition, it is capable of reading input files that describe a graph and returning the C_B values for each node in the graph.

We developed a Python script that takes an input file describing a true network topology. The input file contains a list of all nodes in a network, each node's IP address and each of the links in the network. The script then performs an exhaustive search of the graph, testing the resulting overall C_B after adding a single new link between each pair of nodes in the graph. Figure 3.4 is the algorithm that describes this process. The output of the algorithm is what our deception will implement to minimize the maximum C_B.

NetworkX was then used to find the new shortest path between the intelligent router and the web server in the network with the minimized C_B. The output of this script is a description of the new shortest route, specified by IP address only, provided in a configuration file to be used as input to a kernel module. The resulting configuration file, named `deceptiveroute.conf`, is shown in Figure 3.3 where `deceptiveroute` is the name of the module and `hop_addr` is the name of the variable to be initialized with the following data (in this case, IP addresses).

The configuration file must be placed in `/etc/modprobe.d/` and, once the kernel module is compiled, loaded using:

```
Data: graph g
Result: The single new edge that produces min($C_B$) for graph g
min_betweenness = $\infty$;
for each node i in g do
    for each node j in g do
        if i != j then
            new edge e = (i,j);
            new graph h = g.copy();
            h.add_edge(e);
            max($C_B$) = find_max_$C_B$_in(h);
            if max($C_B$) < min_betweenness then
                minimum = max($C_B$);
                a = i;
                b = j;
            end
        end
    end
    return (a,b)
end
```

Figure 3.4: This function returns an edge whose inclusion in graph g produces the lowest C_B.

```
# modprobe deceptiveroute
```

The following section describes the implementation details of the kernel module.

3.6 Kernel Module Implementation Details

We considered two methods of deceiving a traceroute probe of the route to a protected node. In the physical domain, an EW suite has the capability to either saturate the radio frequency (RF) spectrum in response to an adversary's radar sweep or it may subtly alter the attributes of the adversary's radar sweep response so as to disguise the actual distance between the adversary and target. Likewise, a response to a traceroute probe may be "noise" in the form of Time Exceeded messages with randomly generated source IP addresses for each reply packet. Such a deception may prove valuable in frustrating automated traceroute probes as with the method employed by LaBrea [22]. Or, the deceptive response may be a plausible and complete path to the target. Both methods were implemented during our experimentation. The implementation of noise was the first milestone we set to prove the overall feasibility of deceiving traceroute. We

hop_addr	Initialized from configuration file.
arr_argc	Initialized to the number of IP addresses in hop_addr.
preserve_ttl	Initialized to zero. Stores incoming packet's TTL from start of prerouting hook until end of postrouting hook.
DEFAULT_TTL	Initialized to 65 per RFC 1700 with 1 added to account for programming logic.
router_ip	192.168.0.2
webserver_ip	192.168.5.2

Table 3.2: Variables initialized at kernel load time.

used a random number generator to generate random source IP addresses, what we considered to be the "noise." The remainder of our discussion focuses on providing a plausible path to an adversary's traceroute probe.

The heart of the deception lies in the implementation of a Linux kernel module. There is a route to the web server in which one of the hops is a node with the highest C_B value in the network. The configuration file used as an input to the kernel module describes a route that bypasses the vulnerable node, thus lowering its perceived C_B and making it a less attractive target for attack.

Several variables are initialized when the kernel module is loaded. Table 3.2 details the important variables. The variable arr_argc contains a count of the number of elements passed to hop_addr and represents the maximum number of hops provided in the deceptive route. The recommended default TTL value given by IANA [44] is 64. However, many vendors choose different values such as 32, 128 and 255. The Cisco routers used in our experiment have a default TTL of 255. The DEFAULT_TTL variable in the reply message may be set to one of these values if there is a desire to make the deceiving node appear to be a device from a different vendor. There are methods to determine a device type based on attributes other than its default TTL but masking those attributes is outside the scope of this research.

In order to deceive a traceroute scan and report to it the route described in the configuration file, the prerouting and postrouting hooks from the Netfilter [16] packet filtering suite were used. The prerouting hook is available to intercept a packet as soon as it arrives from the network interface card (NIC) with no processing by the host computer having yet occurred. Conversely, the postrouting hook is available to intercept a packet just before it leaves the host computer with no further processing to be done. See Figure 3.7 for an illustration of Netfilter's packet hooks. Figure 3.5 is a pseudocode listing for our preroute function. Figure 3.6 is a pseudocode

```
Data: socket_buffer skb
Result: For input packet with TTL>1, set TTL=0
ip_header = get_ip_header(skb);
if ip_header->protocol == UDP then
    udp_header = get_udp_header(ip_header);
    if (33434 >= udp_header->dest_port <= 33434) AND ip_header->dest_address ==
    webserver_ip then
        if ip_header->ttl == 1 then
            preserve_ttl = ip_header->ttl;
        else if ip_header->ttl > 1 AND ip_header->ttl < arr_argc then
            preserve_ttl = ip_header->ttl;
            ip_header->ttl = 0;
            ip_header->checksum == new_ip_checksum(ip_header, ip_header->length);
        else if ip_header->ttl == arr_argc then
            preserve_ttl = ip_header->ttl;
            ip_header->dest_addr = router_ip;
            ip_header->checksum == new_ip_checksum(ip_header, ip_header->length);
        end
    end
end
return ACCEPT_PACKET;
```

Figure 3.5: This function identifies incoming traceroute probes and conditionally modifies the TTL value.

listing for our postroute function.

In the prerouting hook, the packet is inspected to see if it matches certain criteria. Is it a UDP packet? Is it destined for a port in the range of 33434 to 33534? Recall from §2.1.2 that the 33434 to 33534 port range is reserved by IANA for traceroute. Is it destined for the protected web server? If the answer to all of these questions is "yes" then the kernel module begins employing the deception.

The deceptive hop that the kernel module chooses to send in response to the adversary's traceroute scan is dependent upon the TTL value of the incoming packet. Traceroute sends packets with incrementally increasing TTLs to identify every node in the path to a destination thus the first packet to arrive at a border router for an AS will always have TTL=1. Since all traceroute probes arrive at the intelligent router with TTL=1, the kernel module assumes the packet is the first in a set of traceroute packets. For the first packet, no deception is returned to the adver-

```
Data: socket_buffer skb
Result: Transmits a deceptive packet of type Time Exceeded or Port Unreachable
ip_header = get_ip_header(skb);
if ip_header->protocol == ICMP AND ip_header->src_addr == router_ip AND preserve_ttl > 0 then
    icmp_header = get_icmp_header(ip_header);
    if icmp_header->type == ICMP_TIME_EXCEEDED then
        ip_header->src_addr = hop_addr[preserve_ttl-1];
    else if icmp_header->type == ICMP_PORT_UNREACHABLE then
        quoted_ip_header = get_quoted_ip_header(ip_header);
        quoted_ip_header->dest_addr = webserver_ip;
        quoted_ip_header->checksum = new_ip_checksum(quoted_ip_header, quoted_ip_header->length);
    end
    ip_header->ttl = DEFAULT_TTL - preserve_ttl;
    ip_header->checksum == new_ip_checksum(ip_header, ip_header->length);
    delay_sending_packet(DELAY * preserve_ttl);
end
preserve_ttl = 0;
return ACCEPT_PACKET;
```

Figure 3.6: This function provides a deceptive Time Exceeded or Port Unreachable message to the adversary.

sary. The intelligent router truthfully replies to the adversary with an ICMP Time Exceeded message [5]. The kernel module has entered into its deception logic but because the first IP address assigned to hop_addr is that of the router, no deception is actually made. This design decision was made based on the fact that the deception is deployed at the border router of an AS. Deceiving anyone of its presence on the Internet would have potentially negative effects on the AS and expose the deception.

When the second traceroute packet arrives (using UDP and a destination port range of 33434 to 33534) it has a TTL of two. The reply to this packet is the first to be deceptively returned to the adversary. From the prerouting hook, the packet's TTL is saved in the temporary variable, preserve_ttl, to be accessed later in the postrouting hook. The packet's TTL is then set to zero. Doing so causes the packet to be processed by the host computer in the "local processes" bubble shown in Figure 3.7. In effect, the intelligent router is tricked into processing the packet as if it expired while on its way to the destination. The host generates another ICMP Time Exceeded message and prepares the reply to be sent to the adversary.

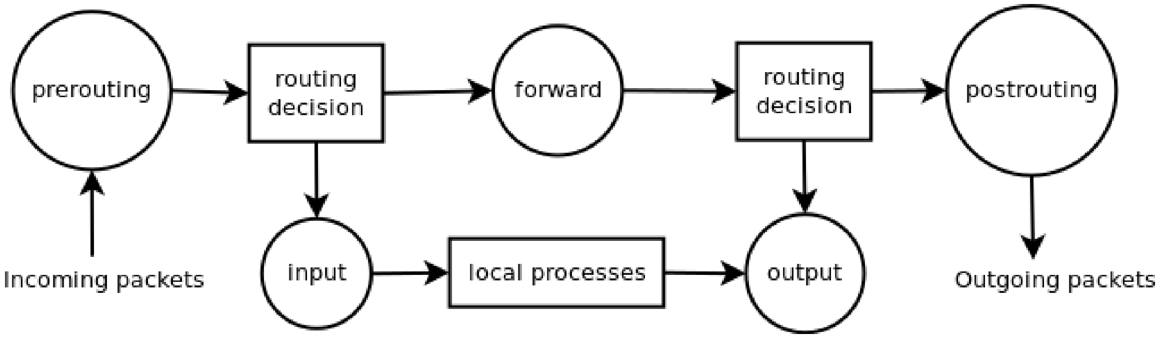


Figure 3.7: Linux packet routing scheme. The circles provide representation of the available Netfilter hooks. After [16].

When the Time Exceeded message arrives at the postrouting hook it has been prepared for delivery to the adversary but a few more finishing touches are applied to it to complete the deception. At this point the source IP address of the Time Exceeded message is that of the intelligent router (192.168.0.2). The source address is thus replaced with the source address of the first deceptive hop as described in the configuration file. In our experiment that IP address is 192.168.4.2.

Next, the TTL value of the new packet is decreased to reflect the metric the adversary expects for a hop that is more distant than the intelligent router. This is where the temporary variable, preserve_ttl, and the DEFAULT_TTL value is used. Recall that the first packet sent in response to the traceroute scan was authentic - no deception was employed. Thus it left the intelligent router with a default TTL of 64. With this second packet, the adversary expects to reach a machine that is exactly one node more distant than the intelligent router. Thus the TTL of the reply packet should be one number less than the previous packet. We can accomplish this by using the inbound packet's TTL that was preserved in the prerouting hook. The TTL of the second outbound packet is set to DEFAULT_TTL - preserve_ttl. Following this step, preserve_ttl is set to zero to prepare it for the next packet.

Finally, the packet is delayed by a multiple of the original packet's TTL value to account for the additional time it should take for the packet to reach a hop that, again, is further away from the adversary than the intelligent router. The total delay imposed on each packet was 2 milliseconds (ms) multiplied by the inbound packet's TTL. A delay of 2 ms was selected based on observations of the average delay observed between hops while withholding the deception. A better approach than applying a multiple of a specific value may be to impose a delay based

```
traceroute to 192.168.5.2 (192.168.5.2), 30 hops max, 60 byte packets
 1  192.168.9.1  1.039 ms
 2  132.65.218.87  3.996 ms
 3  240.184.140.169  3.935 ms
 4  247.10.122.16  4.178 ms
 5  153.55.189.76  3.956 ms
 6  255.253.22.13  4.126 ms
 7  112.52.193.63  3.942 ms
 8  *
 9  213.218.8.151  2.829 ms
10  48.15.140.52  3.905 ms
11  251.113.95.239  3.955 ms
12  55.254.68.156  3.961 ms
13  254.174.72.103  3.959 ms
14  *
15  126.95.77.207  3.408 ms
16  39.9.224.214  3.939 ms
17  149.247.6.143  4.249 ms
18  172.210.140.129  4.652 ms
19  30.225.206.152  3.979 ms
20  *
21  32.50.230.226  3.939 ms
22  173.195.102.154  3.896 ms
23  68.243.79.20  3.981 ms
24  181.126.253.88  3.976 ms
25  153.185.116.129  4.025 ms
26  *
27  88.35.248.68  3.182 ms
28  176.206.216.208  3.930 ms
29  67.104.227.82  3.990 ms
30  133.124.42.86  4.060 ms
```

Figure 3.8: Traceroute results with a “noisy” deception deployed.

on a probability distribution. Experiments in this area are left to future research. The process described repeats for each subsequent packet for which the deception requires a Time Exceeded message in reply.

In a typical traceroute probe, when a traceroute packet reaches its destination, the destination computer replies with an ICMP Port Unreachable message. A slightly different approach, as that described for a Time Exceeded message, is needed when sending the deceptive ICMP Port Unreachable message. In the prerouting hook, when the incoming TTL value matches the maximum number of hops (the value stored in arr_argc) required by the deception, the destination address of the packet is replaced by that of the host machine. In “local processes” the intelligent router creates the Port Unreachable message it would create had the incoming traceroute packet been destined for it. It then sends the reply packet to the postrouting hook. In the postrouting hook, the same TTL manipulation is imposed but there is one more thing that must be manipulated. An ICMP Port Unreachable message contains as its data the message that prompted the Port Unreachable message to be sent [45]. In this case, that seemingly original message contains the intelligent router's IP address in the destination field. If the web server's IP address is not replaced in this field the deception is exposed. Thus, the destination IP address is replaced by the web server's IP address and the reply is ready to be sent following the necessary delay.

3.7 Noise

An early milestone in the development of our technique is the demonstration of the ability to generate random hop information in response to a traceroute probe. Reaching this milestone demonstrates the overall feasibility of the research. It also illustrates similarities between the research herein and the radar jamming capability of an EW suite or the entrapment capability of LaBrea.

We tested our ability to generate noise in which we return random IP addresses to an adversary's traceroute probe. The results of such a deception are provided in Figure 3.8. The deception returns Time Exceeded messages containing randomly generated source IP addresses thus the adversary never receives the Port Unreachable message it expects and continues to probe until it reaches traceroute's default probe limit of 30 hops before quitting. Through experimentation we observed that our implementation periodically fails to respond to incoming traceroute packets resulting in missed hops being reported by traceroute. The missed hops are denoted by the asterisks shown in the output listing in Figure 3.8. We leave resolution of the bug to future work.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
FINDINGS

This chapter discusses experimental results based on our candidate network topology and the kernel module that implements the deception.

4.1 Experiment Overview

Once a suitable deceptive network topology was selected it was fed as input to our Python script that found the node with the highest C_B and suggested the addition of a single new link to lower the C_B. The output of this analysis is in the form of a kernel module configuration file and describes the hops necessary, including the new link, to reach the target web server. The new route described is one that routes traffic around the node with the highest C_B.

With the kernel module running, the adversary commences a traceroute scan against the IP address of the web server. The intelligent router, detecting the incoming scan, returns the prescribed deceptive route to the adversary.

In order to assure that the deception is not impeding any form of traffic, the browser on the adversary's machine is used to view the web pages provided by the web server. Correct operation of the deception was further determined by attempting to traceroute to devices other than the web server. In such cases, the deceptive route was not returned to the adversary.

4.2 Most Vulnerable Node

Figure 4.1 repeats our selected network topology, now with labels on the routers for quick reference. In terms of betweenness centrality, the most vulnerable node is R3 with C_B=0.714. By running our algorithm to minimize the maximum centrality (in Figure 3.4), we find that adding a link between the intelligent router and R5 yields a new betweenness value for R3 of C_B=0.381. Implementing this deception used the kernel configuration shown in Figure 3.3 and repeated here:

```
options deceptiveroute hop_addr=192.168.0.2,192.168.4.2,192.168.5.2
```

Comparing the configuration file to Figure 4.1, we note that 192.168.0.2 corresponds to the intelligent router, 192.168.4.2 to R5, and 192.168.5.2 to the web server. Figure 4.1 illustrates

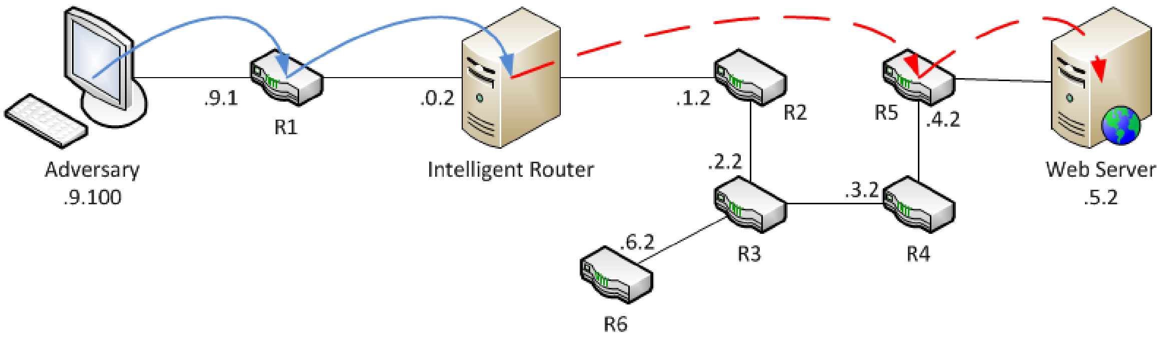


Figure 4.1: The deceptive route implemented. The solid arcs represent the truthful route provided to the adversary in response to a traceroute probe. The dashed arcs represent the deceptive route supplied by the intelligent router.

the deceptive route described by the configuration file. The solid arcs represent the truthful route provided to the adversary in response to a traceroute probe. Once the probe reaches the intelligent router the adversary is then deceived of the route by the deception implementation running on the intelligent router. The dashed arcs represent the deceptive route supplied to the adversary by the intelligent router.

4.3 Physical Route

We assessed the quality of our deception using the standard Linux UDP-based traceroute:

```
# traceroute -q1 -n -N1 192.168.5.2
```

As discussed in §2.1.4, traceroute by default sends three probe packets per hop. The parameter “-q1” sets the number of probe packets per hop to one. We used this parameter only as a convenience during debugging; it is not fundamental to deceiving the adversary. We also tried values of 1, 2 and 3 during testing of the implementation with no obvious bugs found. The parameter “-n” prevents traceroute from using the DNS to attempt to map IP addresses to host names. As with the “-q1” parameter, the “-n” parameter was used as a convenience and is not fundamental to deceiving the adversary. The parameter “-N1” instructs traceroute to have only one group of packets with the same TTL in flight at any time. The parameter “-N” should not be confused with “-q”. Where “-q” sends *x* number of packets with the same TTL, “-N” controls how many groups of packets with varying TTL traceroute may have in flight at any time. The default operation for traceroute is to simultaneously send multiple packets with increasing TTL. Doing so reduces the time taken to complete the probe by allowing traceroute to identify several nodes at once. In testing we found our implementation to work without the "-N*x*" parameter, and we successfully tested for values of *x* <= 5.

Figure 4.2 shows the output of the traceroute command when the deception implementation is withheld. For each hop, the IP address of the forward-facing interface of each node is shown in the traceroute output.

4.4 Deceptive Route

Figure 4.3 shows the output of the traceroute command when the deception implementation is deployed. From Figure 4.3 we can see that the deception implementation reported to the traceroute command the results we expected given the configuration file in Figure 3.3. Because R1 is in front of the intelligent router, it truthfully reports its presence to the traceroute command executed by the adversary. The intelligent router also truthfully reports its presence to the adversary but all subsequent packets sent elicit deceptive replies generated by the intelligent router in response. In particular, the adversary's traceroute probe receives the Port Unreachable message it expects for 192.168.5.2 thus the deception is believable.

4.5 Network Performance Impact

We performed a basic evaluation of the impact on network performance the deception imposed. To perform the evaluation we used Iperf [46], a tool for measuring maximum UDP and TCP bandwidth between two points, a client and a server. In our evaluation, the adversary's computer served as the Iperf client and the web server acted as the Iperf server. We ran 20 test iterations. For each protocol (UDP and TCP) we conducted five tests with the deception withheld and five

```
traceroute to 192.168.5.2 (192.168.5.2), 30 hops max, 60 byte packets
 1  192.168.9.1  1.280 ms (R1)
 2  192.168.0.2  3.966 ms (Intelligent Router)
 3  192.168.1.2  5.997 ms (R2)
 4  192.168.2.2  10.097 ms (R3)
 5  192.168.3.2  12.135 ms (R4)
 6  192.168.4.2  14.330 ms (R5)
 7  192.168.5.2  16.109 ms (Web Server)
```

Figure 4.2: Traceroute results (prior to deception). Device labels from Figure 4.1 are added for ease of reference.

```
traceroute to 192.168.5.2 (192.168.5.2), 30 hops max, 60 byte packets
 1  192.168.9.1  2.478 ms (R1)
 2  192.168.0.2  15.078 ms (Intelligent Router)
 3  192.168.4.2  22.520 ms (R5)
 4  192.168.5.2  32.739 ms (Web Server)
```

Figure 4.3: Traceroute results (after deception). Device labels from Figure 4.1 are added for ease of reference.

more tests with the deception deployed.

The UDP tests were configured to run on port 33434 (the first port of the standard traceroute range). Each test consisted of sending datagrams of 1470 bytes in size (the Iperf default) in 10 seconds. Bandwidth was unaffected by the deployment of the deception, showing a transfer of 1.25 Megabytes (MB) and 893 datagrams in each 10-second test, for a transfer rate of 1.05 Megabits per second (Mbps) regardless of whether the deception was deployed. Of note, for all 10 tests the results were identical. The performance measured is significantly lower than expected (100 Mbps) and may be due to the overhead imposed by executing the experiment in a virtual machine (VM), however more testing in this area is left to future work.

For the TCP test we again used port 33434. The Iperf default window size for TCP is 21 Kilobytes (KB). Unlike the UDP results, the TCP results were not identical. A minor variance existed even amongst the results while withholding the deception. While withholding the deception we observed the average bandwidth to be 5.64 Mbps. With the deception deployed we observed the bandwidth to be 5.65 Mbps. We don’t consider the marginal improvement in performance with the deception deployed to be attributable to the deception but rather a symptom of the variance in results while testing using TCP. It is our assessment that our implementation does not negatively impact bandwidth on the defended network. As with the UDP tests, future tests in the area of TCP performance are left to future work.

CHAPTER 5: CONCLUSION AND FUTURE WORK

This thesis demonstrates that a UDP-based traceroute probe may be deceived as to the actual route to a designated protected device. We implemented a kernel module for a router running the Linux OS. The kernel module inspects traffic at an ingress point of an AS and is able to identify an inbound traceroute probe and reply to the prober with a route of the deceiver's choosing. The deceptive route is specified by use of a kernel configuration file that is read by the kernel module at run-time. Having tested our implementation, we now discuss alternative approaches as well as aspects of this research which are left to future work.

5.1 Alternative Approaches

Before we began our experiments we considered how best to implement the planned deception. The following is a discussion of some of the alternative implementations considered. Each approach has its own advantages and disadvantages. The practicality of implementing each approach is left to future work.

5.1.1 An Overlay of Virtual Machines

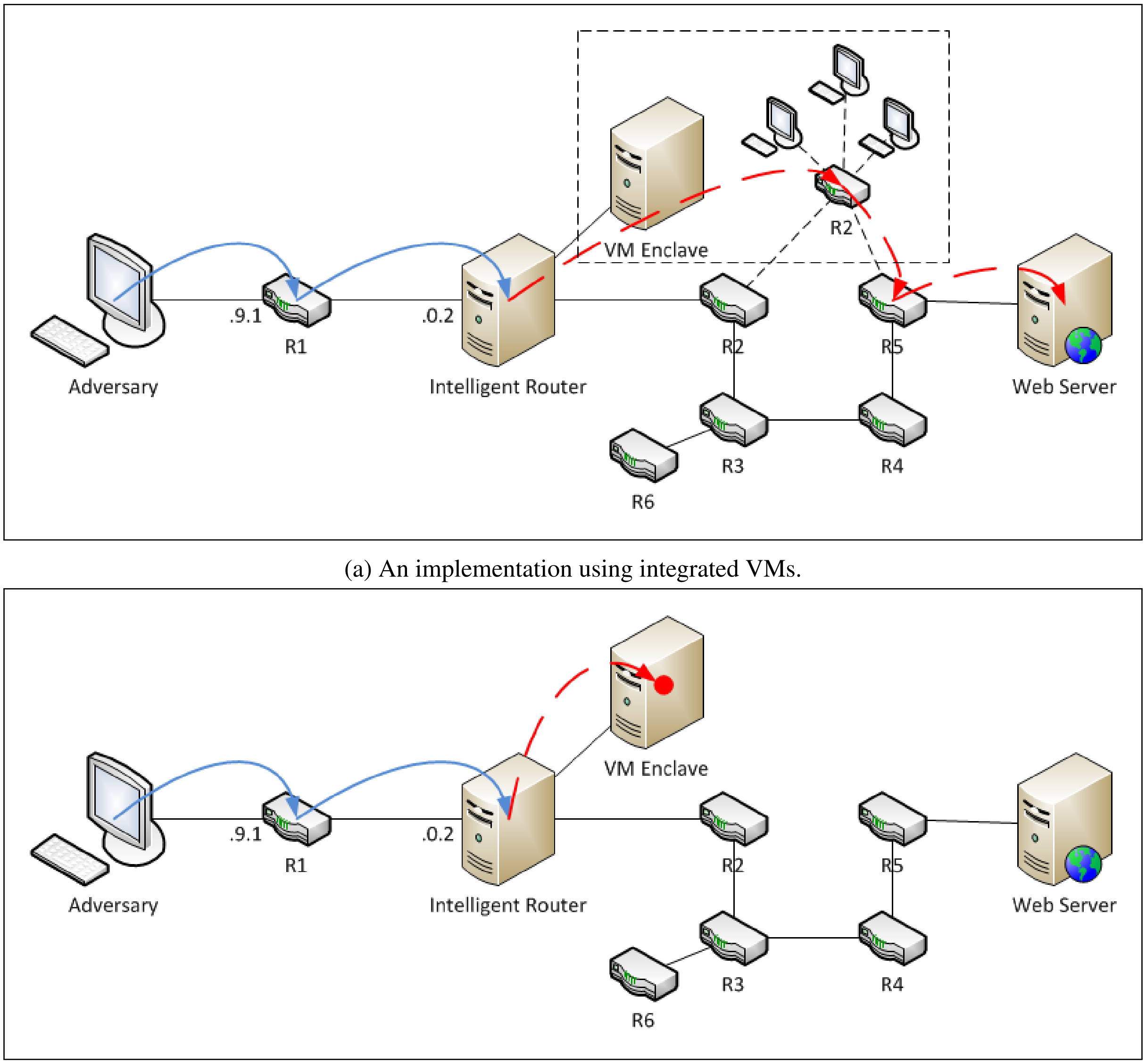
Early in this research we considered employing deception using a physical network overlaid with VMs. The VMs then act as the deceptive nodes in a route. The routing in such an implementation would require the support of a VM hypervisor. Virtualization software such as Oracle's VirtualBox [47] allows VMs to seamlessly integrate with host networks. It would not be necessary to present a deceptive route to an adversary as one would find the VMs through typical network surveillance. Furthermore, no intentional influence of router tables or the performance of any other manipulation on individual devices is fundamental to the implementation of the deception. The mere existence and ordinary behavior of the VMs is sufficient to implement the deception. Instead, the challenge in this approach would be to prevent the adversary from discovering that the VMs are, in fact, virtual. Refer to Figure 5.1a for an illustration of an overlay of VMs.

We did not pursue the VM implementation because we deemed it to be too resource-intensive. A network of VMs overlaying a physical network would necessitate a greater investment in hardware. Not only would hardware be required for each physical node in the network but an additional and substantial investment in hardware would be needed to host the required VMs. A VM would be required for every node in the deception scheme. Adding to the challenge are the issues of fault tolerance and management complexity. Not only would the VMs require maintenance but so too would the hardware that serves the VMs. There may also be a network performance penalty by integrating VMs into a real network. Recall the relatively low performance we reported in §4.5. Comparatively, the approach used herein requires minimal additional hardware investment or maintenance of VMs and does not impact network performance. Whereas an implementation using VMs requires a VM for every deceptive node, our implementation does not require such a one-to-one mapping. To meet the minimal additional hardware requirement of our implementation, existing border routers may need to be replaced by other devices, such as servers, that may act in place of dedicated routers while running the software that implements the deception.

The use of VMs has its merits. The advantage to using VMs is that for every deceptive node, one has at their disposal all tools necessary to mount a plausible deception. For example, if a deception required convincing an adversary of the existence of a web server at a deceptive node, our implementation would require writing code to accurately implement the facade of a web server whereas on a VM one could simply install web server software. Another benefit of employing virtual machines is that it would negate the necessity of having specially coded software running on an intelligent router. In our implementation, the specially coded software is the kernel module developed specifically for the purpose of deceiving an adversary. Along with the implementation come all the software life-cycle issues such as maintenance, evolutionary updates, bug fixes and configuration changes. In our case, the responsibility for such maintenance falls on the organization employing the deception.

5.1.2 Shunting

Another approach we considered was to develop an intelligent router capable of identifying attempts to probe the network topology, but rather than have the intelligent router reply with a deceptive route, it instead shunts the incoming network traffic into an enclave of VMs. See Figure 5.1b for an illustration of shunting. The difference in the shunting approach and the approach described in §5.1.1 is that the enclave of VMs is entirely self contained and independent of the defended network rather than being dispersed over and integrated with the defended network. By shunting probing traffic into the enclave, we prevent it from causing congestion on the defended network (as is the case with the implementation presented herein) at the expense of having to create a duplicate of the defended network, plus the desired deception scheme, in



(a) An implementation using integrated VMs.

(b) An implementation using shunting.

Figure 5.1: Alternative implementations of deception.

the enclave.

5.1.3 Implementation in User Space

We chose to implement the deception using a kernel module instead of an application in user space. The reason for this decision was the assumed performance improvement gained by running in kernel space rather than user space. By implementing the deception in kernel space we avoided having to pass packets to user space, performing the packet manipulation, then passing the packets back into kernel space before transmitting the results. The trade-off in this decision is that the standard C library is not available in kernel space. Thus, unusual solutions needed to be coded for otherwise familiar problems that are more easily solved in user space (such as string concatenation and random number generation for IP addresses). It may be likely that any performance gained by coding in kernel space is likely negated by the induced packet delay necessary to present a plausible deception.

5.2 Future Work

The technique and implementation presented in this research provides a potentially important first step in the use of topological deception, but there is much room for additional development.

5.2.1 Internet Protocol version 6

The research presented herein involved the use of Internet Protocol version 4 (IPv4) addresses exclusively. As the IPv4 address space reaches exhaustion, more organizations will begin using Internet Protocol version 6 (IPv6) [48]. Thus, for the deception implementation presented here to remain relevant, it should be modified to enable support for IPv6 addresses.

5.2.2 Kernel Module vs. Virtual Machines

As discussed in §5.1.1 and §5.1.2, different approaches would employ the use of VMs. It is left to future work to perform experiments to determine the benefits of each approach and to determine in which cases one is more desirable than the others. The following is an example of relevant research questions. Can VMs be configured to hide the attributes that reveal that the machines are virtual? In other words, can a VM be configured to appear as a real machine so as to thwart fingerprinting attempts? Does the integration of VMs with a host network adversely affect network performance compared to shunting? Can a network be accurately simulated using virtualization? Can the deception be implemented on a production network to assess which parts of the network are attacked? These research questions may be tested using Iperf [46], nmap [11] and other performance measuring tools. Future researchers may also consider enlisting the help of human participants in surveys to identify whether a remote machine is real or virtual.

5.2.3 Kernel Module vs. User Space

As discussed in §5.1.3 a different approach would involve coding the implementation in user space rather than in kernel space. It is likely that a more complete implementation of the deception may be achieved by coding in user space. The kernel module employed herein is suitable for the simple deception implemented, however, as more features are added and the potential for introduction of bugs increases, the greater the risk of causing the intelligent router to crash. Should an implementation in user space crash, it would not cause the router to go offline. Furthermore, a programmer, using the familiar C library, may be able to write an implementation that more generally identifies a traceroute probe that is aimed at any device in the defended network and is using ICMP or TCP rather than UDP. Such a general and more flexible implementation is harder to achieve in kernel space given the limited library of C functions available compared to user space.

5.2.4 Creating Robust Deceptive Nodes

Rather than employing VMs to gain a robust tool set, our implementation may be further developed to emulate certain services (such as FTP, SSH, etc.) that may exist at end-point devices such as servers. While there are normally no user accessible services on routers other than those afforded by ICMP, the provision of a deception that employs deceptive services is applicable mainly to end-points. Again, the example of a web server is appropriate. If we wish to convince an adversary that a web server exists at a deceptive node, a facade of a web server must be implemented. If an adversary is presented with a suspect node, he may attempt to access the deceptive web server or other services on that node to determine its authenticity. Using our approach, the deception is exposed if an adversary attempts to access any service on a deceptive node. A more robust approach would be to have the deception emulate certain services on a given deceptive node. By doing so, the adversary is stalled by a deception methodology similar to the one implemented by LaBrea [22] as discussed in §2.2.5.

5.2.5 Packet Delay

Our implementation uses a deterministic value to cause delay between replies to a traceroute probe. This is not an ideal approach for two reasons. First, end-to-end delays in a network are not always consistent due to the variable effects of queuing and processing along a path. Second, links further away from a prober are not always slower to respond than a closer link. A more suitable approach to imposing accurate delay may be to base the delay on measurements taken from models of the deceptive network the intelligent router is presenting to the adversary.

King [49] is a tool that uses Domain Name System (DNS) name servers to estimate host-to-host latency. Gummadi *et al.* [50] describe the technique used by King as follows, “First, given a pair of end hosts to be measured, in most cases it is possible for King to find DNS name servers that are topologically close to the end hosts. Second, given a pair of DNS name servers, King can accurately estimate the latency between them using recursive DNS queries.” With an accurate delay model obtained from King, one could then estimate the appropriate delays necessary for the deceptive links used in the deception scheme. The approach employed by King may be used in an organization sufficiently large enough to need multiple and topologically distributed DNS name servers. However, in our experiments, the network was kept small enough to demonstrate the proof-of-concept of topological deception thus DNS name servers were unnecessary.

Zhang *et al.* [51] show that delay can be estimated by placing a model of the network in a plane. The Euclidean distance between each pair of connected nodes is then used as the delay. While this simple approach is viable, it precludes the use of any kind of scheme for introducing variance into the delays between nodes.

Lastly, the delays imposed in the deception may be taken from direct measurement of the delays that exist between each node of the defended network. For example, in the topology shown in Figure 4.1, the deceptive route resembles the actual route in that the last hop before reaching the web server is R5. With the deception withheld, we could run traceroute many times, perhaps 100 or 1,000 times, and record the observed minimum and maximum packet delay between R5 and the web server. We could then use those values as the upper and lower bounds on the range of numbers from which to randomly select a deceptive delay. Using this method would require changes in the kernel configuration file to support defining upper and lower bounds for packet delays on a per-hop basis. Unfortunately, direct observation of packet delays is not possible for deception schemes that omit real nodes or introduce new deceptive nodes. In such cases, one of the methods described above may be used.

5.2.6 ICMP and TCP Probes

The implementation presented herein deceives traceroute probes sent using UDP, the default method on Linux systems. But traceroute is capable of using ICMP and TCP packets, circumcenting firewall schemes that block certain ports (such as 33434 to 33534) or protocols (such as UDP), to determine the path to a destination. The deception implementation presented herein does not work for ICMP and TCP unless the source code for our implementation is modified. Our implementation uses “if” statements to identify incoming packets as UDP packets originating from a traceroute probe. Three conditions must be satisfied to identify a packet as a traceroute probe. The packet must be: (1) UDP, (2) within the port range traceroute uses for UDP (33434 to 33534), and (3) destined for a protected device (the web server in our experiments). Once a packet is identified as being part of a traceroute probe, its TTL is inspected. The first packet in a traceroute probe will always arrive at the intelligent router with TTL=1 thus the

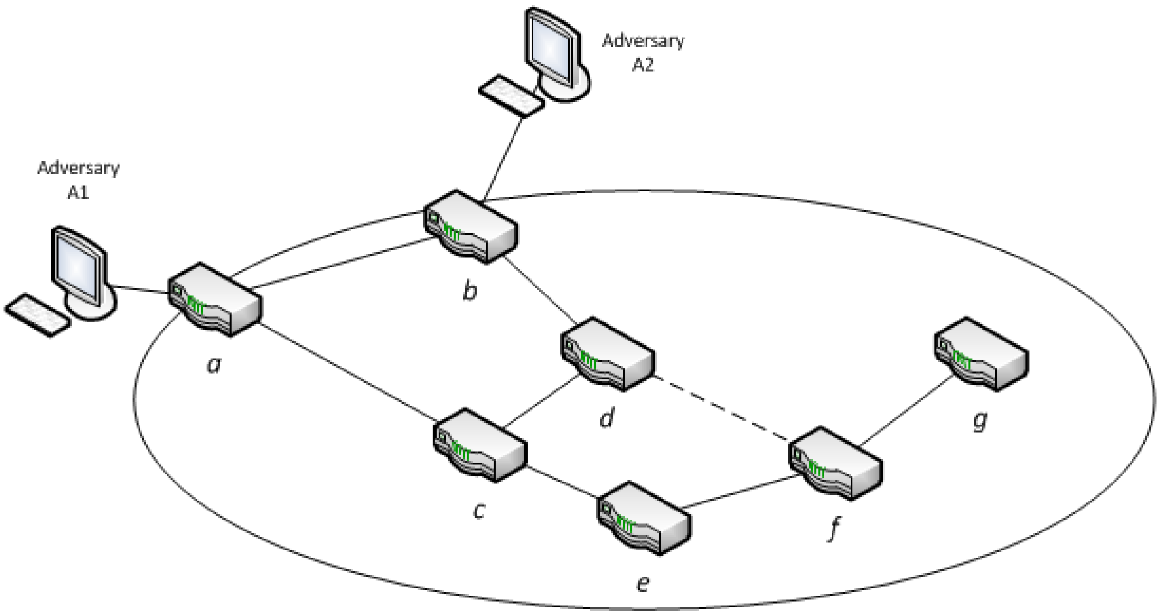


Figure 5.2: Topological deceptions must be consistent across multiple ingress points.

deception begins. Using the three conditions for identifying a UDP packet as a framework, it is trivial to edit the conditional statements to identify TCP packets arriving at port 80 and destined for the protected device. The subsequent confirmation of the packet having TTL=1 assures the intelligent router that a TCP-based traceroute probe is incoming and that it is time to deploy the deception. A similar process may be applied to account for traceroute probes using ICMP.

5.2.7 Multiple Ingress Points

The implementation presented herein is deployed in a single border router for an AS. Organizations may have more than one border router. Each border router must be able to present a topological deception that is consistent with but not identical to the deceptive topologies presented by the other border routers. Figure 5.2 illustrates the problem with routers labeled *a* to *g*. Recall that traceroute probes elicit interfaces, not labels as shown here. The labels in Figure 5.2 are provided for expediency. Assume adversaries A1 and A2 are probing for node *g* and the topological deception calls for a virtual link between nodes *d* and *f*. The deception should be implemented such that adversary A1 sees a deceptive route presented by *a* that is described as {*a*,*b*,*d*,*f*,*g*}. Adversary A2 should see a deceptive route presented by *b* that is described as {*b*,*d*,*f*,*g*}. The two routes are different but do not contradict each other. Rather, they reinforce the deception of a link existing between *d* and *f*.

Our vulnerability analysis determines the route around a vulnerable node based on the knowledge of only a single ingress point to the network. A more robust approach would be to determine a route around the most vulnerable node from any ingress point into the network and generate tailored kernel configuration files for each border router. Such is not achieved herein due to the existence of just a single ingress point in the model used for our experiments.

5.2.8 Varying Design Dependencies

The scope for our implementation is narrow in that only a single protected device (the web server) is defined. A traceroute probe from outside the defended network to the protected device, and only the protected device, met with the result of the intelligent router presenting the topological deception. An improved implementation may consider delivering a believable deception regardless of the destination being probed. Such an implementation would likely determine the best deceptive route to present at the time the traceroute probe is received, rather than replying with a pre-defined route prescribed by a configuration file.

Another design dependency worth evaluating is the use of deception to protect certain destination prefixes. Compared to the approach described above which attempts to account for every device in the network, it may be more appropriate to protect only those devices with a certain destination prefix. For example, an organization may wish to implement deception to protect servers but not workstations or other devices.

Our implementation inspects all inbound traffic to identify a traceroute probe however it does not inspect the source IP address. The consequence of this is that the topological deception is presented regardless of who originated the probe. There may be some value in deploying or withholding the deception depending on where the probe originates. A set of known good IP addresses may be spared the deception while a set of known bad IP addresses may be presented with the deception. This would be especially valuable for system administrators working remotely who may wish to inspect the network to diagnose are resolve network problems.

5.2.9 Evaluation of Success

This is perhaps the most necessary element left to future work. A thorough evaluation of the implementation's ability to deceive a human user offers the best indication that the methodology and implementation presented here is viable and successful. In a laboratory environment, human volunteers may be presented with a view into multiple networks and asked to determine, using traceroute and other tools, whether the network exhibits any unusual behavior or attributes.

The success of our approach may also be measured in the real world similar to the arrangement employed by Frederick. [33]. By placing the implementation on a network outside an AS firewall and observing probes against it, researchers may see a decrease in port scanning against vulnerable nodes in the physical network and an increase in scanning on nodes in the deceptive network.

5.3 Adversary's Assessment

Counter to evaluating the defender's success, one may evaluate the adversary's ability to infer that topological deception is being employed. We have noted herein the potential of issues such as packet delay, network bandwidth and fingerprinting to reveal that deception is being used. From the adversary's perspective, how valuable are these points? Can an analysis of the delays between replies in a traceroute probe enable the adversary to deduce the use of deception? Do bandwidth limitations in portions of a defended network indicate that deception is being used? To what degree must an adversary fingerprint a deceptive node in order to determine the actual nature of the node? Viewing the problem space from the adversary's perspective is potentially a more challenging problem as an accurate assessment would require operating much as the adversary would by using a set of intelligence-collecting tools and by operating outside the defended network.

5.4 Concluding Remark

As the cyber domain continues to grow in importance to military operations and the DoD in particular, the United States will need novel techniques for defending its networks. Although the implementation of topological deception in this thesis represents proof-of-concept only, ongoing development of the research presented herein may well complement other DoD cyber operations and prove to be a useful application of MILDEC in cyberspace.

THIS PAGE INTENTIONALLY LEFT BLANK

REFERENCES

[1] L. Panetta, “Remarks by Secretary Panetta on Cybersecurity to the Business Executives for National Security,” [Transcript], Intrepid Sea, Air and Space Museum, New York, NY, October 2012. [Online]. Available: http://www.defense.gov/transcripts/transcript.aspx?transcriptid=5136.

[2] U.S. Department of Defense. (2011, July) Department of Defense Strategy for Operating in Cyberspace. [Online]. Available: http://www.defense.gov/news/d20110714cyber.pdf.

[3] Chairman of the Joint Chiefs of Staff. (2006, July) Joint Publication 3-13.4: Military Deception. [Online]. Available: http://www.c4i.org/jp3_13.pdf.

[4] *User Datagram Protocol*, Internet Engineering Task Force Std. 768, 1980. [Online]. Available: http://www.ietf.org/rfc/rfc768.txt.

[5] *Internet Control Message Protocol*, Internet Engineering Task Force Std. 792, 1981. [Online]. Available: http://www.ietf.org/rfc/rfc792.txt.

[6] M. J. Muuss. (2013) The Story of the PING Program. [Online]. Available: http://ftp.arl.army.mil/mike/ping.html.

[7] M. Shema and B. C. Johnson, *Anti-Hacker Toolkit*. Emeryville, CA: McGraw-Hill/Osborne, 2004.

[8] R. Schemers. (2013) fping. [Online]. Available: http://fping.sourceforge.net/.

[9] S. Sanfilippo. (2013) hping. [Online]. Available: http://www.hping.org/.

[10] R. Shirey, “Internet Security Glossary, Version 2,” RFC 4949 (Informational), Internet Engineering Task Force, Aug. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4949.txt.

[11] G. F. Lyon. (2013) Nmap. [Online]. Available: http://nmap.org/.

[12] W. Willinger, D. Alderson, and J. C. Doyle, “Mathematics and the Internet: A Source of Enormous Confusion and Great Potential,” *Notices of the AMS*, vol. 56, no. 5, May 2009. [Online]. Available: http://www.ams.org/notices/200905/rtx090500586p.pdf.

[13] R. Oppliger, “Internet security: firewalls and beyond,” *Communications of the ACM*, vol. 40, no. 5, pp. 92–102, May 1997. [Online]. Available: http://doi.acm.org/10.1145/253769.253802.

[14] Check Point Software Technologies LTD. (2013) ZoneAlarm. [Online]. Available: http://www.zonealarm.com/.

[15] Comodo Group, Inc. (2013) Comodo Firewall. [Online]. Available: http://www.comodo.com/.

[16] The Netfilter.org Project. (2013) iptables. [Online]. Available: http://www.netfilter.org/.

[17] J. Hawkinson and T. Bates, “Guidelines for creation, selection, and registration of an Autonomous System (AS),” RFC 1930 (Best Current Practice), Internet Engineering Task Force, Mar. 1996. [Online]. Available: http://www.ietf.org/rfc/rfc1930.txt.

[18] N. Provos. (2013) HoneyD. [Online]. Available: http://www.honeyd.org/.

[19] MITRE. (2013) HoneyClient. [Online]. Available: http://www.honeyclient.org/.

[20] The Honeynet Project. (2013) The Honeynet Project. [Online]. Available: http://www.honeynet.org/.

[21] Microsoft. (2013) HoneyMonkey. [Online]. Available: http://research.microsoft.com/en-us/um/redmond/projects/strider/.

[22] T. Liston. (2013) LaBrea. [Online]. Available: http://labrea.sourceforge.net/.

[23] C. Trowbridge, “An overview of remote operating system fingerprinting,” SANS Institute, Tech. Rep., July 2003. [Online]. Available: http://www.sans.org/reading_room/whitepapers/testing/overview-remote-operating-system-fingerprinting_1231.

[24] L. Spitzner, *Honeypots: Tracking Hackers*. Boston, MA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[25] C. A. Fowler and R. F. Nesbit, “Tactical deception in air-land warfare,” *Journal of Electronic Defense*, vol. 18, no. 6, pp. 37–40, 1995. [Online]. Available: http://www.highbeam.com/doc/1G1-17620824.html.

[26] B. Whaley, “Toward a general theory of deception,” *Journal of Strategic Studies*, vol. 5, no. 1, pp. 178–192, 1982. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/01402398208437106.

[27] R. Beverly, “A robust classifier for passive TCP/IP fingerprinting,” in *Passive and Active Network Measurement*, ser. Lecture Notes in Computer Science, C. Barakat and I. Pratt, Eds. Springer Berlin Heidelberg, 2004, vol. 3015, pp. 158–167. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-24668-8_16.

[28] M. Smart, G. R. Malan, and F. Jahanian, “Defeating TCP/IP stack fingerprinting,” in *Proceedings of the 9th conference on USENIX Security Symposium - Volume 9*, ser. SSYM’00. Berkeley, CA: USENIX Association, 2000, pp. 17–17. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251306.1251323.

[29] N. C. Rowe and H. S. Rothstein, “Two taxonomies of deception for attacks on information systems,” *Journal of Information Warfare*, pp. 27–39, July 2004. [Online]. Available: http://faculty.nps.edu/ncrowe/mildec.htm.

[30] J. F. Dunnigan and A. A. Nofi, *Victory and deceit : dirty tricks at war*, 1st ed. W. Morrow, New York, NY, 1995.

[31] K. E. Huber, “Host-based systemic network obfuscation system for Windows,” M.S. thesis, Air Force Technical Institute, Wright-Patterson Air Force Base, OH, 2011.

[32] Gemini Security Solutions, Inc. (2013) OSfuscate. [Online]. Available: http://tinyurl.com/3eh869.

[33] E. E. Frederick, “Testing a low-interaction honeypot against live cyber attackers,” M.S. thesis, Naval Postgraduate School, Monterey, CA, 2011.

[34] F. Zhang, S. Zhou, Z. Qin, and J. Liu, “Honeypot: a supplemented active defense system for network security,” in *Parallel and Distributed Computing, Applications and Technologies, 2003. PDCAT’2003. Proceedings of the Fourth International Conference on*, aug. 2003, pp. 231–235.

[35] D. J. Watts and S. H. Strogatz, “Collective dynamics of ’small-world’ networks,” *Nature*, vol. 393, no. 6684, pp. 440–442, Jun. 1998. [Online]. Available: http://dx.doi.org/10.1038/30918.

[36] J. Grossman, B. Marsili, C. Goudjil, and A. Eromenko. (2013) GNS3 Graphical Network Simulator. [Online]. Available: http://www.gns3.net/.

[37] The Lubuntu Project. (2013) Lubuntu. [Online]. Available: http://lubuntu.net/.

[38] G. Combs. (2013) Wireshark. [Online]. Available: http://www.wireshark.org/.

[39] C. Lincoln. (2013) SliTaz. [Online]. Available: http://www.slitaz.org/.

[40] P. Jakma, V. Jardin, D. Lamparter, A. Schorr, D. Tejblum, and G. Troxel. (2013) Quagga Routing Suite. [Online]. Available: http://www.nongnu.org/quagga/.

[41] L. C. Freeman, "A Set of Measures of Centrality Based on Betweenness," *Sociometry*, vol. 40, no. 1, pp. 35–41, Mar. 1977. [Online]. Available: http://links.jstor.org/sici?sici=0038-0431\%28197703\%2940\%3A1\%3C35\%3AASOMOC\%3E2.0.CO\%3B2-H.

[42] A. Hagberg, D. Schult, and P. Swart. (2013) Networkx. [Online]. Available: http://networkx.lanl.gov/.

[43] Python Software Foundation. (2013) Python. [Online]. Available: http://www.python.org/.

[44] IANA, "IP Option Numbers," 2013. [Online]. Available: http://www.iana.org/assignments/ip-parameters/ip-parameters.xml.

[45] D. Malone and M. J. Luckie, "Analysis of ICMP Quotations," in *Passive and Active Network Measurement, 8th Internatinoal Conference, PAM 2007, Louvain-la-neuve, Belgium, April 5-6, 2007, Proceedings*, ser. Lecture Notes in Computer Science, S. Uhlig, K. Papagiannaki, and O. Bonaventure, Eds., vol. 4427. Springer, 2007, pp. 228–232.

[46] J. Dugan and M. Kutzko. (2013) Iperf. [Online]. Available: http://iperf.sourceforge.net/.

[47] Oracle. (2013) VirtualBox. [Online]. Available: http://www.virtualbox.org/.

[48] L. Colitti, S. H. Gunderson, E. Kline, and T. Refice, "Evaluating IPv6 adoption in the Internet," in *PAM 2010*, 2010. [Online]. Available: http://www.pam2010.ethz.ch/papers/full-length/15.pdf.

[49] K. P. Gummadi, S. Saroiu, and S. D. Gribble. (2013) King. [Online]. Available: http://www.mpi-sws.org/~gummadi/king/.

[50] K. P. Gummadi, S. Saroiu, and S. D. Gribble, “King: Estimating latency between arbitrary internet end hosts,” in *SIGCOMM Internet Measurement Workshop 2002*. [Online]. Available: http://www.mpi-sws.org/~gummadi/king/king.pdf.

[51] B. Zhang, T. S. E. Ng, A. Nandi, R. Riedi, P. Druschel, and G. Wang, “Measurement based analysis, modeling, and synthesis of the internet delay space,” in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, ser. IMC ’06. New York, NY, USA: ACM, 2006, pp. 85–98. [Online]. Available: http://doi.acm.org/10.1145/1177080.1177091.

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Dr. Robert Beverly
   Naval Postgraduate School
   Monterey, California

4. Dr. David Alderson
   Naval Postgraduate School
   Monterey, California

5. Dr. Cynthia Irvine
   Naval Postgraduate School
   Monterey, California

6. LT Samuel T. Trassare, USN
   Naval Postgraduate School
   Monterey, California