3-2004

Comparative Analysis of Active and Passive Mapping Techniques in an Internet-Based Local Area Network

James B. Kuntzelman

AFIT/GCS/ENG/04-09

COMPARATIVE ANALYSIS OF ACTIVE AND PASSIVE MAPPING TECHNIQUES IN AN INTERNET-BASED LOCAL AREA NETWORK

THESIS

James B. Kuntzelman
Master Sergeant, USAF

AFIT/GCS/ENG/04-09

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/04-09

COMPARATIVE ANALYSIS OF ACTIVE AND PASSIVE MAPPING TECHNIQUES IN AN INTERNET-BASED LOCAL AREA NETWORK

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

James B. Kuntzelman, B.S.C.S.

Master Sergeant, USAF

March, 2004

AFIT/GCS/ENG/04-09

COMPARATIVE ANALYSIS OF ACTIVE AND PASSIVE MAPPING TECHNIQUES IN AN INTERNET-BASED LOCAL AREA NETWORK

THESIS

James B. Kuntzelman, B.S.C.S.

Master Sergeant, USAF

Approved:

/signed/	10 Mar 2004
Dr. Richard A. Raines Thesis Advisor	Date
/signed/	10 Mar 2004
Maj. Rusty O. Baldwin, PhD Committee Member	Date
/signed/	10 Mar 2004
Dr. Gilbert L. Peterson Committee Member	Date

Acknowledgements

I would like to express my sincere appreciation to my thesis advisor, Dr. Rick Raines, for his guidance and support throughout the course of this thesis effort. Your continually asking "why?" and "so what?" were pointed ways to help me keep on track. See? I can turn it in on-time. Also, Major Rusty Baldwin, thesis committee member, the performance analysis expert and all-around good guy–the insight and experience was certainly appreciated.

To Capt Eugene Turnbaugh, thanks for the statistical spreadsheet to get me thinking in the right direction. To Capt Danny Bias, thanks for the TCP client/server software, Java style. Thanks to 2Lt Zachary Gray and 1Lt Mark Kleeman for nurse-maiding me through the LaTeX learning experience. Thanks to Donald Knuth and Leslie Lamport for creating TeX and LaTeX respectively, without which, this thesis would have been really difficult to manage. Thanks also to Capt Josh Green for his friendship throughout this ordeal. Thanks for sharing.

To my fellow GCE/GCS-04M classmates: We came, we saw, we whined a bit, and then we conquered. A special thanks to my fellow first-time enlisted students. Don't forget, we're all special! In a related thanks, I'd like to thank SMSgt Hobson for opening the door to enlisted personnel at AFIT. Timing is everything!

A very special thanks to SMSgt Stephanie Carroll, who dragged me kicking and screaming through the Dark Times (aka, STAT583.)

Finally, and obviously most importantly, I would like to thank my wife and children for putting up with me, lo, these many years.

James B. Kuntzelman

Table of Contents

Page

Page

List of Figures

List of Tables

AFIT/GCS/ENG/04-09

Abstract

Network mapping technologies allow quick and easy discovery of computer systems throughout a network. Active mapping methods, such as using *nmap*, capitalize on the standard stimulus-response of network systems to probe target systems. In doing so, they create extra traffic on the network, both for the initial probe and for the target system's response. Passive mapping methods work opportunistically, listening for network traffic as it transits the system. As such, passive methods generate minimal network traffic overhead. Active methods are still standard methods for network information gathering; passive techniques are not normally used due to the possibility of missing important information as it passes by the sensor. Configuring the network for passive network mapping also involves more network management.

This research explores the implementation of a prototype passive network mapping system, *lanmap*, designed for use within an Internet Protocol-based local area network. Network traffic is generated by a synthetic traffic generation suite using *honeyd* and *syntraf*, a custom Java program to interact with *honeyd*. *lanmap* is tested against *nmap* to compare the two techniques.

Experimental results show that *lanmap* is quite effective, discovering an average of 76.1% of all configured services (server- and client-side) whereas *nmap* only found 27.6% of all configured services. Conversely, *lanmap* discovered 19.9% of the server services while *nmap* discovered 92.7% of the configured server-side services. *lanmap* discovered 100% of all client-side service consumers while *nmap* found none. *lanmap* generated an average of 200 packets of network overhead while *nmap* generated a minimum of *minimum* 8,600 packets on average–up to 155,000 packets at its maximum average value.

The results show that given the constraints of the test bed, passive network mapping is a viable alternative to active network mapping, unless the mapper is looking for unused server-side services.

COMPARATIVE ANALYSIS OF ACTIVE AND PASSIVE MAPPING TECHNIQUES IN AN INTERNET-BASED LOCAL AREA NETWORK

1. Introduction

With the ever increasing requirements placed on network administrators due to mission needs and the reliance on networks and network-based systems, network and network security administrators cannot afford to be ignorant of the systems connected to their networks. It is paramount they maintain a comprehensive view of their areas of responsibility. One technique available to them is network mapping. Network mapping is a discovery process enumerating network devices and the services provided by those systems. With this information, a network administrator can determine if there have been significant changes to the network and take action to resolve discrepancies.

1.1 Background

There are two basic types of network mapping, active and passive. Active methods rely on a stimulus-response from network systems. The active mapper sends special probes to target systems and awaits an expected response and builds a map based on those responses (or lack thereof). A passive mapper simply listens to traffic on the network and builds a map based on the normal interaction of client and server systems in the network.

Active network mapping creates network overhead by sending probes to target systems which further elicit a response. Passive methods rely solely on the traffic already on the network and introduce little, if any, overhead onto the network.

1.2 Research Problem

This research compares these two techniques and performs a side-by-side comparison to determine which is more accurate and which has less network overhead. This research explores both techniques, using a freely available, "off-the-shelf" program representative of the active mapping technique. A passive technology is represented by a custom system that includes a network sensor and a data collector to provide the mapping to a user. Both approaches are compared in a simulated network environment and quantified results to the comparison question generated.

1.3 Scope

The scope of this research is limited to examining the results of active and passive network mapping systems. *nmap* is selected as the active mapping tool due to its overwhelming popularity and its widespread use. Since no passive mapping system is easily available that meets the needs of this research, a custom system is constructed.

Both systems are presented with identical networks on which to perform their mapping function. Results of the mapping are compared statistically using accuracy and efficiency as metrics. Accuracy metrics capture the correctness and completeness of the mapping. Efficiency metrics describe the amount of network overhead generated by the different techniques.

1.4 Approach

This investigation uses empirical results of real systems in operation. Using freely available software and standard desktop computer systems, an experimental test bed is built. This test bed provides facilities for both the active and passive mapping system to perform their task in a realistic yet controlled environment. The custom passive mapping system is developed using freely available components and libraries. A client-server network environment is constructed, complete with network

traffic for the passive mapper to examine and the active mapper to contend with. Finally, trials are run using various configurations and time frames to gather metrics used for comparison.

The network is based on *honeyd*, an open-source honeypot program, to provide virtual machines for the server-side environment and a custom traffic generator to interact with those servers.

1.5 Summary

The remainder of this document is organized into four chapters. Chapter 2 contains the literature review where background associated with the Internet Protocol (IP) and mapping methods are discussed. The methodology for the experimental phase of this investigation is given in Chapter 3. The analysis of the results and comparison between active and passive techniques follow in Chapter 4. Finally, Chapter 5 provides a summary of the thesis effort and identifies areas of the research to be explored in future research efforts.

2. Background / Literature Review

2.1 Introduction

This chapter provides an overview to network mapping techniques. Section 2.2 describes network mapping in general. Section 2.3 provides definitions of terms as they apply to this research and Section 2.4 discusses reasons for and the uses of network mapping. Section 2.5 describes active mapping techniques in common use while Section 2.6 reveals advantages and disadvantages of those techniques. Section 2.7 describes some passive methods and Section 2.8 defines drawbacks of passive methods. Finally, Section 2.9 discusses other research in these and related areas.

2.2 Description

With the massive growth of and dependence on computer networks and the Internet for e-commerce and mission-critical command-and-control messaging, those same computer networks have greatly increased in capabilities to accommodate increased network traffic. This increased need has led to higher levels of complexity as these networks and network systems have spread and grown over time. Thus, management and control of the enterprise network has also become quite complex. Network mapping techniques give network managers and network security personnel the capability to know, discover and understand their network topology, the services that their systems provide, and the location and types of consumers of those services. There are a number of mapping techniques currently being used in production environments such as Hewlett-Packard's OpenView (HPOV) Network Node Manager (NNM) [31] and Ipswitch's WhatsUp Gold [13], all of which are stimulus-response or active systems. Passive mapping methods are not being widely used in production networks. This chapter discusses the importance of network mapping, currently used and proposed methods and techniques, the success expectancy and potential limitations hindering implementation.

2.3 Definition of Network Mapping and Network Maps

To alleviate confusion, "network mapping" will be explicitly defined. According to Merriam-Webster, a network is defined as "a system of computers, terminals, and databases connected by communications lines" [9] whereas a mapping is defined as (in this context) "to make a survey of for or as if for the purpose of making a map" [9]. So, a definition of network mapping could be "a survey of a system of computers, terminals, and databases for the purpose of making a map." While this definition is close to what is needed, the definition is modified to suit the purposes of this investigation:

> *Network mapping* is the information gathering process by which information about a network is collected regarding its topology, its systems and the services those systems provide.

Other related definitions are given to provide a fuller understanding of this investigation. These definitions are:

> *Client*: a computer program, which accesses a service and typically provides a user interface for that service.
>
> *System*: any device which can load and execute programs.
>
> *Network System*: the computers, networks, clients and servers which encompass the actions, tasks, and programs to be executed.
>
> *Server*: a system which provides services to clients.
>
> *Service*: a computer program which provides a given information resource to a client (i.e., e-mail service or web service).
>
> *Accuracy*: a measure of the correctness of mapping decisions.
>
> *Efficiency*: used to describe the network overhead the mapping algorithm induces to perform its job. The fewer packets introduced into the network, the more efficient the algorithm is.

Using these definitions, a "client system" is nothing more than a system which loads a client program and uses it to access a service.

2.4 Why Network Mapping?

A network map provides simplified understanding of a complex system or network of systems. The age-old cliché of "a picture is worth a thousand words" certainly applies here. Using spreadsheets or databases to track and represent network system configurations simplifies the management of that information via automated means. This type of information may mean little to the network manager must understand the systems under his/her control. Therefore, if the information is reduced to its essence–that representative information the network manager needs to understand–and represented it in a way which is easily grasped and readily understandable, a complex system is simplified.

Useful network maps have several properties in common. They are typically pictorial in nature and provide an overview of the systems that comprise the network. However, depending on the experience or working-level of the person using it–technician or manager–the network map could be a simple list of network systems and specific, critical attributes. Probably the most important piece of information on a network map is the function of a given network device. A network map should also disclose the system's type without confusion. Most network mapping tools like the aforementioned WhatsUp Gold and Network Node Manager represent the system type with standardized pictures or icons. However, it is up to the network administrator to configure the visual portion of network maps appropriately for the type of equipment.

Another important representation is the system location within the network. The topology of enterprise networks is typically hierarchical in nature, with the primary service-providing systems on a few network segments (for redundancy and load balancing) and the client workstations spread throughout other segments within the enterprise. Other information is and will be available via the map, but these two pieces of information are those that interest both network administrators and managers. The network administrator needs to know what systems are doing on his/her

network (regarding interoperability, or the effect of new systems on the network as a whole, etc). Managers want to keep track of how many systems are online and their functions for inventory, cost, manpower levels, and budget planning for future systems.

Network mapping also aids the network technician for remotely administered networks. Managing networks remotely is a difficult task because of the lack of direct supervision and physical control of those networks. However, remote administration is becoming more and more popular by organizations that have a small cadre of network and system savvy individuals and numerous enterprises to run. Indeed, that is the thrust of the network consolidation effort going on throughout the Air Force–consolidation of common services by the major commands [4, 24]. Administrators of these systems remotely manage and monitor critical systems at the Air Force bases under their control. A standardized and accurate network map of the systems at their remote sites provides administrators consistent information.

Finally, network maps and network mapping play a large role in the hacker and information warfare community. Tools and techniques developed over time enable these individuals to quickly and easily perform reconnaissance on an adversary's network. This information gives the attacker a "footprint" of the topology of the enemy's network and knowledge of critical systems within that network. The techniques and tools in-use are primarily active methods.

2.5 Active Network Mapping Techniques

Network mapping techniques fall into one of two broad categories: active and passive. Active techniques involve a person or process who actively probes a network to elicit information. Active techniques typically rely on stimulus-response–the network or network system responds to a stimulus given by the network mapping process. Passive techniques simply "listen in" and do not actively probe the network.

The simplest technique (and perhaps the most primitive method of active mapping) is a hands-on inventory of all systems connected to the network. While archaic compared to the techniques discussed later, this still remains a viable option. Indeed, Automated Data Processing Equipment (ADPE) custodians throughout the Air Force must perform a hands-on inventory of their computer-related assets at least once yearly, a tedious and often arduous task [25]. While performing this task, the ADPE custodian, either with pen and paper or by automated process, collects information to build a network map. In fact, this technique provides more information than the automated methods listed below. The ADPE custodian is supposed to provide physical and environmental attributes of a given system. Specific information includes the system's owner or primary user, and the location (building and office) in which it is located. This is a daunting task, especially when users move computers without the custodian's knowledge. Simpler, more automated methods are needed and luckily, are available.

Next on the complexity level would be a *ping sweep*. The *ping* command works much like sonar–sending out a signal and waiting for the echo of that signal to return. Using the Internet Control Message Protocol (ICMP) Echo function, *ping* sends a single, serial-numbered packet to a given address [41]. If a host exists at that address, it is supposed to reply, as required by the Internet Protocol (IP) standard, to the originator using the same serial number [42]. The *ping* command gives the network mapper one important piece of information: whether a system exists at a given IP address. A ping sweep extends the *ping* model and uses a simple looping technique to ping all of the addresses within a given IP address range. The sweep typically encompasses an entire network. Ping sweeps are simple to implement because the *ping* command is a common part of the IP protocol suite that exists on Internet capable systems. A simple script can use the *ping* command, iterate through the given addresses, and save the results in a log file for later perusal by an individual or an automated system.

The primary drawback to this technique is that quite a few administrators disable or block ICMP echo packets at their border gateways or even within their metropolitan area networks (MANs) or LANs. As a result, this technique cannot be used on those networks. Second, the system being probed must respond to the echo request. The given system could simply be turned off, as is the case for workstations which are routinely shutdown at the end of the workday. They could even be configured not to answer the ping request. Finally, the ping sweep doesn't reveal much more information than "exists and answering."

When used in a "local" network, one which the administrator has control, a ping sweep is a fast and effective method of finding out which IP addresses within the network are in use by hosts. The administrator is able to reach out and "touch" systems via the network without obstacles. When used against an adversary, however, this becomes a tricky matter. As stated above, security-minded network administrators often block ICMP messages at the network border router or gateway firewall to keep outsiders from gaining information about their network. Even if a system does respond, it could be a false flag, or even a honeypot designed to draw attention away from actual production network resources [50]. Results of a ping sweep against an adversary should be used with a great deal of skepticism.

Another standard tool built upon the ICMP protocol is *traceroute*. *traceroute* uses the Time-To-Live (TTL) field present in every IP packet. The TTL is simply a network "hop" counter. Every device along a packet's path automatically decrements the packet's TTL. When the counter reaches zero, the network device is supposed to ignore or drop that packet and send an ICMP Time Exceeded message back to the originator, giving it the chance to take further action with that packet [41, 42]. By starting with a TTL of one, and incrementing one hop at a time, the traceroute utility can send an IP packet destined for a target address and keep track of the systems that report ICMP Time Exceeded for each "hop distance" [3]. Thus, *traceroute* can be used to discover the topology of the network. By using the list of active hosts

generated by a ping sweep, the network mapper can discover the pathways between various machines on the network. When using the *traceroute* command, a mapper tracks nodes which are repeated when contacting different hosts and can determine where primary nodes (i.e., routers) are throughout the network. Thus, the mapper can build a hierarchical topology of systems and network devices which support those systems.

Using *traceroute* in a LAN or MAN which a network manager has absolute control over makes for quick discovery of those network devices and the routes information takes through the network. When used against an adversary, this method works only as well as that adversary's network administrator permits. Simply blocking the ICMP Time Exceeded from traversing out a network keeps *traceroute* from working against the devices within the network. However, that same network trace information can give a good picture of the interconnection of these systems and the critical paths which the network needs to function.

Researchers at Lumeta Corp have devised an automated way to use *traceroute* to generate map data and then create visualizations of the Internet from that data. Termed "peacock maps," their stunning visualizations have actually become quite popular within the Internet community and are being sold on-line to enthusiasts and art galleries alike [17].

Similar research is being conducted by the"skitter" project developed by the Cooperative Association for Internet Data Analysis (CAIDA) [33]. This tool maps the Internet using round-trip time calculations to over half a million Internet devices around the world. It provides a picture of the topological connectivity of the Internet, but doesn't go the next step of discovering services. Granted, for the scale of their project, gathering "services-provided" data for 500,000 hosts would be impractical, at least on a frequent and recurring basis. Furthermore, this mapping technique doesn't "discover" new information about the network, simply the topology to known

machines [33]. Indeed, discovery of the millions of nodes throughout the entire Internet would be a vast undertaking in and of itself.

Moving up the network stack to the transport layer, the Transmission Control Protocol (TCP) provides a standard method of end-to-end connections using the IP protocol. TCP guarantees message delivery and "in-order" packet delivery. The TCP layer further separates a given IP address into sub-addresses called ports. Consider an apartment building analogy: the address specifies the building location while the apartment number tells the exact location of an individual. IP addresses are equivalent to the apartment building address and ports are equivalent to the apartment number. In TCP, the port address is a 16-bit value. Therefore, the ports take on decimal values between 1 through 65535 (port 0 is not normally used). Each port number represents a service provided by a server. Ports 1-1024 are reserved for well-known services. For instance, the Simple Mail Transfer Protocol (SMTP), the de-facto standard for passing electronic mail, is typically found at port 25 and Hypertext Transfer Protocol (HTTP, the protocol of the world-wide web), is typically found at port 80 [45]. "Typically" is used because a savvy network administrator can change the ports from their default values to keep an adversary, or curious user, guessing.

When a service is running and attached to its specified port, the service is considered to be "listening" on that "open" port–it is listening to the network for a connection request by a client. During the setup of a TCP connection, the two systems creating the connection execute a three-way handshake. The system initiating the connection, usually a client system in the client-server model, begins the handshake by sending a synchronize (SYN) message which, among other things, identifies the port to which it wants to connect. The service provider acknowledges the handshake by replying to the client with a synchronize and acknowledge (SYN|ACK) message. The initiating system completes the handshake with an acknowledgement (ACK) message. It is at this point that the higher-level protocol may begin to exchange

information. A similar tear-down handshake is required to remove this connection when the transaction completes. If the server doesn't have a listening service on the requested port, it is supposed to reply with a reset (RST) message after the initial SYN. That is, both sides abort the connection before it has begun–no further messages need to be exchanged [43].

Using this protocol knowledge, a program can be written to perform what has been termed a *port scan*. The program performs a three-way handshake on all available ports either using a predefined list of IP addresses or building one on the fly (using a technique similar to a ping sweep). When the server completes the three-way handshake, the port scanning program knows the server has an open or listening port and, potentially, which service that server is providing. The process of initiating and tearing down the connection requires a minimum of seven packets–three for the initial handshake and four for the tear-down [43]. To omit the extra packets for the tear-down, once the port scanner receives the SYN|ACK message and knows that the server has a service listening, it can answer the third part of the handshake with a RST message which, as stated above, aborts the connection. Therefore, it only takes three packets to determine whether a given port is open. This reduces the number of packets that cross the network. As an example port scan tool, *nmap* is an extremely popular port scanner available for multiple operating systems. Its technology has been incorporated as an integral part of several commercial and commercial-quality network inspection and network defense tools [56]. Figure 2.1 shows a typical *nmap* run on a single system.

One side-benefit of a port scan is the knowledge of how the probed system responds to the scan. Using the TCP header fields and the flags within those fields, coupled with prior experimental knowledge, *nmap* can often determine the operating system/platform of the host system, solely based on how it replies. *nmap* could (but currently does not) make similar guesses based on the services it finds open (e.g., Microsoft uses several specific ports to provide file and printer resource sharing, which

```
C:\ > nmap -sS -P0 -p 1-2048 -O -T 3 192.168.0.37
Starting nmap V. 3.00 ( www.insecure.org/nmap )
Interesting ports on SERVER (192.168.0.10):
(The 2042 ports scanned but not shown below are in state: closed)
Port State Service
13/tcp open daytime
37/tcp open time
135/tcp open loc-srv
139/tcp open netbios-ssn
445/tcp open microsoft-ds
1025/tcp open NFS-or-IIS
Remote operating system guess: Windows Me, Win 2000, or WinXP
Nmap run completed – 1 IP address (1 host up) scanned in 1 second
```

Figure 2.1: *nmap* execution [56]

other operating systems do not ordinarily use). This process is often called "fingerprinting" the host's operating system. As a network administrator, this information is quite useful for managing a homogeneous network. When used in multiple session over time, a network administrator can discover new systems, and possibly new services, within the network. Non-standard or unauthorized operating systems within the network can also be discovered.

A more aggressive operating system fingerprinter is *Xprobe*. Instead of the brute force methods of a port-scanner, *Xprobe* sends a minimal number of specially crafted ICMP packets to the target system to determine key aspects of the system and then crafts its next packet to refine its information. While still using a signature-based database to match against, it uses "fuzzy logic" to make decisions about how well the signature matches its database entries and streamlines the probing packets to better fit the guess [15]. Note, however, that *Xprobe* only fingerprints systems–it does not map services to systems like *nmap*.

When used against an adversary, a network aggressor can use the results of fingerprinting to focus an attack. Given a particular operating system (or class of operating systems), the aggressor can determine whether or not a certain vulnera-

bility exists and possible exploits of the adversarial system. For instance, if a system is fingerprinted as a Windows 2000 Server, it is unlikely that an exploit against a Cisco Ethernet switch would be effective, leading the network warrior down a more fruitful path with more effective exploits.

Another pseudo-active technique uses the Simple Network Management Protocol (SNMP). SNMP is an application-layer communication protocol used as a method of managing TCP/IP networks, including individual network devices, and devices in aggregate [27]. SNMP also reverses the client-server architecture. A network monitoring workstation or server is actually the client requesting information while an SNMP agent resides and runs on the system to be monitored, answering requests for information and performing functions on the specific system. The protocol allows a network programmer to peruse specific data about networked systems. Since device enumeration was a commonly performed task, the tool *snmpwalk* was developed. *snmpwalk* uses the SNMP "GET" and "GETNEXT" requests to connect to a SNMP-enabled system. It enumerates all of that system's SNMP variables and their contents, effectively dumping the entire contents of the SNMP agent's database. A sample *snmpwalk* command and its results are shown in Figure 2.2, below.

```
> snmpwalk -Os -c public zeus system

sysDescr.0 = "SunOS zeus.net.cmu.edu 4.1.3_U1 1 sun4m"
sysObjectID.0 = OID: enterprises.hp.nm.hpsystem.10.1.1
sysUpTime.0 = Timeticks: (155274552) 17 days, 23:19:05
sysContact.0 = ""
sysName.0 = "zeus.net.cmu.edu"
sysLocation.0 = ""
sysServices.0 = 72 [snm2002]
```

Figure 2.2: *snmpwalk* execution [38]

This process provides a great deal of information which can be used by the network mapper to create detailed maps of the network. The primary drawback is that every system included in the mapping process requires a properly configured

SNMP agent to respond to the monitoring station. Thus, the network administrator requires physical or remote control of the system in order to provide the correct information and passwords into the systems. The use of SNMP can grow unwieldy if the network manager is expected to configure the whole enterprise for SNMP, down to individual workstations. Consider an Air Force base as large as Travis Air Force Base. Its 10,000 person population requires over 5,000 computer systems, client and server alike, to keep its people on the network. Usually, only those critical systems (e.g., electronic mail servers, gateway routers, server-side Ethernet switches, etc.) under the system and network administrator's direct control are configured with an SNMP agent for monitoring and control, greatly simplifying configuration management of those systems and the SNMP configuration in general. Configuring a large system of network devices to work with SNMP is both an intimidating and highly political tasking due to the large number of hosts and organizational spans of control the system crosses. SNMP also allows changes to be made on SNMP-enabled systems (if the agent and its configuration on that system support and allow it). Indeed, hackers and adversaries can exploit SNMP in order to send commands to cause critical systems to fail. SNMP requires a great deal of understanding as well as a security-minded individual to provide strong protection against external or internal invaders.

2.6 Strengths and Weaknesses of Active Mapping

Active techniques are a boon for network administrators. Using all of the above techniques, Hewlett Packard's OpenView - Network Node Manager automatically discovers, maps and tracks systems and network devices throughout the network. It uses a combination of active techniques during its network discovery process. "An ICMP ping is issued for each device ... and an *snmpwalk* command is issued to gather information about each discovered device" [31].

Active methods are accurate. A ping can tell the mapper whether or not a system exists as well as the address of that system. It can infer the network address based on its address and can determine where that system lies within the network hierarchy. Using *nmap*'s port-scanning and operating system fingerprinting techniques, the mapper can further refine the data within the map and provide an accurate picture of the network at hand, the services it provides and a basic topology. In general, if a machine is providing a service, active techniques will find it. These techniques are listed as the second step of exploration in *Hacking Exposed* as a primary information gathering technique against an adversary's systems [47].

Active methods don't require a great deal of configuration. As long as systems are left in their default configuration (i.e., not configured to block ports and allow ICMP echo messages to flow), a single command line can generate most, if not all of the data needed to generate a cohesive network map. There is no need to touch each machine (either through the network or via hands-on) to enable an active scan of that system. This is especially helpful when the mapping target is an adversary's system.

Active methods are quick. It only takes one round-trip time from the mapper to the target to determine the existence of a system and one more round-trip time to determine if a given service port is listening–if the probed machine isn't configured to drop connections to non-listening ports. With one ping packet, the mapper can determine whether the system exists; with another three packets, it can determine whether or not a given port is open or closed (not responding). Using *nmap*, a network manager can scan the network for live systems using a myriad of functions. Comparing multiple results over time, scans can determine if new or changed services exist. The scan can also discover rogue system architectures–possible attempts by an adversary to infiltrate the network, all within the round trip time of a packet to flow across the network.

There are drawbacks to active scanning. Probably the most obvious is that the probe can be detected by an adversary. Active scanning techniques are analogous to active radar or sonar; the adversary knows their system is being probed. When a stream of ping requests or port scan packets comes from a system, two pieces of information are conveyed. The first is the probe itself and second is the probed system identify the source of the probe via the IP addresses. There are techniques such as reflection or "zombied" machines which provide some anonymity for the probe, but the adversary still knows a probe has taken place. With that knowledge, the adversary has the capability to react to the probe. The reaction can take the form of shutting down access by creating filters on routers and firewalls. Certain portions of the network architecture can be changed after the probe so the systems are not the same after the probe, rendering the data from the probe useless. If the adversary decides to retaliate, "hack backs" or denial of service attacks could be launched against the probing system.

These active techniques also add extra traffic to the network. In an adversarial role, with the probe pointed outward, the additional traffic is probably a not an issue. However, during the probe, traffic needs to exit the probe's network, transit through the larger Internet to the target system, then finally navigate its way back to the probing system. This could have the effect of rendering the adversary's network unusable or unreachable if it becomes inundated. It could also affect the performance of the local network during the probe, due to the exiting and returning packets through the gateways. Connectivity to the outside network could fail due to the overwhelming response.

When active techniques are used to scan an internal class B network of potentially 65535 nodes, the network can be quickly inundated with probing packets. For instance, in its default configuration, *nmap* initially attempts an ICMP ping to check for liveness, and then scans for 1,675 well-known ports. Scanning each system requires 3 packets x 1,675 ports or 5,025 packets. For a modest class B network

with 1,000 nodes, this equates to over 5 million *extra* packets on the network. While most networks can easily handle that much traffic, this could overwhelm a small, poorly implemented or hastily deployed network. In addition, a network administrator might use the complete spectrum of port addresses to provide services. Indeed, a standard scan of an Air Force base's network infrastructure checks every port for accessibility as well as enumerating potential system vulnerabilities using Internet Security Scanner [26]. Therefore, there are 3 packets x 65,535 ports = 196,605 packets for one machine and 196,605,000 packets for the same 1,000 node network! A configuration item within *nmap* can throttle the frequency of requests. Normally *nmap* sends its next packet as soon as it receives the results from the previous one, thereby hitting the target host with its next request. Use of this technique will keep the mapper from overwhelming the network or any one system with packets, but increases the completion time of the port scan.

Finally, active measures can be easily foiled. Simply setting up a firewall around the critical network systems, especially in proxy mode–rewriting packets instead of simply filtering and passing them on–greatly hinders active mapping. Many operating systems now include built-in firewalls (i.e., Windows XP, Linux) which accept only known port or IP address traffic and block active mapping.

2.7 Passive Network Mapping Techniques

Passive network mapping techniques collect the network traffic generated by normal, day-to-day activity to form a map of the network. No extra traffic is needed nor generated because the mapper has full access to the network traffic as it passes through the system. Therefore, it is quite possible to use passive techniques without detection.

Similar to active mapping, passive techniques rely on the stimulus-response of network systems talking to one another. The stimulus is provided by the normal activity of clients and their users, not by the third-party mapping system. While

active techniques allow the mapper to send specially crafted packets to obtain focused information, passive techniques do not unless the passive mapper happens to see responses to the extra active mapping traffic occurring on the network it is monitoring.

Passive techniques should be designed to be truly passive. That is, these techniques should not add traffic to the network while performing mapping duties. As a side-effect of this approach, the passive mapper can discover low-uptime systems. While an active measure "misses" a system because it is offline, passive techniques would ultimately catch it when it transmits data [29]. Finally, these passive techniques only detect ports and services on systems that are in use. Conversely, active systems waste a great deal of time looking for ports that may not even be open. A passive system can immediately determine whether a system is providing a service on a given port. Like the active system, it can make note of a successful three-way handshake upon connection of a client to a server and then record that server as having the service. This is similar to the way the active port scan decides whether or not the port is open based on a (possibly aborted) TCP connection. Since a passive mapper receives all network traffic, it is quite possible that the mapper could miss packets or become overwhelmed by large volume networks. There are techniques, to be discussed further in Chapter 3, that address this problem.

A device or sensor that can observe all of the traffic coming across a network is needed. This device is essentially a protocol analyzer, or "sniffer." The sniffer must be in a position within the logical network where a great deal of a given site's traffic and a wide range of traffic types are present to be effective. This location depends a great deal on the purpose of the information gathering. For a friendly installation, the sensor can be placed anywhere. Ideally, the sensor would be placed where it could listen in on the service providers or possibly near gateways to other networks (such as the Internet at large).

Giovanni suggests a network choke point, such as a gateway router or firewall for the location of the sniffer [30]. This is how an intrusion detection system (IDS) is typically placed. The IDS monitors all traffic in- and out-bound through the network boundary to detects from outside (or even inside) the network.

There is, however, a problem with this technique. Systems targeted for monitoring must have a system connection on the other side of the choke point. In other words, by placing the sensor at the gateway border, all of the local "self" traffic of systems is lost. Recovering the self traffic requires the sniffers to be logically close to the server or client requesting the service.

For use in an adversarial role, the choke-point model can be flipped. Instead of listening to traffic "departing" through the gateway, sensors are distributed throughout the Internet in front of educational, commercial and government gateways. These sensors listen for and analyze inbound traffic coming from an adversary's network to categorize the adversary's network systems.

This leads to a number of obstacles, not the least of which is that the adversary must connect to those institutions or corporations where the sensors are placed. Second, getting permission for the installation of these sensors could be extremely difficult, especially if the installation was a government-run project. Third, the number of provided services would have to be quite diverse. Since most services provided these days are simply web or ftp services (especially for anonymous remote connections), a diverse set of requests may not be seen. Indeed, the only traffic would be from systems which are acting as clients. These would be end-user systems (normally, administrators do not use their mission-critical systems to arbitrarily "surf the 'net") and little would be gained. Finally, a data collection and aggregation system is needed along with a method to get data from disparate sites to that aggregation system to sort through the volume of data collected. Doing so would suddenly increase the site's network traffic by transporting information about the incoming traffic to the aggregation point. Global organizations likely will not be

willing to decrease bandwidth resources and violate customer/employee privacy to support this. However, given a diverse enough set of sites and services, this would probably work. The Russian Federal Security Service (the successor of the KGB) thinks so. Indeed, an AP article [48] reported that this organization has "opened a hole" in most Russian Internet providers' systems for monitoring and possible flow control of system data. Not only does this organization have access to their own citizens' communications, but also the traffic generated by their requests and can make inferences about the services outside of their sphere of influence. Information gathered from these systems on the inbound side would be of great intelligence value, if it could be captured, analyzed, and mapped.

This is similar in concept to the distributed Honeynet built by the Honeynet Alliance [8]. While not fully implemented, the Alliance has placed production honeynets and honeypots for hackers to attack in various places around the world. The remaining step is to set up a central database, a clearinghouse, to aggregate signatures and attack logs from those honeynets for further analysis for trends and the like.

2.8 Stumbling Blocks

Passive techniques are quite functional for a local network manager who has full control over network devices. These systems are simpler to configure and manage than active systems because no actual configuration is necessary. Since the passive mapper would not require any specific addressing, it could simply be plugged into the network, turned on and begin mapping the network by listening.

However, using passive techniques against an adversary is quite problematic to implement. While the placement of the sensor is obviously important, the amount of traffic means absolutely nothing if the sensor cannot see any traffic (other than the traffic destined for itself). Ethernet is a shared, broadcast medium with each node on a segment being able to "hear" all the traffic on the wire, no matter if it is destined for

that system or not–it is up to the network driver on each host to determine whether or not the traffic is destined for its host. The shared medium allows a sniffer to function. However, with the advent of layer-2 switches, every device connecting to the switch becomes its own Ethernet segment. By default, the intelligence inside the switch only sends packets destined for a given system to that system and does not broadcast to all the attached systems. As a means of increasing performance by reducing the number of collisions on an Ethernet segment, this switching technique has decreased the ability for a sniffer to work. To overcome this, one of two things must happen to troubleshoot a switched network. One, a non-switched hub is used, connecting the system under test and the network sniffer to it, as well as providing an uplink to the original switch. Thus, the hub extends the original segment so that the sniffer can ride the same wire. This method requires physical access to the systems and would not work in an adversarial role unless human agents infiltrate and place these devices in the adversary's systems. A second option is to configure the original switch to use what Cisco Networks terms a Switched Port Analyzer (SPAN) port [20]. With this function enabled, the given port with the sniffer attached can be mapped to receive duplicate traffic which transits one or more of the other ports. This allows the sniffer to see all traffic that traverses the switch. This technique works in an adversarial role; break into the switch, reconfigure it and start listening. An alert network administrator would notice that something had changed. In this case, success depends on the inefficiency of the adversarial administrator.

Another problem the passive mapper experiences against an adversary is well-administered firewalls or filtered gateways. Not only do these tools block the placement of a sensor, the firewall could be configured in such a way as to keep needed mapping data from getting out to a collection point. There are ways around this such as using covert channels, piggy-backing data, or packets on top of other system's communications, but again, the adversary's ineptitude is needed for success.

Even though passive network mapping can provide valuable information, the task is extremely hard to accomplish. This is a hard problem to solve for many reasons, not the least of which is the placement of the sensor platform. Several research and white papers discuss the technique of passive system fingerprinting [29,39,50,53]. This technique captures third party and the stimulus-response of normal network communications to assess the operating system and platform of two communicating systems. More often than not, packets are captured at a local IDS as an adversary connects to a system or someone connects to theirs. As mentioned above, most operating systems implement the TCP/IP stack with slight differences due to RFC compliance, protocol optimization concerns, or simply different interpretations of the standard. Each stack implementation can be fingerprinted based on reaction to given packets, packet sizes, and options within those packets. Since a passive listener does not have control over system packet flow, additional information may be needed (i.e., more similarities or more differences) to determine the characteristics of the encountered operating system/platforms. Also of note is the fact that knowledge is gained via the traffic which flows past the sensor. While fingerprinting is helpful, it does not reveal which services are running on a given system.

While the above referenced works were largely theoretical, some built working systems were built. The first, a prototype system based on Giovanni's first paper [29], is the program called siphon [16]. It implements a functional passive fingerprinting system, but the developers haven't updated it since 2000. Another, more mainstream passive fingerprinting systems is p0f (which is a "hacker" acronym for "passive operating system fingerprinting"). This application is currently in version 2.0.3 (updated in September 2003) and is actively maintained by Michal Zalewski [57]. It is packaged as part of several Linux operating system distributions. Again while very helpful in determining the probable operating system of two communicants, it does not neces-

sarily help in the field of mapping (although, implementing some of this code in a mapping system might provide additional insight).

Built on the success of earlier versions of p0f and other tools, the program ettercap [7] is a general purpose network utility. This tool has several functions including standard network sniffing, man-in-the-middle sniffing via ARP poisoning, and passive network mapping. The passive mapping function relies on the network traffic being visible to the host it is running on, and does not work remotely. Not only does it map TCP-based services to a given IP address, but it also identifies the Ethernet address (often called the MAC–media access control–address), host names gleaned from name service packets, and it uses an early p0f engine to fingerprint the hosts of packets it sees.

Other research related to passive monitoring lies within the analysis of the traffic itself, not specifically mapping topologies and the system services provided. Brownlee [18] used passive techniques to analyze the utilization of his university's external link to the Internet. This research focused primarily on the Transport layer, examining TCP and User Datagram Protocol (UDP) traffic flows. A secondary focus investigated web-centric traffic. No attempts were made to perform mapping, only load and flow analysis. Cohen [23] also used passive techniques (basically sniffing) to discover differences between the observed and actual network topology. These differences were present for months in reports but analysis had not been accomplished to reveal them. Ultimately, Cohen developed a new network map for the system to assist in the troubleshooting. Finally, three studies [21,22,28] concentrated on network sampling technologies to profile Internet traffic. These studies examined high-speed, high-utilization links where even high-performance analysis systems failed because of the volume of traffic that needed to be analyzed. Instead of analyzing every single packet, the researchers looked into statistical methods and temporal analysis to determine traffic flows. While analysis of high-speed links and characterization of

that traffic is important, it is not of much use to the network manager or the person who desires to find out information about a specific network.

2.10 Summary

This chapter discussed the need for network mapping, active and passive techniques in general and specific versions of those techniques in detail. Active techniques have already proved their usefulness. Passive systems have yet to prove their worth. There is potential in these systems that have not yet been tapped. The difficulty in implementation lies with devising network devices and keeping the systems stealthy.

3. Methology

3.1 Introduction

This chapter describes the thesis methodology. Section 2 describes the specification of a notional passive mapping system. Section 3 describes the design of a custom passive mapping program. Section 4 describes the design and development of the test-bed environment which meets the needs of both active and passive mappers. Section 5 discusses the design of the thesis experiments. Section 6 describes the evaluation technique. Section 7 describes the overall experiment, including the execution of an experimental trial.

3.2 Active Mapping

There are a number of freely and commercially available network port-scanners which can actively map networks. However, *nmap* is selected in this investigation as a representative sample–and possibly the best–of all active mapping systems. This tool, *nmap*, is freely available, the source code is open to the public and it is one of the most popular and widely-used freeware hacking tool in-use [37, 44]. *nmap* is currently ranked #12 in popularity of the registered 31,708 projects on freshmeat–an online index of Unix-based software projects [2]–and was ranked #9 in September of 2003 [55]. In fact, it has reached mainline stardom being featured as a hacker tool in a sequence in the motion picture The Matrix Reloaded [54]. The more difficult part of this research is finding a passive system that meets the needs of a network administrator (or hacker).

3.3 Specification of a Passive Mapping System

A passive mapping system is needed to compare the performance of a passive technique with that of an active one. The requirements of a notional passive mapping system are developed, below.

3.3.1 Functional Requirements. First and foremost, a passive mapping system must use non-invasive means to discover information about the network. Eliciting a reaction from the network environment by sending out specially crafted packets not acceptable. Second, the system should provide a method for getting the discovered information out of the network it is monitoring. This can be done with a two-part system. First, is the sensor which listens to the network and forwards information to a collector. The collector processes the input from the sensor and formats the data into a readable format and/or visualization. The sensor should be self-configuring. In other words, after connecting the system/device into a network (or running the program on an existing system), the program should discover everything it needs to operate.

Since the system must remain stealthy, the sensor cannot transmit a persistent stream of information through the network. Instead, it must buffer as much information as possible and only then send an information packet to the collector. Built-in timers must allow an unfilled buffer to be transmitted after a given timeout value to keep data from stagnating or getting lost due to sensor system failure. To further optimize outbound traffic, the sensor must remember previously transmitted items and not repeat transmission of those items. The sensor must make note of the addresses and ports of transactions which transit outside the network boundaries. Then, it can masquerade as a known (and probably trusted) host inside the network. Using such a technique may enable the sensor to more easily and stealthily get its information past border gateways and firewalls. The collector must present the gathered information in an appropriate manner for user viewing, whether a list of discovered servers, clients and services or a map laying out the discovered network.

As no freely available system has been designed that meet all these specific requirements, it is necessary to choose a subset of these requirements, find a program which implements a portion or all of them or build a new one. This chosen subset includes:

- two part system,
- buffered data output,
- timer-based data output,
- non-repeated output, and
- collector formatted data for output.

3.3.2 Non-Functional Requirements. There are non-functional requirements for the passive mapping system. First is keeping the system as simple as possible. Learning how to write or modify a multi-threaded program is a time-consuming process. Keeping the program single-threaded further simplifies code maintenance later on. In addition, using external data sources, such as SQL databases or other database techniques would depend on the system it is running on. Implementing a full-up database is not in the best interest of a tool running in an adversarial mode, trying to remain stealthy. Second, the program needs to be kept as small as possible. If running on an active system in an adversary's network, using a large number of system resources to keep running is not recommended. The more resources consumed, the higher the risk of detection. Finally, the system must be reasonably fast in terms of execution time. A Java-based system, which uses a virtual machine, is slower than, say, a C-language counterpart. Java is also detrimental due to the number of resources consumed during execution. It must be fast enough to examine inbound packets without dropping them.

3.3.3 Code Exploration. With these requirements in-hand, suitable candidates for use or modification were sought. This investigation discovers the following tools:

siphon - Multi-threaded C-language program that uses the PCAP packet capture library [34]. This particular tool is old, unsupported and marginally functional. It is, however, the first program to implement Giovanni's ideas about passive operating system fingerprinting [16, 29]. Operating system fingerprinting, while not part of the mapping research, would add an additional feature to the existing feature set.

p0f - Single-threaded C-language program which uses the PCAP library. This tool performs passive operating system fingerprinting on distant-end systems connecting to the fingerprinting host. This is the initial candidate.

tcpflow - Single-threaded C-language program also uses the PCAP library. The tool stores captured packets as flows in multiple files based on end point IP addresses and connected ports. The entire conversation between two hosts is dumped into a file–one file per conversation–and analyzed later. Like most PCAP-based sniffers, *tcpflow* can also read the input from a PCAP dump file.

ettercap - Multi-threaded interactive C-language program. This is the third candidate after *p0f* and *tcpflow*. This tool has code for the port mapping built-in and is a native network-sniffing tool. As an additional feature, *ettercap* captures "banners" of service providers (such as the banner that an FTP server displays when an individual logs in) which aids in discovery of services as well as fingerprinting the operating system. *ettercap* uses text-based user-interface. It displays mapping information, but requires user-interaction to fully display the information gathered about a given host. *ettercap* has a command-line mode, which non-interactively logs the gathered information. This mode displays only the Ethernet address, the IP address and any fingerprint information it discovers. It does not display ports in-use or the banners that it captures.

3.4 Design of a Passive Mapping System

Study of the main portion of *p0f* and additional study of *ettercap*'s dryad module ultimately led to development of a custom program using similar techniques. *lanmap* applies dryad's mapping technique and uses the PCAP library to capture packets.

The method *ettercap*'s dryad module uses is similar to *nmap*'s–based on the response to stimuli given to the network device. If, upon a probe, *nmap* receives a SYN|ACK response, the port is open and providing a service. If *nmap* instead

receives a RST response, the port is not open. If *nmap* does not receive a reply after a certain timeout period, it knows nothing new about the state of that service or machine (unless it had earlier "detected" the machine with an ICMP ping or another open port–then the port attempt would be considered "filtered" or, essentially firewalled).

Therefore, when dryad sees a SYN packet, the source IP is usually a client requesting a service on the destination IP's system. Dryad does not care much about clients; it is looking for active, responding services. So, if the destination IP responds with a SYN|ACK, dryad marks that IP address as a service provider for the requested port. Dryad ignores RST messages; omission of a port in its "map" simply means it has not seen evidence of the existence of that port in use.

Figure 3.1 is pseudo code which describes the operation of the *lanmap* program. The pseudo code used here is really simplified source code, to ease understanding of the flow of the program.

The pseudo code accurately describes the way the program *lanmap* works. Since *lanmap* is a program designed to discover everything it can about a network, clients findings are important as well. Therefore, SYN messages, are use to mark the source as a client on the attempted port connection.

The Address Resolution Protocol (ARP) and Reverse Address Resolution Protocol (RARP) map unique IP addresses to unique hardware addresses [40]. The hardware address range determines the network card manufacturer. This leads to exploitation of known vulnerabilities in the network card's drivers for a given operating system. For systems such as Dell or Compaq servers, their embedded network hardware addresses uniquely identify the manufacturer of the server so even the brand of machine can be discovered.

The information that ARP/RARP messages provide is redundant: PCAP already provides full Ethernet headers to the calling program. Mapping a destination

```
lanmap()
  open network port with pcap
  initialize emission_timer
  initialize emission_queue
  set interrupt hooks
  loop
    packet ← next pcap packet
    if (packet.protocol = IP) // IP packet?
      // add source Ethernet/IP mapping
      AddToQueue(type = ether, eth_addr = src.ether, ip_addr = src.ip)
      // add destination Ethernet/IP mapping
      AddToQueue(type = ether, eth_addr = dst.ether, ip_addr = dst.ip)
      if (packet.IP.protocol = TCP) // TCP packet?
        if (TCP.flag = SYN and TCP.flag ≠ ACK) // initial request from client?
          AddToQueue(type = client, address = src.IP, port = dst.port)
        else if (TCP.flag = SYN and TCP.flag = ACK) // response from server?
          AddToQueue(type = server, address = src.IP, port = src.port)
        end if // TCP
      else if (packet.IP.protocol = UDP) // UDP packet?
        // add both source and destination IPs as UDP server and client
        AddToQueue(type = uclient, address = src.IP, port = src.port)
        AddToQueue(type = userver, address = src.IP, port = src.port)
      end if // UDP
    end if
    if (emission_timer.expired or emission_queue.nearlyFull)
      emit emission_queue packet
      clear emission_queue
      reset emission_timer
    end if
  end loop
  /* shouldnt get past here, unless interrupted */
  if (interrupted)
    /* by user break (Ctrl-C) or other terminating signal from operating system */
    if (emission_queue.notEmpty)
      /* dying, try to get the last bit of information out */
      emit packet
    end if // queue not empty
  end if // interrupted
  display statistics
end lanmap
```

Figure 3.1: *lanmap* algorithm

IP address to a destination Ethernet address (and the same for the source addresses) only requires one IP packet. Thus, ARP/RARP packets can be ignored.

TCP constitutes the substance of this investigation. Using the methods described above and in the pseudo-code in Figure 3.1, a vast amount of information can be learned about client and server machines and their IP addresses.

UDP is similar to TCP, but no handshaking is into the protocol. There is no way to conclusively determine which end of the connection is a client and which is the server. Therefore, both of the source and destination addresses are marked as a client and server on the appropriate ports. This ensures no mappings are missed, is simple, and results in fewer state variables to track (the last time a given address and port combination were seen).

A particular challenge of *lanmap* is creating efficient internal data structures to maintain knowledge of previous mappings. Instead of using an external database, *lanmap* uses an internal hash table to store mapped items. The hashing function is a simple exclusive-or hash over the mapped item data fields. Finding some simple code online saves time here [1].

collector, the central repository of mapping data, needs a similar database, but requires sorting for display purposes. Therefore, an ordered, doubly linked list is used which collects the mappings transmitted by *lanmap* [49]. See Figure 3.2 for *collector*'s pseudo-code.

3.5 Design Of The Test Environment

During the final stages of programming the *lanmap* and *collector* programs, more realistic testing of the program's correctness in identifying traffic as it traversed the network is needed. It is time to select and start utilizing a suitable network traffic generator. A traffic generator should emulate multiple network clients connecting to multiple network servers on multiple ports and predetermined or randomized times in a repeatable manner.

```
collector()
  open UDP socket for listening
  set interrupt hooks
  loop
    packet ← nextpacket
    for i ← 0..sizeof(packet)
      entry ← packet_i
      if (database.contains(entry))
        do nothing
      else
        AddToDatabase(entry)
      end if
    end for
  end loop
  /* normally don't get here, unless interrupted */
  if (interrupted)
    /* by user "break" (Ctrl-C) or other terminating signal from operating system */
    dump report to screen/log
  end if
end collector
```

Figure 3.2: *collector* algorithm

3.5.1 Traffic Generators. Several candidate traffic generators with promising features are readily available.

tg is one of the earlier traffic generators for the Unix operating system world. *tg* tests end-to-end capacity and throughput. It consists of a pair of programs, one the sink side and the other the source side. Once the sink is running and the source launches, the source sends as much data as fast as it can while keeping track of how much is sent. The sink also keeps track of what it receives. A third script/program combines the source and sink log files and describes the performance and capacity of the channel.

While a good tool for empirically testing end-to-end capacity, error rate, and bandwidth, it does not meet the needs of the passive mapper. The passive mapper needs to see diverse traffic from multiple sources to multiple sinks. *tg* only provides for connectivity on one port at a time. Worse, neither the source nor sink can be bound to a specific IP address on the local system.

NetSpec provides an array of scripting functions that can perform simultaneously or in synchronous lock step and (also has a single point to control the number of systems involved in the test. *NetSpec* is used to provide network traffic while testing *lanmap* and collector during final stages of completion. However due to the manner in which TCP connections expire (i.e., entering the TIME_WAIT state and the server port remaining in-use, but unusable at the same time) [43]. *NetSpec* will not start a second test using the same server-side port number because that port is still marked as "in-use." [11]

NISTNet generates traffic and can inject delay and jitter into network streams. *NISTNet* is installed by by patching into the Linux kernel. Since *NISTNet* destabilizes the test bed computer systems and constantly caused kernel panic dumps, it does not provide for any salient evaluations [12].

tcpreplay injects TCP trace traffic (sniffer data files) onto the network from a *tcpdump* file. *tcpreplay* was developed "in the hopes that a more precise testing methodology might be applied to the area of network intrusion detection." It is able to change the IP addresses and Ethernet addresses of source and destination nodes so that passive devices like sniffers and intrusion detection systems as well as routers and firewalls can react as if the traffic were coming from real hosts within the network. This method is not appropriate–the complex interplay of client-to-server connections is captured with single-sided script and reduced to network traces. Moreover, as further explored below, this method does not provide an appropriate environment for the active mapping system [52].

Chariot was originally coded and provided by NetIQ but development has been taken over by Ixia. *Chariot* is also a network testing application and is quite robust. Like *NetSpec*, *Chariot* runs on multiple machines and provides a single console to control the experiment. It also has a flexible GUI for control as well as after-action data gathering and analysis. It will automatically send a RST message to open ports when testing is complete. This means the same system can restart using the same

server port without delay. Additionally, *Chariot* has many scripts for simulating various network conversations (e.g., client and server-side ftp session or client and server side Windows login session) to further extend its possibilities [6].

LARIAT (Lincoln Adaptable Real-time Information Assurance Test bed) was funded by the Air Force and DARPA, built by Massachusetts Institute of Technology's (MIT) Lincoln Labs and is capable of generating LAN-like traffic based on observed user behaviors. The system uses a complex database and multiple Windows and Unix/Linux machines to generate traffic, simulate servers, DNS domains and sampled Internet web pages.

NISTNet is essentially a pass-through or traffic generating product; it is not meant to provide services. *tcpreplay* transmits packets around the network for IDS-like systems to react to and provides no port-based services. The other three traffic generators are still configured pair-wise; even *Chariot*, with its superior GUI and capability to create banks of test systems, ultimately configures the experiment as a group of client-server pairs. When the phase of an experiment has two systems talking to one another on a specific server port, that port is open and available (and detectable by an active scanner such as *nmap*). However, when that part of the experiment is completed, the port is not detectable because it is not open constantly as a true server port would be. A method is needed which works properly with both active and passive methods.

The best method is one that generates traffic on various network ports using appropriate protocols. At the same time, it needs to look like a full network configuration with servers providing properly operating open ports, providing services. The best candidate, *LARIAT*, provides both of the requirements but it requires too many resources [14]. Therefore, this investigation uses *honeyd*.

3.5.2 honeyd. *honeyd* is a program that creates virtual honeypots. A honeypot is an information resource that looks like a true server, but has no real

production value except for being attacked or exploited by hackers. It is a lure, not a true service-provider. Therefore, any network connections to or from the system is likely a probe, an attack or a compromise [8, 50]. *honeyd* can emulate not just one system, but a complex network of them. Each virtual system in the honeypot network (honeynet) has a profile so that it reacts as a real server would. For example, given the profile of a Windows 2000 server, the virtual honeypot would react, at the TCP/IP stack level, just like a Windows 2000 server would. Thus, active and passive operating system fingerprinting systems like *nmap* or *Xprobe* identify the virtual honeypot system as a Windows 2000 server. *honeyd* also has a scripting function which allows specialization or expansion of various aspects of the operation of the virtual machine. Network packets are passed to the script, which acts on them and generates output. Output is sent back into *honeyd* as a reply to the originator of the connection. So, a complex system of virtual systems can be configured and act and react like suite of servers. The *honeyd* configuration is developed, tested and deployed (see Appendix D).

3.5.3 syntraf. Setting up a suite of clients to inject traffic into the network and interact with the newly constructed suite of servers is more complex than it seems. However, when a standard client program, such as *sendmail*, accesses the network, it usually uses the default IP address for the network adapter; the program could choose an arbitrary IP address to use during the communication with the server, but unless that functionality is built into the program, a rewrite of the utility is required. Without this support, all output connecting to the servers appears to come from one IP address, not a suite of clients, which, while not wrong, still does not generate the number of clients connecting to many servers model of a network. Since no viable tool was discovered in the traffic generator search, some sort of custom program is needed. The tool *syntraf* was created to serve as the client-side traffic generator.

Figure 3.3 shows a high-level block design of *syntraf*. Development of several small Java classes, each built upon a simple TCP client-server model program [32,51] creates a suite of client-side modules capable of exchanging messages with servers using similar protocols. For example, consider a client module that does nothing more than perform simple mail transfer protocol (SMTP, e-mail) sessions. It is capable of creating an e-mail message and connecting to an SMTP server to fully transmit it.

A client sequence class schedules the various client modules. Each client sequence simulates a given client machine and executes its client modules in some given, repeatable order. This generator randomly selects which client modules executes next. No weight or proportionality is given to any single module; each is as likely as the next to get chosen. While this technique does not address the self-similar nature of network traffic and is simplistic in nature, it is able to provide appropriate traffic for the passive mapper–the primary thrust of this investigation.

Finally, *syntraf* has a simulation controller class which reads and parses a user-specified configuration file and creates all of the client sequencers and then launches them as threaded processes. The only remaining part is a configuration file describing the client environment. Using personal knowledge and a mix of client-side users, a client-side configuration file is generated which gives an interesting mix of client-side modules. Appendix C, Section C.1.4 for Traffic Generator Source Code and Section C.2.2, Traffic Generator Configuration Files can be consulted for more information.

3.5.4 Verification of Code Operation.

3.5.4.1 lanmap and collector. To ensure that *lanmap* and *collector* are responding as designed, they were incrementally tested with actual e-mail and web traffic while using the UNIX-based e-mail program pine and the Netscape-like web browser Mozilla Firebird to connect to real servers. At each phase of imple-

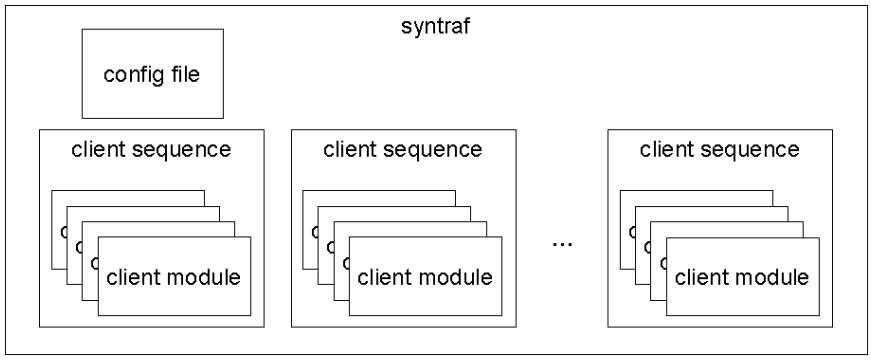

Figure 3.3: *syntraf* System Block Diagram

mentation, both of these tools are used to ensure continuing operation of *lanmap* and *collector*. When a single connection is made between client and server, four mappings are expected:

- Client IP to Client Ethernet Address
- Client IP to Client-side Service
- Server IP to Server Ethernet Address
- Server IP to Server-side Service

When repeating the connection, no additional mappings are created. As long as this holds for each client-server-service connection, the program is acting properly. If the server IP address is on a different network (which requires a hop across a gateway), then the server IP is mapped to the gateway's Ethernet address. If the client attempts connection to a target IP address which does not exist, either zero or two mappings are generated. If the IP address is within the network address range assigned to the client (calculated through the given IP address and network mask and/or given network address), the client host will perform an ARP lookup to find the local Ethernet address assigned to the requested IP address. If this ARP fails, then no IP traffic is generated and, therefore, no mappings are generated. However, if the target address is outside of the network scope, the TCP/IP connection will attempt to connect through the default gateway configured on the client. In this case, the following three mappings are generated:

- Client IP to Client Ethernet Address
- Client IP to Client-side Service
- Server IP to Gateway Ethernet Address

Since the client is attempting to connect, it is assumed that the client is a consumer of the given service, even if it cannot successfully connect. Finally, when a client attempts connection to an existing IP address where the service is not being provided, three mappings are generated:

- Client IP to Client Ethernet Address
- Client IP to Client-side Service
- Server IP to Server Ethernet Address

The server-side connection will not succeed. It will generate a RST message on a normally configured system. On a firewalled system, it should not reply. In either case, the server IP-to-Ethernet address mapping would not be generated.

3.5.4.2 syntraf. During the development of *syntraf*, each client module (e-mail, ftp, web, etc.) is tested against live servers to ensure proper operation. Each client's protocol document (Internet Request for Comments) is also consulted to take into effect any deviations unknown to the researcher as well as specific error codes and messages emitted by a server of the given protocol. Finally, *syntraf*, *lanmap*, and *collector* are tested together. *tcpdump* traces are collected to ensure that *lanmap* sees what *syntraf* generates and *collector* sees what *lanmap* outputs. The traffic generated by each program is meticulously checked to ensure full system flow of information. During the experimentation phase, several traces are hand analyzed to ensure the flow matches.

3.5.4.3 honeyd. The *honeyd* configuration was debugged using controlled client connections (again, using the telnet, mail, and web programs described above) to ensure proper operation [19]. Then, the *syntraf* and *nmap* programs were incorporated into the testing to ensure proper operation.

3.6 Design Of Experiments

3.6.1 Approach. The methodology used herein is based on the empirical study of actual systems in a controlled environment. Each mapping system is runs on a real network to perform its mapping function. Built-in software measurements and third-party sniffer logs provide the metric collection functions. The results of the mapping session are tallied and compared to the known network configuration and then compared to the other method using statistical methods.

3.6.2 System Boundaries. Each mapping system consists of the mapping probe/sensor itself, and a collector which may coexist on the same system. Figure 3.4 is a block diagram of the overall system.

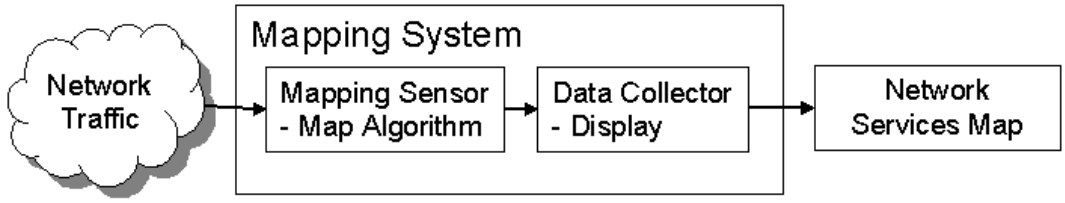

Figure 3.4: Mapping System Block Diagram

3.6.3 System Services. Both the active and passive mapping systems generate a list of IP addresses and the client and/or server ports observed to be functional at that IP address.

3.6.4 Performance Metrics. With respect to the mapping techniques, four separate metrics are collected. Table 3.1 shows the metrics chosen in determining the performance of a network mapping system.

Relevant metrics are those which measure the accuracy and efficiency of the mapper's decision algorithm. The number of server or client address-to-service matches measures the accuracy of the system. This metric is simple to collect in the experimental environment: simply count the number of correct address-to-service matches as appropriate for the client or server. These metrics are gathered from

Table 3.1: Performance Metrics

Performance Metric	**Measure of**
Server address-to-service matches (Server Discoveries)	Accuracy
Client address-to-service matches (Client Discoveries)	Accuracy
Count of packets received (Packets In)	Efficiency
Count of packets transmitted (Packets Out)	Efficiency

logs and reports generated by the mapping system using custom *Perl* scripts (see Appendix C, Sections C.1.1 and C.1.2 for *uncollect.pl* and *ungrep.pl*, respectively) to normalize their respective results. These normalized logs are compared against the traffic generator configuration (for the Client Discoveries) and against the honeyd configuration (for the Server Discoveries) using a custom Java matching program, *matchem* (see Figure 3.5 for pseudo-code or Appendix C, Section C.1.3 for full source code).

Since the experimental network is known, comparing the results from the mapping system to the actual network is simple. False positives mappings are also tracked. If the given mapping system decides that a service exists at an address when it does not, that decision gets added to the false positive count. This metric is also gathered at the time of the matching procedure.

To measure efficiency, the number of packets received by and generated from the mapper are counted. *tcpdump* logs all incoming and outgoing IP and ARP packets that the mapping system's host receives or generates [35]. Since there are no other services running on the mapping system's host during the experiment, only mapping traffic is received by the host's network interface. Additionally, the number of packets dropped by the operating system kernel is collected. Depending on processor speed and the network load, it is possible for the passive network mapper and/or the *tcpdump* sniffer to drop packets. While this is a performance issue, dropped packets can change the outcome of an experiment. "Discovery" metrics are both "higher-is-better" metrics–the more discovered about the network, the better. Conversely, the "Packets" metrics are both "lower-is-better" metrics–the more pack-

```
matchem()
  clientHitCount ← 0
  serverHitCount ← 0
  falsePositiveCount ← 0
  totalItemCount ← 0
  read honeyd configuration
    add to known item database
    update totalItemCount
  read traffic generator configuration
    add to known database
    update totalItemCount
  open discovery log file
  while (not EOF(discovery log)) loop
    entry ← nextrecord
    if (database.contains(entry))
      if (entry.type = server)
        serverHitCount ← serverHitCount + 1
      else if (entry.type = client)
        clientHitCount ← clientHitCount + 1
      end if
    else
      falsePositiveCount ← falsePositiveCount + 1
      dump entry to screen/log
    end if
  end loop
  close discovery log file
  display serverHitCount
  display clientHitCount
  display falsePositiveCount
  display totalItemCount
end matchem
```

Figure 3.5: *matchem* algorithm

ets on the network needed to make those decisions the worse the method is. Total Discoveries is the sum of the Client Discoveries and Server Discoveries metrics for a given trial. It is also "higher-is-better" metric. These metrics are used to further derive the metrics in Table 3.2.

Table 3.2: Derived Performance Metrics

Derived Performance Metric	**Measure of**
Total Discoveries (Combined client and server matches)	Accuracy
False Positives	Accuracy
Overhead Packets In (Count of Inbound Overhead Packets)	Efficiency
Total Network Overhead (Overhead Packets In + Packets Out)	Efficiency

The only metric that bears further definition is the "count of overhead packets in." Both mapping methods (given the passive mapper works in a mapper / collector pair) generate packets. Thus, packets emitted by either system are considered overhead. Input packets, however, are a different matter. The active mapper generates extra traffic, anticipating the target will respond which would generate further overhead traffic, seen as input to the active mapper. The passive mapper only uses the traffic on the network and, therefore, does not observe this phenomena. The passive mapper has no overhead with respect to input. The number of overhead packets in, O_{in}, is calculated as follows:

$$O_{in} = \begin{cases} 0 & : \quad if Method = "Passive" \\ Packets\ In & : \quad if Method = "Active" \end{cases}$$

The "Total Network Overhead" is simply the sum of the two Overhead factors. The Overhead metrics are "lower-is-better" metrics.

3.6.5 System Parameters. System parameters are those configuration items or settings which cause the system to change and affect the way it acts or reacts to a given workload. There are several system parameters that effect the mapping systems. First, of course, is the method by which the network is mapped. Both active

and passive methods have unique characteristics in their operation and generation of results. Second, placement of the sensor within the network is an important parameter. The active mapper needs unfettered access to the hosts it is mapping. This means that it cannot be on the other side of a firewall or equivalent network device. The probes must be allowed through and the responses must be allowed to return to the mapper in order to generate an effective map. A passive mapper needs even more access to the network traffic. Is the mapper on a network hub in the same Ethernet collision domain as the servers? Is it on a hub with clients? Or is it on an Ethernet switch only receiving packets destined for its hardware address? With the passive mapper, the more traffic and greater diversity of traffic it can collect gives it more data to map. The amount of time a mapping system uses to discover the network also changes its performance. With either system, the more time available, the more information gathered and the more accurate the picture formed. The speed and capacity of the sensor platform may also contribute differences in success or failure. In both cases, the sensor platforms need sufficient memory and processor speed sufficient to the task. Otherwise, they are overwhelmed with input and network packets could be dropped, losing the information that they were to provide. Maximum available bandwidth goes hand-in-hand with the CPU and memory capacity. If the mapper is running with a 10 megabit network interface on a moderately utilized 100 megabit network segment, it is likely that the sensor will drop or even miss packets. Finally, latency due to long-haul circuits and multiple hop paths through a network can change the speed at which each mapper receives packets. These parameters are summarized in Table 3.3.

3.6.6 Workload Parameters. The workload the systems are offered affects the performance and outcome of the systems, which result in several parameters. The amount of existing network traffic can affect the speed of the active mapper by increasing delay due to congestion and possibly dropped packets. The greater the amount of this background traffic, the more network services the passive mapper

Table 3.3: System Parameters

Parameter	**Potential Levels**
Mapping method used	active or passive
Amount of time given to perform mapping	0 sec - 1 week
Number of mapping sensors in the network	1 to any number
Location of sensors within the network	same as clients, same as servers
Type of network connection	switch or hub
Location of the *collector*	inside or outside (the probed net)
Available system memory	32MB to any size
Processor type and speed	Pentium 3 500MHz or faster
Network interface type and speed	10MB, 100MB or 1000MB
Network latency	0.0 msec to any latency

can discover. Again, if the passive mapper is overloaded, packets are dropped and accuracy is potentially lost. The pattern of the existing network traffic can vary widely minute-to-minute. Since most traffic is generated by user action (i.e., logins, reading and writing e-mail, fetching web pages, etc.), it is essentially random. The number of clients and the number of servers will also affect the workload. A workload with a large number of clients connecting to a large number of servers will be much different than one with a small number of clients connecting to a large number of servers or vice versa. These workload parameters are summarized at Table 3.4.

Table 3.4: Workload Parameters

Parameter	**Potential Levels**
Number of service providers in the network	1 to any number
Number of services provided by the network	1 to any number
Number of clients in the network	1 to any number
Number of packets traversing the network	1 - 99% utilization
Packet arrival rate	0 up to bandwidth limits
Type of environment	normal or firewalled

3.7 Factors

The following system and workload factors are chosen to provide for a smaller set of experiments. These factors are also chosen as it appears they have the most affect on the outcome of a mapping session.

3.7.1 System Factors. To narrow the scope of this research, this investigation limits the factors to the method used for mapping, either active or passive. Since the method used is the premise of this investigation, this factor is a necessity. The second factor is the Time Given for the mapping function to run. There are five levels to the Time Given factor–times chosen during pilot runs of each technique due to interesting things happening at those times. These two factors are summarized at Table 3.5. Since the testing environment is not capable of putting a sniffer device on the "server" side since it only exists virtually, other factors were not considered. Furthermore, the PCAP library is incapable examining the internal structure of *honeyd*. Using more than one sensor on this test network is not needed; one sensor can see all of the traffic from all the clients in the simulation. A Cisco 2950 switch is used for all tests. Since Ethernet switches are becoming more prolific, finding a hub in a production network is becoming increasingly difficult. Hardware factors such as the CPU, available memory, and the speed of the network adapter are all fixed. Network latency is minimal; the only network device is the switch.

Table 3.5: System Factors

System Factor	**Levels**
Method	active or passive
Time Given (minutes)	0.5, 1.0, 5.0, 10.0, or 40.0

The bandwidth and available CPU speed and memory are fixed for both the active and passive mapping systems. The mapping system is one Dell Dimension 4100 running a 500 MHz Pentium III with 512MB of RAM. The network interface is a 3Com 3C905 10/100Mb Ethernet adapter running at 100Mb, full-duplex. The location of the sensor is on the same network switch as the clients.

3.7.2 Workload Factors. To limit the number of experiments and to focus on the goal of this investigation, the workload factors are limited and adjusted. First, the workload is adjusted by using three randomized traffic generation patterns, the third of which revealed interesting properties during pilot testing. Second, the server environment in which the mapper works is adjusted to two scenarios: firewalled and normal. In the firewalled scenario, each service-providing system is configured such that a connection to any port not configured with a service is dropped, instead of the RST message back to the originator. The normal scenario is just the opposite, and more like a normal system inside a firewalled boundary. These factors are summarized at Table 3.6. The levels in the Traffic Pattern factor are numeric seeds to the random number generator in the *syntraf* traffic generator program.

Table 3.6: Workload Factors

Workload Factor	**Levels**
Traffic Pattern (seed)	1, 2 or 4
Scenario	Firewall or Normal

The other parameters remain fixed. Both of the honeyd scenario configurations provide for 136 unique address-to-service mappings and the traffic generator configuration provides for another 321 unique client-side address-to-service mappings. This amounts to 457 unique items for the mapper systems to find. The packet or message arrival rate is dependent upon the traffic generator and the response of the honeyd network system, and is not considered as a factor.

3.8 Evaluation Technique

This investigation measures real systems in a controlled network environment. Since the network topology and configured services are known a priori, it is a simple matter of comparing found items to the known items.

3.8.1 Systems Environment. The test system environment hardware is illustrated in Figure 3.6 and Table 3.7 shows the specific system information summary.

Table 3.7: Hardware Environment Specifications

System	CPU (MHz)	RAM (MB)	Firmware	Network Card
Switch	Cisco 2950-24	20	12.1(13)EA1	n/a
Sniffer/Collector	PIII/933 (C)	256	A11	3C905
Mapper	PIII/500 (K)	512	A11	3C905
Honeynet	PIII/1000 (C)	512	A11	3C905
Generator	PIII/800 (C)	256	A11	3C905

Each system is running Debian GNU/Linux "unstable" distribution, with Linux Kernel version 2.4.22. The unstable distribution provides the latest updates and program fixes as well as some leading-edge code, but has not yet been officially released. Debian has not released a full stable distribution since late 2000 [5]. Linux is selected as the operating system since its source code is freely available. Many of the hacker tools developed over the past few years have been targeted at Linux. *nmap* and *siphon* were both Linux targets before spreading to other operating systems. Finally, Linux runs well on the limited hardware resources available to this research. Debian has a very robust package dependency checking using the APT utility; two simple commands update the installed software and include any dependencies if additional software is installed.

Each system has two network interfaces, one connected to a control network, the other connected to the test network. The control network is used to remotely login to each machine and execute the commands to perform each experiment. The control network interface uses 10Mbit network cards. The test network is 100Mbit and is the network which is used during the experiments. The two networks are kept logically separate by using separate virtual lans (VLANs) within the Cisco 2950 switch. See Figure 3.6 for an overview.

3.8.2 Software Environment. Each of the systems have specific roles and therefore have specific software installed in the test bed. The software items are detailed in Table 3.8.

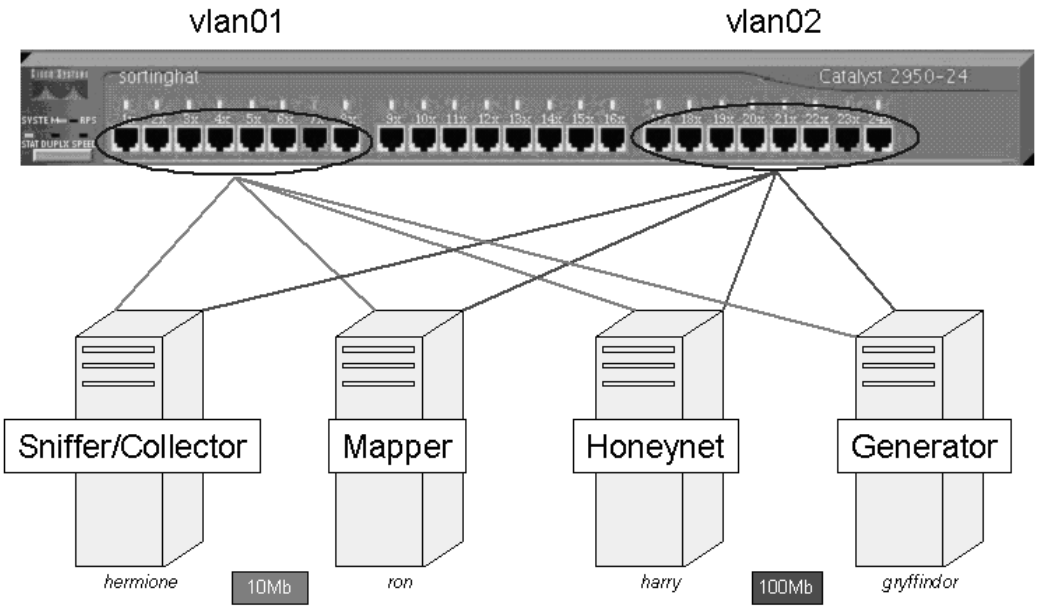

Figure 3.6: Hardware Environment

Table 3.8: Major Software Components

Software	**Primary System**
tcpdump v3.7.2	All
honeyd v0.7a	Honeynet
lanmap / collector	Mapper and Sniffer / collector
nmap (timer modification) v3.48	Mapper
syntraf	Generator
KDevelop v2.1 (KDE)	Generator

Since time is a factor in the experiments, the mapping systems must be instrumented with a timing device to halt execution of the mapping systems. It was easier, however, to set a timer in the traffic generator than it was in the C program *lanmap*. When terminating *lanmap*, it dumps collected information to a log. Since it does not rely on the traffic generator for input, *nmap* needs to be stopped. However, when terminating an *nmap* session, *nmap* merely releases memory in use, closes any open files and terminates in an orderly fashion. Any collected information is lost. By adding a command-line option, moving selected procedural variables into the global scope and adding a timer signal handler, a timer-capable *nmap* is born. A session-timeout value, measured in seconds, can be specified. When the timer expires, *nmap* stops probing, ignores any incoming packets, and displays or logs the network information it collected thus far.

3.8.3 Test System Interaction. The systems interact as depicted in Figure 3.7 for an active experiment and Figure 3.8 for a passive one. *honeyd* is initialized and prepared for receipt of traffic. When performing an active mapping experiment, the sniffer system is "tapped" into all network traffic by using a SPAN port on the 2950 switch which redirects incoming traffic on the remaining systems' switch ports to its switch port. The sniffer system runs a *tcpdump* session to capture all IP and ARP traffic. The mapping system gets a command-line prepared for the current experiment (see Figure C.1 in Appendix C for specifics). A *syntraf* session is started for the appropriate time period plus 30 seconds allowing sufficient time to start the mapping system. The experiment ends when the *nmap* session is complete. A "Ctrl-C" break stops the *syntraf* and *tcpdump* sessions and the *tcpdump* logs are gathered. Logs from the honeyd side are saved into their own directory.

When performing a passive experiment, there are differences in the system operation. *honeyd* is initialized first. Instead of the sniffer machine pm a SPAN port, however, the mapper system, *lanmap* is connected to the SPAN port. Therefore, the *tcpdump* session to keep track of packet counts must be executed on the map-

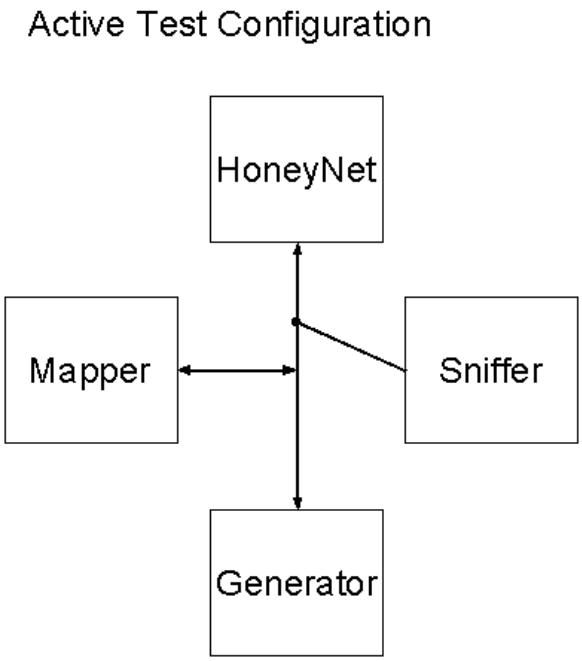

Figure 3.7: Active Test System Configuration

per. Since the Cisco 2950 switch is limited to one SPAN session, both *lanmap* and *tcpdump* need to run on the same system. The *collector* machine starts a *collector* session. *lanmap* starts on the mapper and redirects its network output to the *collector* machine. Another limitation of the Cisco 2950 switch is when using the SPAN port, no traffic may be input into the switch through that port. Therefore, the mapper requires two network interfaces to function. In this case, the mapper uses the control network to emit its packets to the *collector*. The *lanmap* program keeps track of the number of packets and their sizes sent to the *collector* as the *tcpdump* process cannot watch more than one network interface at a time. Finally, a *syntraf* session is begun on the traffic generator system. The experiment ends when *syntraf* ceases transmitting (at its timeout value). *lanmap* is stopped, then *collector*. *tcpdump* is stopped and finally *honeyd*. As before, logs from all processes are collected and saved.

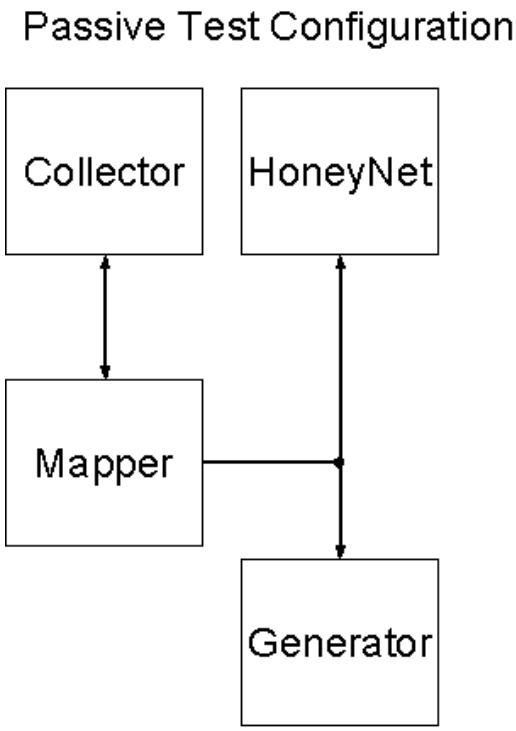

Figure 3.8: Passive Test System Configuration

3.9 Experimental Execution

A full factorial design with three repetitions results in a total of 2 (method) x 5 (time slots) x 2 (scenarios) x 3 (traffic patterns) x 3 (replications) = 180 trials for the all experiments. To perform a given experiment the script in Figure C.1 is followed.

3.10 Summary

This chapter has discussed the requirements of a notional passive network mapper, examined possible candidates to use as a passive mapper and the design of a custom passive mapper. Further, this chapter discussed the requirements of a network mapping test bed, taking both methods, active and passive, into account and construction of such a test bed. Next, it discussed the design of experiments, including system and workload parameters and the factors chosen for this research.

Finally, this chapter discussed the evaluation technique and overall interaction of all the systems, custom as well as off-the-shelf.

4. Analysis

4.1 Introduction

This chapter describes the execution of the experiments and the analysis of the data. Section 4.2 describes the method of data collection and the software packages used for generating statistical results, and the initial Analysis of Variance (ANOVA). Section 4.3 describes the comparison of the accuracy metrics. Section 4.4 discusses comparison of the efficiency metrics. Section 4.5 discusses overall findings. Finally, Section 4.6 discusses limitations of the experiments and methods chosen.

4.2 Collection of Data and Analysis of Variance

As described in Chapter 3, all the experiments are performed, in an identical fashion, varying the factors as appropriate to the given trial. Results are collected using the *Perl* scripts to normalize the output log files of *lanmap* and *nmap* and then read by the *matchem* program to determine their accuracy. These results are initially entered into an *Excel*® spreadsheet (see Appendix A). Later, the statistical software package *JMP*® is used to gather the information for better statistical analysis.

A quick examination of the response variables indicate a large range between the minimum and maximum value of each response variable. According to Jain [36], because the ratio of y_{max}/y_{min} is large (where y is the appropriate response variable), a logarithmic transformation is useful. Therefore, for the remainder of this analysis, all of the response variables are log (base 10) transformed; any zero results in the original data, remains a zero result in the log transformed data.

Also note that during the testing there were **no** false positives generated by either mapping method, probably due to the relatively pristine configuration of the network. This response variable is ignored for the remainder of this examination.

4.2.1 Eliminating Traffic Pattern Factor. Up to and including the five minute time category, the Traffic Pattern appeared to have some effect on the outcome of the tests. However, THE variance chart on "Log Total Discoveries" and the ANOVA as seen in Figure 4.1, indicate otherwise. The results show that the Traffic Pattern has virtually no effect on "Log Total Discoveries." So, in an effort to remove Traffic Pattern as a factor, confidence interval tests on all of the gathered response variables were conducted.

Component	Var Component	% of Total	Sqrt(Var Comp)
Time Given	0.1382511	11.0	0.3718
Scenario	0.0048139	0.4	0.0694
Time Given*Scenario	0.0002533	0.0	0.0159
Method	0.7633438	60.8	0.8737
Time Given*Method	0.2837773	22.6	0.5327
Scenario*Method	0.0339431	2.7	0.1842
Time Given*Scenario*Method	0.0311766	2.5	0.1766
Traffic Pattern	0.0000000	0.0	0.0000
Time Given*Traffic Pattern	0.0000661	0.0	0.0081
Scenario*Traffic Pattern	0.0000009	0.0	0.0010
Time Given*Scenario*Traffic Pattern	0.0000000	0.0	0.0000
Method*Traffic Pattern	0.0001454	0.0	0.0121
Time Given*Method*Traffic Pattern	0.0002645	0.0	0.0163
Scenario*Method*Traffic Pattern	0.0000000	0.0	0.0000
Time Given*Scenario*Method*Traffic Pattern	0.0000000	0.0	0.0000
Within	0.0001003	0.0	0.0100
Total	1.2561364	100.0	1.1208

Negative Variance Components were set to zero

Figure 4.1: Overall Analysis of Variance of the Effects for "Total Discoveries"

Generating confidence intervals (CIs) for the Server and Client Discoveries response variables produces Table 4.1 and Table B.1 in Appendix B, respectively. As these results show in both client and server cases, all three means are contained in the other confidence intervals. This means that they are statistically the same and can be ignored with respect to the accuracy metrics.

The results in Table B.3 and Table B.4, both in Section B.1 of Appendix B, show the confidence intervals of "Packets In" and "Packets Out" metrics. These results also show that the mean and standard deviation estimators for all three Traffic Patterns are statistically equivalent given a 95% confidence interval. Therefore, the

Table 4.1: 95% CI on Server Discoveries by Log Traffic Pattern

Estimator	Pattern	Estimate	Lower CI	Upper CI
Mean	1	1.264	1.100	1.429
	2	1.271	1.106	1.435
	4	1.257	1.094	1.421
Std Dev	1	0.636	0.539	0.779
	2	0.636	0.539	0.775
	4	0.634	0.537	0.773

Traffic Pattern factor can be ignored as it does not result in any significant variation in the response variables.

This is not a surprising outcome, however. When run over a short amount of time, it is quite possible for the traffic generator not to use given services and/or client combinations and produce varied results. But when the random number generators are given long enough (apparently between five and ten minutes worth of events), the traffic generator will have used all possible combinations within its configuration, obviating any differences due to the Traffic Pattern.

4.2.2 Examining Factor Effects on Metrics. With the Traffic Pattern factor eliminated, the effects of each of the factors in each of the response variables are tested. In Table 4.2, the bold highlighted probabilities shows those factors which are statistically significant to the outcome of the "Client Discoveries" metric.

Table 4.2: Effects Test for Log Client Discoveries ($\alpha = 0.05$)

Source	Nparm	DF	SS	F Ratio	Prob > F
Method	1	1	252.170	760001.357	**4.53E-296**
Time Given*Method	4	4	1.146	863.325	**3.86E-107**
Time Given	4	4	1.146	863.325	**3.86E-107**
Scenario*Method	1	1	0.006	18.945	**2.39E-005**
Scenario	1	1	0.006	18.946	**2.39E-005**
Time Given*Scenario	4	4	0.003	2.517	**4.35E-002**
Time Given*Scenario*Method	4	4	0.003	2.517	**4.35E-002**

The statistics show that all of the first level, second-level and the one third-level effects are statistically significant to the outcome of the Client Discoveries response variable. However, the statistics also show that the first level Method factor has the strongest affect on the outcome, with the first level Time Given and the second level Time Given*Method factor having an equal secondary effect.

Interestingly, when testing the Server Discoveries response variable as in Table 4.3, the Time Given factor provides slightly more effect than the Time Given*Method second-level factor. The first-level Method factor has the third-highest effect. While the Method factor does not dominate here, it is a contributing factor. Obviously, the number of Server Discoveries is sensitive to time.

Table 4.3: Effects Test for Log Server Discoveries ($\alpha = 0.05$)

Source	Nparm	DF	SS	F Ratio	Prob > F
Time Given	4	4	32.334	23486.751	**2.23E-220**
Time Given*Method	4	4	28.895	20988.414	**1.77E-216**
Method	1	1	3.829	11124.780	**8.56E-150**
Scenario	1	1	2.188	6357.945	**1.01E-130**
Scenario*Method	1	1	1.860	5405.586	**3.12E-125**
Time Given*Scenario	4	4	1.106	803.682	**9.08E-105**
Time Given*Scenario*Method	4	4	1.157	840.224	**3.06E-010**

When considering the "Packets In" response variable, different mix of effects were significant as shown in Table 4.4. While the outcome is still dominated by the Method factor, the second-highest effect is the first-level Scenario factor. This is due to the solely to nature of the *nmap* active mapper. The Scenario factor varies the type of server environment between a firewalled suite to a standard suite. When *nmap* probes a firewalled host and receives no response in a given time frame, it repeats that probe up to three more times. If no answer is received, that server/port combination is marked as "filtered" because it cannot mark it open (no SYN|ACK response) and cannot mark it as closed (no RST|ACK response). So, during the firewall scenario, the mapper does not receive packets back in response to a non-existent service. On the other hand, when mapping during the normal scenario, **all**

the non-existent probes return the RST|ACK. So dependent upon how many ports are scanned, *nmap* gets as much as it gives.

Table 4.4: Effects Test for Log Packets In ($\alpha = 0.05$)

Source	Nparm	DF	SS	F Ratio	Prob > F
Method	1	1	145.536	1438839.450	**0**
Scenario	1	1	65.585	648407.756	**1.49E-290**
Scenario*Method	1	1	60.915	602238.296	**5.47E-288**
Time Given	4	4	48.103	118892.467	**1.12E-276**
Time Given*Method	4	4	4.617	11410.679	**2.32E-195**
Time Given*Scenario	4	4	1.382	3415.676	**9.68E-154**
Time Given*Scenario*Method	4	4	1.404	3469.597	**2.80E-154**

The test for Packets Out is similar to the Client Discoveries effects and can be found in Table B.8 in Section B.2 of Appendix B. It is not duplicated here.

As can be seen in three of the four response variables, the Method factor is the primary effect. Clearly, the Method factor, either active or passive, has a great deal of effect in the outcome of these three variables, and is in two of the top three effects of the final response variable. Looking at the derivative metrics may provide more information about these effects.

4.2.3 Examining the Derivative Metrics. Continuing the exploration of the effects of the response variables, the "Total Discoveries" metric, Table 4.5 shows the effects of the two accuracy outcomes (Client Discoveries and Server Discoveries) in aggregate. Again, the Method factor is the strongest, followed by the Time Given and second-level Time Given*Method effects. This is not too surprising due to the fact that the number of Client Discoveries is driven primarily by the Method factor and has a larger outcome. On average, there are approximately 124 clients discovered relative to the 38 servers discovered. Since the Total Discoveries metric is a simple sum of the Server and Client metrics, it stands to reason that the Client Discoveries and its effects, would drive the Total Discoveries metric to a great extent.

Table 4.5: Effects Test for Log Total Discoveries ($\alpha = 0.05$)

Source	Nparm	DF	SS	F Ratio	Prob > F
Method	1	1	75.623	185190.932	**4.89E-247**
Time Given	4	4	41.490	25401.171	**4.26E-223**
Time Given*Method	4	4	21.561	13199.974	**2.09E-200**
Scenario	1	1	2.246	5499.876	**8.14E-126**
Scenario*Method	1	1	1.808	4427.709	**1.62E-118**
Time Given*Scenario	4	4	1.141	698.388	**3.86E-100**
Time Given*Scenario*Method	4	4	1.123	687.270	**1.30E-099**

Examining the effects with respect to the Overhead Packets In metric, again the Method factor has the greatest amount of effect on the outcome. See Table B.10 for details. Remember as discussed in Chapter 3, due to the nature of the mappers, any packet outbound of the mapper system would be over-and-above the normal traffic transiting the network. Therefore, the Packets Out metric is functionally equivalent to an Overhead Packets Out metric and is not discussed further.

Finally, in Table B.11, the Method factor once again dominates the outcome of this response variable. Not surprising since the two values feeding this aggregate metric were dominated by the Metric factor.

Seeing as how three out of four primary response variables are dominated by the Method factor and all of the derived response variables are dominated by the Method factor, it appears that the Method factor is the primary factor driving the response of the system.

4.3 Comparison of Accuracy Metrics

4.3.1 Overall Accuracy. Next, we evaluate the 95% confidence intervals of the accuracy response variables (Server, Client and Total Discoveries) and from the two Method levels of active and passive. Then compare the CI's to see if the CI's capture the other response variable's mean or standard deviation estimator. Since the overall data is skewed due to the time factor, this comparison focuses on the

40-minute Time Given factor, where the discovery results are maximized for both methods.

Table 4.6 shows this comparison for Server Discoveries. Since neither the confidence interval of the means nor the standard deviation captures the estimators, they are statistically different, and, in this case the active mapper discovered more information about the network than the passive mapper did.

Table 4.6: 95% CI on Log Server Discoveries by Method (Time=40)

Estimator	Method	Estimate	Lower CI	Upper CI
Mean	**Active**	2.101	2.097	2.104
	Passive	1.431	1.431	1.431
Std Dev	Active	0.007	0.006	0.011
	Passive	0	0	0

Due to the method the passive mapper uses, it would not see any traffic for a service that goes unused. In this case, most servers were not fully exercised by the clients. If the clients had been configured (rather artificially) to connect to every server on every port, the passive mapper **would** have seen it and these results would be more similar. As it stands, however, the active mapper is the one which exercises more ports–1656 in this case–on all of the systems it can discover. If the passive mapper were listening during the active mapper's scan, its server discoveries metric would be much higher.

With respect to the Client Discoveries, however, the active mapper found nothing. Table 4.7 shows that the passive mapper is superior. As clients do not *provide* services but rather *consume* them, the active mapper cannot discover them. Indeed, active mapper systems are designed to find open service-providers, not the clients who use them.

These two comparisons don't provide a clear-cut winner, so it becomes necessary to compare the Total Discoveries to determine best overall performance with respect to accuracy. Generating this confidence interval as shown in Table 4.8 dis-

Table 4.7: 95% CI on Log Client Discoveries by Method (Time=40)

Estimator	Method	Estimate	Lower CI	Upper CI
Mean	Active	0	0	0
	Passive	2.507	2.507	2.507
Std Dev	Active	0	0	0
	Passive	0	0	0

closes that the passive mapper is the better of the two with regard to overall number of discoveries. Indeed, since there are typically more users (and, therefore, clients) using a network than servers, the passive mapper discovers more about a given network while watching transactions cross its path.

Table 4.8: 95% CI on Log Total Discoveries by Method (Time=40)

Estimator	Method	Estimate	Lower CI	Upper CI
Mean	Active	2.101	2.097	2.104
	Passive	2.542	2.542	2.542
Std Dev	Active	0.007	0.006	0.011
	Passive	0	0	0

4.3.2 Accuracy over Time. Part of this examination looks at performance of the methods over time. Figure 4.2 shows the mean values plotted over time for the respective mapping method. The overall effect of the graphic is evident: the passive method is the overall winner.

Interestingly, the active mapper found no clients but it did discover more servers than the passive mapper, as mentioned above.

4.4 Comparison of Efficiency Metrics

4.4.1 Overall Efficiency. A similar examination of the efficiency metrics, specifically looking at the "overhead" metrics shows similar results. Remember that the overhead factors are describing extra traffic on the network and are considered detrimental to the function of the mapper. In fact, both the Overhead Packets Out

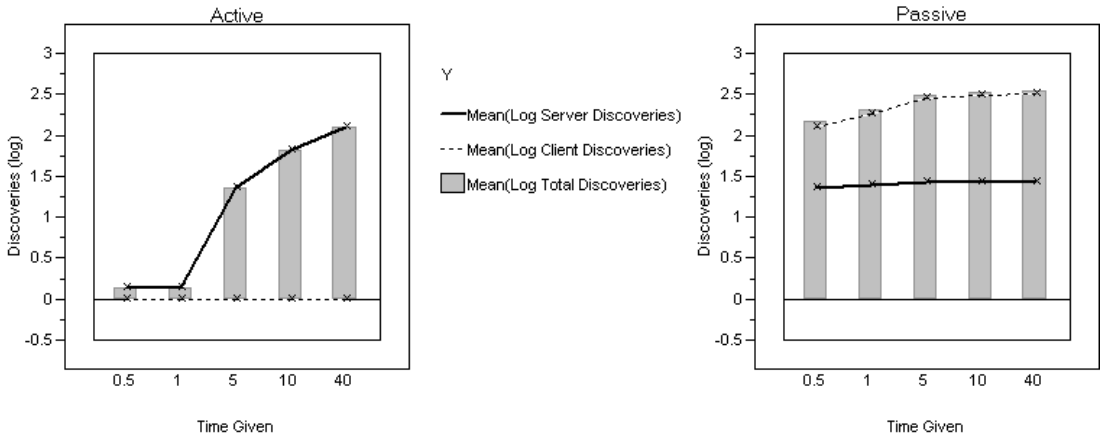

Figure 4.2: Discoveries over Time (higher is better)

and Total Network Overhead show that the passive mapper is superior. Table 4.9 demonstrates this.

Table 4.9: 95% CI on Total Network Overhead (Log # Packets)

Estimator	Method	Estimate	Lower CI	Upper CI
Mean	Active	4.523	4.413	4.634
	Passive	2.082	2.030	2.134
Std Dev	Active	0.527	0.459	0.617
	Passive	0.249	0.217	0.291

The active mapper generates many packets and also stimulates other network devices to generate packets in response. The passive mapper does its job with fewer overhead packets.

4.4.2 Efficiency over Time. Figure 4.3 shows the mean values plotted over time for the respective mapping method. Note that the Time Given (the x-axis scale) is categorical rather than a true time scale, due to the time periods chosen. However the passive method is clearly more efficient.

4.5 Summary

honeyd and *syntraf* were configured with 136 total server-side services and 321 client-side service consumers for a total of 457 discoverable services.

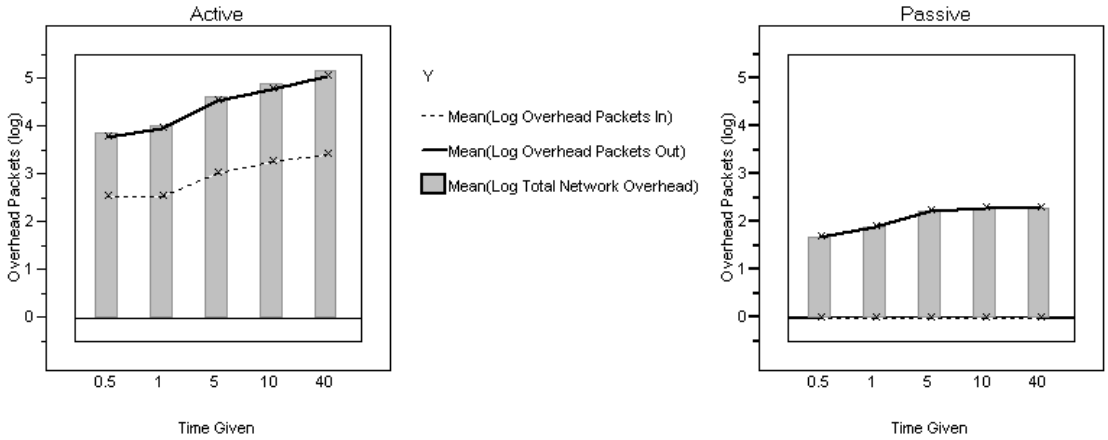

Figure 4.3: Network Overhead over Time (lower is better)

lanmap is more accurate than *nmap*. It discovers at its peak, an average of 76.1% of all configured services (server- and client-side) whereas *nmap* only found 27.6% of those same services. When considering only the server-side services, *lanmap* discovered only 19.9% of those services while *nmap* discovered 92.7% of the configured server-side services. Clearly, *nmap* is better at finding unused server-side services. With respect to client-side service consumers, *lanmap* discovered 100% of all those services while *nmap* found none. Since most networks contain considerably more clients than servers, finding the client services in-use provides for more network discoveries.

When considering the efficiency of each system, *lanmap* generated an average of 200 packets of network overhead at its peak network utilization. *nmap*, on the other hand, generated over 8,600 packets on average at its ***minimum*** utilization and up to 155,000 packets at its ***maximum*** average value.

4.6 Limitations

4.6.1 UDP. The server-side UDP echo server implemented in honeyd would not reliably react to incoming packets and would block, awaiting input even when input was made available. It was determined that there were differences with

how honeyd and the supporting scripts for the service differ with respect to data formats and/or some unknown conversion was occurring. Network traces did not reveal any obvious problems. Packets were arriving at the honeyd system but the script was not properly handling the input.

In order to "simulate" those UDP services, the configuration files were changed to put those services which are normally UDP at the same port with a TCP configuration. A side-effect of this is that the active mapper now had more data. In its default configuration, *nmap* does not search for UDP ports–those services would have been invisible to *nmap* if left on UDP.

4.6.2 syntraf.

4.6.2.1 Thread Execution Order. Each thread in *syntraf* has its own random number generator and is seeded by the main program, which also has its own random number generator. Thus, each thread gets a different seed and therefore its own order of traffic generation. However, there is still a certain amount of randomness due to the uncertainty with respect to the execution order of these threads. Given enough repetition, however, this problem did not seem to have much of an effect.

4.6.2.2 Similarity to Real Traffic. *syntraf* was developed as a stable platform to generate traffic for the passive mapper to analyze. However, due to its use of the uniform random number generator built-in to the Java libraries, it does not address the self-similarity of users in a real system. It could be improved to provide this functionality, but it was not the object to build something wholly complex. *LARIAT* exists for that purpose [46]. The needs of this research does not warrant the complexity of a full-fledged network simulator.

4.6.2.3 Client-side Active Mapping. *nmap* can and did discover client computers. However, due to the vagaries of the network configuration, *nmap* actually

mapped the host system's open ports. Since these were outside the control of the experiment, these results were discarded.

The result is that the clients that *syntraf* provides do not have any open services and do not look "normal." Instead, they are completely firewalled. They do not respond in any way to any probes. This was not an oversight; the complexity of creating a two-sided honeynet-like system, which provided services on one side and simulated client machines on the other, injecting traffic into both systems was beyond the scope of this research.

4.6.3 Number of SPAN Sessions. The Cisco Catalyst 2950 switch is only capable of having one SPAN session–that is, there can only be one port chosen to host a sniffer. Because of this, the passive mapper had to essentially run *two* sniffers on it–*lanmap* and a tcpdump to capture the traffic seen by *lanmap*. While there did not appear to be any problem with the systems used during this investigation (no packets were dropped due to congestion on any of the experimental trials), it is obvious that a more loaded network would easily swamp the two sniffers.

4.6.4 nmap.

4.6.4.1 Mapping of the Test-bed Hosts During the Session. *nmap*'s discovery capability was complete. In fact, it mapped the host computers hosting the test-bed. Due to the nature of the configuration, each of the hosts test network connection had an IP address assigned, in a higher range to keep them "out of the way." When *nmap* was given enough time, however, it found these hosts and mapped them. These results were discarded and the experiments redone with those IP address ranges for the host machines excluded from the search.

4.6.4.2 Run Times. Both systems, *lanmap* and *nmap*, were given specific time frames in which they to discover what they could about the network. *lanmap* was designed as a run-forever program that reports its results until it is

terminated. On the other hand, *nmap* systematically probes target networks for services. Once the list of networks is exhausted, *nmap* is done. In an average of 8.24 minutes, *nmap* would complete its scan of the given network in the "normal" scenario. The "firewall" scenario quadrupled that completion time to an average of 37.6 minutes. This extra time is caused by *nmap*'s extra attempts to find a service at a non-responding port. *nmap* gives up after four attempts on a given port. Recall that the normal scenario's servers would respond with a RST message instead of simply dropping the packet and *nmap* would know that the given port is closed.

Another consequence of the full *nmap* scan finishing in under 10 minutes during the Active / Normal scenario, the *40 minute* trials would be meaningless. So, one trial using the 40 minute settings is performed once to ensure that it would still complete within the 10 minute time frame. Since it does, that result and results from the 10 minute trials are averaged and placed into the data tables instead of performing eight more nearly identical trials.

4.6.5 Real Systems.

4.6.5.1 Total Knowledge. It is certainly rare to have total knowledge and control over the systems contained within a network. Indeed, with user systems turned on and off throughout a day, random network outages severing ties to various systems (server and client computers alike), and the day-to-day changes of a network are the primary responsibilities of network management personnel. Having full knowledge of this test bed network enabled right-or-wrong answers about the existence of systems and their services. In a real network, that information is not available or worse, not accurate. When using these tools against an adversarial system where little to nothing is known about the configurations, it is unlikely that either mapping system will know for a fact if it has made a correct answer about a given service on a given machine. Using the test bed simplified this and made the comparisons more accurate.

4.6.5.2 Performance. The test network was built for testing a small subset of a real network. In a real system, with a real, loaded network, this system would be swamped.

After the experimental trials were completed, the AFIT network administrators allowed access to the real network to try *lanmap*'s capabilities on real traffic. *lanmap* worked and was able to discover hosts, however, upon completion of a run, *lanmap* said it had dropped roughly one-half of the available packets. This was due to the operating system's inability to handle the offered load. The operating system's packet filter is responsible for providing PCAP with packets based on PCAP's filter. However, due to the rate at which these packets arrive, the operating system drops arriving packets destined for PCAP to keep from overrunning its internal buffers. When a PCAP function is called to fetch packets from the operating system, the operating system tells PCAP how many packets it received, and how many were dropped. *lanmap* and other PCAP-based programs can determine how much traffic is lost between calls to the PCAP routines.

A real system would need to have one or more of:

1. a larger packet buffer in the operating system space,
2. a more robust processor, and
3. statistical methods such as those discussed by Claffy [21, 22] for sampling the data crossing the network.

4.7 Summary

In this chapter, the performance of *lanmap* and *nmap* were compared in the environment specified in Chapter 3. *lanmap* turned out to work better than nmap in almost every category, except for the discovery of server-side services. *nmap* found 92.7% of the offered services while *lanmap* only discovered those the 19.9% used by clients. Limitations to this research and its experiments were discussed.

5. Conclusions

5.1 Research Contribution

The focus of this research is to empirically compare passive and active network mapping techniques. This research does that and expands the knowledge base for network mapping.

This research also created a new mapping utility which provides a method to map a network passively. An existing algorithm from *ettercap*'s dryad module was simplified and enhanced. Simultaneously, adding a simple UDP transmission allows the system to get the mapping information out of the network being probed. This research also provided a central collection system where data can be aggregated.

Finally, this research provides a method for generating traffic in a realistic environment. Even though a single traffic generator is used during the experiments, there is nothing to keep multiple generators and multiple honeynets from being used as a traffic-based test bed.

5.2 Performance of the Mapping Methods

The data presented in Chapter 4 shows that passive mapping discovers more about the network than the active mapper and does so with less overhead on the network. However, the active mapper excelled at discovering offered but unused services. Indeed, it discovers all offered services, no matter the scenario. When mapping a network, you need as much data as possible to represent the network as accurately as possible.

The traffic generator patterns had little effect on the output. The mapping method used typically had the most effect with the time factor providing an additional variance.

Both methods perform better when given more time to run. In both cases, the amount of new information plateaus and after a certain amount of time, no new information is gathered. Indeed, *nmap* actually finishes its scan while *lanmap* continues to report new items. *lanmap* discovers more services initially and discovers fewer and fewer new items as time goes on.

5.3 Conclusions

Accurate data is necessary for generating a network map. Both active and passive mapping methods provide the information to generate a relatively accurate network map. In the case of *nmap*, only the server-side services are discovered and not the clients which used them. On the other hand, *lanmap* captured more and varied information about the network, specifically services offered and consumed, but was unable to detect those unused services that *nmap* discovered. Neither tool discovered the entire network, but *lanmap* discovered more.

nmap creates much more overhead network traffic than *lanmap*. Conversely, *lanmap* is more difficult to implement in a real environment. Setting up network switches with SPAN ports or using unswitched hubs at network choke points requires extra work and maintenance on the part of the network administrator. *nmap* needs nothing more than a network connection and an IP address. *lanmap* does not need the IP address, but requires more network support to function.

If a network administrator requires a method to map a network without adding traffic to the network, a passive mapping system such as *lanmap* would do well as long as the network management overhead was tolerable. Passive mapping would find more server- and client-side services being provided and used than active mapping. If, however, it is more important to find all the network services being provided, *nmap* or another active scanner would be more appropriate.

5.4 Future Research

5.4.1 lanmap and collector. The *lanmap* program operates nearly as designed. However, the delayed, random output needs to be implemented. The desire to buffer output to maintain a certain level of stealth failed, at least during the first minute of running. First and foremost, *lanmap* needs to discover the IP range of the given network so that it can determine the difference between inside traffic (traffic only traversing the "internal" network) and inside-outside traffic (traffic originating or destined for external networks).

Finally, the ability of the system to leverage information from other network devices is needed to complete the tool. In a well-maintained network, an exiting packet would need to (appear to) be sent from an authorized system. *lanmap* can watch the network and determine which hosts are communicating with systems outside of the known network. When a host successfully communicates with an outside source, *lanmap* can send its information to the *collector* outside of the local network. The *collector* program should be able to draw a network diagram to aid a network administrator.

5.4.2 Flow- versus Sniffer-Based System. *lanmap* uses a sniffer-based approach in discovering information about the network. One consideration at the beginning of this research was to use NetFlow or similar data to discover the network [10]. It would be interesting to see the results of a similar comparison between active, passive-sniffer and passive-flow methods of information gathering.

5.4.3 syntraf. *syntraf* might be a useful tool at a research institution. However, it needs to be more robust and provide more traffic generation options such as Poisson- and exponentially-distributed data. Further research could make this tool into a full-fledged system.

5.4.4 Network Loading. The systems used for this research were modest as were the network loads. This was by design to test if a passive mapper was a viable solution. However, if the passive mapper is incorporated into operational network systems, it would encounter real network loading and could be inundated with network traffic. If the system running *lanmap* (or similar tool) is not adequate to the task, vital information may be lost. Further research could determine what the bounds for a sniffer-based mapping system on given system platforms and what system improvements would be needed to get there [21, 22].

5.4.5 ARP Poisoning. Using ARP poisoning techniques, it would be possible for the passive mapper to see traffic without the need to configure a SPAN (or equivalent) port on a network switch. However, once the mapper begins to receive that traffic, it suddenly becomes an active member of the network, receiving and forwarding packets. This technique was not used as it does not meet the definition of a purely passive mapper, but could be implemented as a follow-on project. Integrating ARP poisoning into this system might negate the need for control over the network infrastructure.

5.4.6 Combining Active and Passive Techniques. Since using either technique by itself did not discover more than 72.6% of the total number of configured items, it might be interesting to see how the two techniques would fair if used together. An active probe could elicit response from an quiet service-provider while the passive sensor would make note of the result.

Appendix A. Gathered Data

This appendix contains the full data-set gathered during the experimentation phase of this investigation.

Scenario	Method	Traffic Pattern	Time Given	Time Used	Server Disc.	Client Disc.	Packets In	Packets Out
Firewall	Active	1	0.5	0.5	1	0	37	4965
Firewall	Active	1	0.5	0.5	0	0	35	4964
Firewall	Active	1	0.5	0.5	0	0	35	4998
Firewall	Active	2	0.5	0.5	0	0	36	4342
Firewall	Active	2	0.5	0.5	1	0	36	4999
Firewall	Active	2	0.5	0.5	0	0	35	4998
Firewall	Active	4	0.5	0.5	0	0	35	4998
Firewall	Active	4	0.5	0.5	0	0	35	4998
Firewall	Active	4	0.5	0.5	0	0	35	4968
Firewall	Active	1	1	1	1	0	37	7741
Firewall	Active	1	1	1	1	0	37	7735
Firewall	Active	1	1	1	1	0	37	7741
Firewall	Active	2	1	1	1	0	37	7741
Firewall	Active	2	1	1	1	0	37	7735
Firewall	Active	2	1	1	1	0	37	7741
Firewall	Active	4	1	1	1	0	37	7741
Firewall	Active	4	1	1	1	0	37	7741
Firewall	Active	4	1	1	1	0	36	7743
Firewall	Active	1	5	5	8	0	46	29944
Firewall	Active	1	5	5	6	0	52	26792
Firewall	Active	1	5	5	8	0	45	29866
Firewall	Active	2	5	5	8	0	45	29920
Firewall	Active	2	5	5	8	0	45	29956
Firewall	Active	2	5	5	8	0	45	29956
Firewall	Active	4	5	5	8	0	45	29922
Firewall	Active	4	5	5	8	0	47	29112

Scenario	Method	Traffic Pattern	Time Given	Time Used	Server Disc.	Client Disc.	Packets In	Packets Out
Firewall	Active	4	5	5	8	0	46	26886
Firewall	Active	1	10	10	35	0	73	57707
Firewall	Active	1	10	10	35	0	73	55445
Firewall	Active	1	10	10	35	0	72	57796
Firewall	Active	2	10	10	35	0	75	57531
Firewall	Active	2	10	10	35	0	74	53533
Firewall	Active	2	10	10	36	0	73	57761
Firewall	Active	4	10	10	35	0	73	55605
Firewall	Active	4	10	10	35	0	73	56748
Firewall	Active	4	10	10	35	0	73	55699
Firewall	Active	1	40	36.85	126	0	167	197795
Firewall	Active	1	40	40	126	0	169	194561
Firewall	Active	1	40	35.991	126	0	165	197735
Firewall	Active	2	40	37.078	118	0	157	183137
Firewall	Active	2	40	35.679	126	0	165	197805
Firewall	Active	2	40	40	126	0	165	197737
Firewall	Active	4	40	36.543	126	0	167	197741
Firewall	Active	4	40	40	126	0	165	194155
Firewall	Active	4	40	36.169	126	0	164	197792
Firewall	Passive	1	0.5	0.5	23	131	7579	48
Firewall	Passive	1	0.5	0.5	23	131	7341	56
Firewall	Passive	1	0.5	0.5	23	131	7053	49
Firewall	Passive	2	0.5	0.5	24	124	7174	55
Firewall	Passive	2	0.5	0.5	24	124	7241	50
Firewall	Passive	2	0.5	0.5	24	124	7197	54
Firewall	Passive	4	0.5	0.5	19	105	7030	39
Firewall	Passive	4	0.5	0.5	19	105	6930	40
Firewall	Passive	4	0.5	0.5	19	105	6821	39
Firewall	Passive	1	1	1	23	185	14936	77
Firewall	Passive	1	1	1	23	184	13907	77
Firewall	Passive	1	1	1	23	188	14064	84

Scenario	Method	Traffic Pattern	Time Given	Time Used	Server Disc.	Client Disc.	Packets In	Packets Out
Firewall	Passive	2	1	1	26	187	13665	86
Firewall	Passive	2	1	1	26	186	13759	85
Firewall	Passive	2	1	1	26	186	14202	78
Firewall	Passive	4	1	1	23	159	13933	67
Firewall	Passive	4	1	1	24	159	13517	70
Firewall	Passive	4	1	1	24	159	13236	61
Firewall	Passive	1	5	5	27	281	68260	163
Firewall	Passive	1	5	5	27	282	67212	160
Firewall	Passive	1	5	5	27	282	67647	159
Firewall	Passive	2	5	5	27	289	65951	174
Firewall	Passive	2	5	5	27	289	65619	166
Firewall	Passive	2	5	5	27	289	65668	176
Firewall	Passive	4	5	5	26	284	68178	170
Firewall	Passive	4	5	5	26	284	66228	159
Firewall	Passive	4	5	5	26	284	65559	168
Firewall	Passive	1	10	10	27	311	134147	199
Firewall	Passive	1	10	10	27	311	132789	187
Firewall	Passive	1	10	10	27	311	132935	185
Firewall	Passive	2	10	10	27	314	130808	181
Firewall	Passive	2	10	10	27	314	132477	190
Firewall	Passive	2	10	10	27	314	132657	187
Firewall	Passive	4	10	10	26	306	134333	191
Firewall	Passive	4	10	10	26	306	134028	198
Firewall	Passive	4	10	10	26	306	134312	191
Firewall	Passive	1	40	40	27	321	536814	188
Firewall	Passive	1	40	40	27	321	530565	192
Firewall	Passive	1	40	40	27	321	531169	195
Firewall	Passive	2	40	40	27	321	530922	201
Firewall	Passive	2	40	40	27	321	531105	196
Firewall	Passive	2	40	40	27	321	533250	201
Firewall	Passive	4	40	40	27	321	533341	206

Scenario	Method	Traffic Pattern	Time Given	Time Used	Server Disc.	Client Disc.	Packets In	Packets Out
Firewall	Passive	4	40	40	27	321	530624	212
Firewall	Passive	4	40	40	27	321	529295	202
Normal	Active	1	0.5	0.5	2	0	3374	8252
Normal	Active	1	0.5	0.5	2	0	3239	7946
Normal	Active	1	0.5	0.5	2	0	3374	8162
Normal	Active	2	0.5	0.5	2	0	3374	8192
Normal	Active	2	0.5	0.5	2	0	3043	7913
Normal	Active	2	0.5	0.5	2	0	3459	8210
Normal	Active	4	0.5	0.5	2	0	3373	8192
Normal	Active	4	0.5	0.5	2	0	3332	8201
Normal	Active	4	0.5	0.5	2	0	3374	8156
Normal	Active	1	1	1	2	0	3104	9673
Normal	Active	1	1	1	2	0	3390	11576
Normal	Active	1	1	1	2	0	3374	10966
Normal	Active	2	1	1	2	0	3374	10996
Normal	Active	2	1	1	2	0	3292	10759
Normal	Active	2	1	1	2	0	3373	10966
Normal	Active	4	1	1	2	0	3372	10996
Normal	Active	4	1	1	2	0	3373	10995
Normal	Active	4	1	1	2	0	3373	10966
Normal	Active	1	5	5	67	0	22227	40051
Normal	Active	1	5	5	67	0	22354	40240
Normal	Active	1	5	5	68	0	22928	40810
Normal	Active	2	5	5	70	0	23432	41379
Normal	Active	2	5	5	72	0	24526	42427
Normal	Active	2	5	5	70	0	23674	41577
Normal	Active	4	5	5	74	0	25053	42962
Normal	Active	4	5	5	69	0	23925	41827
Normal	Active	4	5	5	68	0	22816	40688
Normal	Active	1	10	8.32	127	0	44798	69706
Normal	Active	1	10	8.282	127	0	44802	69746

Scenario	Method	Traffic Pattern	Time Given	Time Used	Server Disc.	Client Disc.	Packets In	Packets Out
Normal	Active	1	10	8.17	127	0	44659	69564
Normal	Active	2	10	8.375	127	0	44800	69723
Normal	Active	2	10	8.118	127	0	44801	69671
Normal	Active	2	10	8.213	127	0	44494	69446
Normal	Active	4	10	8.175	127	0	44829	69711
Normal	Active	4	10	8.25	127	0	44800	69671
Normal	Active	4	10	8.234	127	0	44801	69725
Normal	Active	1	40	8.237	127	0	44753.78	69662.56
Normal	Active	1	40	8.237	127	0	44753.78	69662.56
Normal	Active	1	40	8.237	127	0	44753.78	69662.56
Normal	Active	2	40	8.237	127	0	44753.78	69662.56
Normal	Active	2	40	8.237	127	0	44753.78	69662.56
Normal	Active	2	40	8.237	127	0	44753.78	69662.56
Normal	Active	4	40	8.237	127	0	44753.78	69662.56
Normal	Active	4	40	8.237	127	0	44753.78	69662.56
Normal	Active	4	40	8.237	127	0	44753.78	69662.56
Normal	Passive	1	0.5	0.5	26	149	7955	58
Normal	Passive	1	0.5	0.5	26	147	8086	51
Normal	Passive	1	0.5	0.5	26	149	7929	53
Normal	Passive	2	0.5	0.5	25	137	7880	56
Normal	Passive	2	0.5	0.5	25	138	7939	53
Normal	Passive	2	0.5	0.5	25	138	7888	56
Normal	Passive	4	0.5	0.5	21	115	7912	44
Normal	Passive	4	0.5	0.5	21	115	8031	44
Normal	Passive	4	0.5	0.5	21	115	7864	49
Normal	Passive	1	1	1	26	202	15878	84
Normal	Passive	1	1	1	26	202	15965	80
Normal	Passive	1	1	1	26	205	15935	88
Normal	Passive	2	1	1	27	203	14894	85
Normal	Passive	2	1	1	27	203	14941	87
Normal	Passive	2	1	1	27	204	15627	92

Scenario	Method	Traffic Pattern	Time Given	Time Used	Server Disc.	Client Disc.	Packets In	Packets Out
Normal	Passive	4	1	1	26	175	15581	72
Normal	Passive	4	1	1	26	174	15138	73
Normal	Passive	4	1	1	26	175	15189	75
Normal	Passive	1	5	5	27	294	72805	173
Normal	Passive	1	5	5	27	295	75351	175
Normal	Passive	1	5	5	27	295	74870	169
Normal	Passive	2	5	5	27	304	73626	178
Normal	Passive	2	5	5	27	304	74756	184
Normal	Passive	2	5	5	27	304	74702	175
Normal	Passive	4	5	5	27	301	74707	182
Normal	Passive	4	5	5	27	300	72553	183
Normal	Passive	4	5	5	27	301	74574	185
Normal	Passive	1	10	10	27	317	146112	196
Normal	Passive	1	10	10	27	317	146206	192
Normal	Passive	1	10	10	27	317	146117	188
Normal	Passive	2	10	10	27	319	146154	198
Normal	Passive	2	10	10	27	319	146272	201
Normal	Passive	2	10	10	27	319	146855	204
Normal	Passive	4	10	10	27	318	145792	201
Normal	Passive	4	10	10	27	318	146897	205
Normal	Passive	4	10	10	27	318	144623	201
Normal	Passive	1	40	40	27	321	580291	200
Normal	Passive	1	40	40	27	321	583866	199
Normal	Passive	1	40	40	27	321	584611	200
Normal	Passive	2	40	40	27	321	586542	197
Normal	Passive	2	40	40	27	321	585933	193
Normal	Passive	2	40	40	27	321	582133	198
Normal	Passive	4	40	40	27	321	583057	199
Normal	Passive	4	40	40	27	321	584764	197
Normal	Passive	4	40	40	27	321	584585	208

Appendix B. Data Analysis Charts

B.1 Traffic Pattern Analysis

Table B.1: 95% CI on Log Client Discoveries by Traffic Pattern

Estimator	Pattern	Estimate	Lower CI	Upper CI
Mean	1	1.190	0.879	1.501
	2	1.188	0.878	1.499
	4	1.173	0.865	1.480
Std Dev	1	1.204	1.020	1.468
	2	1.203	1.020	1.468
	4	1.190	1.009	1.452

Table B.2: 95% CI on Log Server Discoveries by Traffic Pattern

Estimator	Pattern	Estimate	Lower CI	Upper CI
Mean	1	1.264	1.100	1.429
	2	1.271	1.106	1.435
	4	1.257	1.094	1.421
Std Dev	1	0.636	0.539	0.779
	2	0.636	0.539	0.775
	4	0.634	0.537	0.773

Table B.3: 95% CI on Log Packets In by Traffic Pattern

Estimator	Pattern	Estimate	Lower CI	Upper CI
Mean	1	3.859	3.508	4.210
	2	3.856	3.504	4.208
	4	3.856	3.504	4.208
Std Dev	1	1.359	1.152	1.658
	2	1.361	1.153	1.660
	4	1.361	1.154	1.660

Table B.4: 95% CI on Log Packets Out by Traffic Pattern

Estimator	Pattern	Estimate	Lower CI	Upper CI
Mean	1	3.259	2.937	3.581
	2	3.266	2.945	3.586
	4	3.249	2.923	3.575
Std Dev	1	1.247	1.057	1.521
	2	1.241	1.052	1.513
	4	1.263	1.071	1.540

B.2 Effects Tests for Response Variables

Table B.5: Effects Test for Log Client Discoveries ($\alpha = 0.05$)

Source	Nparm	DF	SS	F Ratio	Prob > F
Method	1	1	252.170	760001.357	**4.53E-296**
Time Given*Method	4	4	1.146	863.325	**3.86E-107**
Time Given	4	4	1.146	863.325	**3.86E-107**
Scenario*Method	1	1	0.006	18.945	**2.39E-005**
Scenario	1	1	0.006	18.946	**2.39E-005**
Time Given*Scenario	4	4	0.003	2.517	**4.35E-002**
Time Given*Scenario*Method	4	4	0.003	2.517	**4.35E-002**

Table B.6: Effects Test for Log Server Discoveries ($\alpha = 0.05$)

Source	Nparm	DF	SS	F Ratio	Prob > F
Time Given	4	4	32.334	23486.751	**2.23E-220**
Time Given*Method	4	4	28.895	20988.414	**1.77E-216**
Method	1	1	3.829	11124.780	**8.56E-150**
Scenario	1	1	2.188	6357.945	**1.01E-130**
Scenario*Method	1	1	1.860	5405.586	**3.12E-125**
Time Given*Scenario	4	4	1.106	803.682	**9.08E-105**
Time Given*Scenario*Method	4	4	1.157	840.224	**3.06E-010**

Table B.7: Effects Test for Log Packets In ($\alpha = 0.05$)

Source	Nparm	DF	SS	F Ratio	Prob > F
Method	1	1	145.536	1438839.450	**0**
Scenario	1	1	65.585	648407.756	**1.49E-290**
Scenario*Method	1	1	60.915	602238.296	**5.47E-288**
Time Given	4	4	48.103	118892.467	**1.12E-276**
Time Given*Method	4	4	4.617	11410.679	**2.32E-195**
Time Given*Scenario	4	4	1.382	3415.676	**9.68E-154**
Time Given*Scenario*Method	4	4	1.404	3469.597	**2.80E-154**

Table B.8: Effects Test for Log Packets Out ($\alpha = 0.05$)

Source	Nparm	DF	SS	F Ratio	Prob > F
Method	1	1	248.825	385675.206	**1.65E-272**
Time Given	4	4	23.336	9042.411	**2.60E-187**
Time Given*Method	4	4	3.065	1187.500	**8.66E-118**
Time Given*Scenario	4	4	0.738	285.820	**9.53E-072**
Time Given*Scenario*Method	4	4	0.600	232.681	**1.42E-065**
Scenario	1	1	0.0352	54.498	**8.03E-012**
Scenario*Method	1	1	0.001	1.436	2.33E-001

B.3 Effects Tests for Derived Metrics

Table B.9: Effects Test for Log Total Discoveries ($\alpha = 0.05$)

Source	Nparm	DF	SS	F Ratio	Prob > F
Method	1	1	75.623	185190.932	**4.89E-247**
Time Given	4	4	41.490	25401.171	**4.26E-223**
Time Given*Method	4	4	21.561	13199.974	**2.09E-200**
Scenario	1	1	2.246	5499.876	**8.14E-126**
Scenario*Method	1	1	1.808	4427.709	**1.62E-118**
Time Given*Scenario	4	4	1.141	698.388	**3.86E-100**
Time Given*Scenario*Method	4	4	1.123	687.270	**1.30E-099**

Table B.10: Effects Test for Log Overhead Packets In ($\alpha = 0.05$)

Source	Nparm	DF	SS	F Ratio	Prob > F
Method	1	1	393.702	5861006.280	**0**
Scenario	1	1	63.229	941280.880	**1.68E-303**
Scenario*Method	1	1	63.229	941280.880	**1.68E-303**
Time Given	4	4	6.020	22403.433	**9.67E-219**
Time Given*Method	4	4	6.017	22403.433	**9.67E-219**
Time Given*Scenario	4	4	1.393	5183.557	**4.29E-168**
Time Given*Scenario*Method	4	4	1.393	5183.557	**4.29E-168**

Table B.11: Effects Test for Log Total Network Overhead ($\alpha = 0.05$)

Source	Nparm	DF	SS	F Ratio	Prob > F
Method	1	1	268.191	415453.623	**4.30E-275**
Time Given	4	4	24.471	9477.083	**6.18E-189**
Time Given*Method	4	4	3.479	1347.445	**4.83E-122**
Scenario	1	1	0.610	944.206	**5.23E-069**
Time Given*Scenario	4	4	0.626	242.331	**8.85E-067**
Time Given*Scenario*Method	4	4	0.505	195.732	**1.59E-060**
Scenario*Method	1	1	0.389	602.493	**3.98E-056**

B.4 Comparison by Method at Time=40

Table B.12: 95% CI on Log Server Discoveries by Method (Time=40)

Estimator	Method	Estimate	Lower CI	Upper CI
Mean	Active	2.101	2.097	2.104
	Passive	1.431	1.431	1.431
Std Dev	Active	0.007	0.006	0.011
	Passive	0	0	0

Table B.13: 95% CI on Log Client Discoveries by Method (Time=40)

Estimator	Method	Estimate	Lower CI	Upper CI
Mean	Active	0	0	0
	Passive	2.507	2.507	2.507
Std Dev	Active	0	0	0
	Passive	0	0	0

Table B.14: 95% CI on Log Total Discoveries by Method (Time=40)

Estimator	Method	Estimate	Lower CI	Upper CI
Mean	Active	2.101	2.097	2.104
	Passive	2.542	2.542	2.542
Std Dev	Active	0.007	0.006	0.011
	Passive	0	0	0

Table B.15: 95% CI on Log Overhead Packets In by Method (Time=40)

Estimator	Method	Estimate	Lower CI	Upper CI
Mean	Active	3.434	2.811	4.057
	Passive	0	0	0
Std Dev	Active	1.252	0.940	1.877
	Passive	0	0	0

Table B.16: 95% CI on Log Overhead Packets Out by Method (Time=40)

Estimator	Method	Estimate	Lower CI	Upper CI
Mean	Active	5.067	4.952	5.182
	Passive	2.299	2.293	2.305
Std Dev	Active	0.231	0.173	0.346
	Passive	0.012	0.009	0.019

Table B.17: 95% CI on Log Total Network Overhead by Method (Time=40)

Estimator	Method	Estimate	Lower CI	Upper CI
Mean	Active	5.175	5.115	5.234
	Passive	2.299	2.293	2.305
Std Dev	Active	0.120	0.090	0.180
	Passive	0.012	0.009	0.019

Appendix C. Source Code and Configuration Files

The smaller, simpler source code files are listed here. The larger configuration and source code files and multi-file projects are available separately. See Appendix D for obtaining these files.

C.1 Source Code

C.1.1 uncollect.pl.

```
#!/usr/bin/perl -w

my ($ip, $eth, $sports, $cports, $uports, $item);

my @ports;

$ip="";
$eth="";
$sports="";
$cports="";
$uports="";
# eat the first line
$item=<STDIN>;
$item="";

while (<STDIN>) {
  next if /mapped items/;
  next if /received/;
  next if /Done/;

  ($ip, $eth, $sports, $cports, $uports)
        = m/^(.*) (.*) S\((.*)\) C\((.*)\) U\((.*)\)$/ ;

  if (length($sports)>0) {
    @ports = split (" ", $sports);
    foreach $item (@ports) {
      print $ip.":serv:".$item."\n";
    }
  }

  if (length ($cports) >0) {
    @ports = split (" ", $cports);
    foreach $item (@ports) {
      print $ip.":clnt:".$item."\n";
    }
  }

  if (length($uports)>0) {
    @ports = split (" ", $uports);
    foreach $item (@ports) {
      print $ip.":uclnt:".$item."\n";
      print $ip.":userv:".$item."\n";
    }
  }
}
```

C.1.2 ungrep.pl.

```
#!/usr/bin/perl -w

# normalize the output of the nmap 'grep' format log for input to matchem
# only server-side ports are seen by nmap so no client-side processing
my ($ip, $eth, $sports, $cports, $uports, $item);

my @ports;

$ip="";
$status="";
$sports="";
$port="";
$proto="";

# eat the first line
$item=<STDIN>;
$item="";
$dummy="";

while (<STDIN>) {
  next if /^\#/;

  ($ip, $sports, $dummy) =
                m/^Host:\s+(.*)\s+\(\)\s+Ports:\s+(.*)(\s+Ignore.*|\#.*)$/;
  if ($sports) {
    foreach $j (split (/,\s+/, $sports)) {
      ($port, $status, $proto, $dummy) = split ('\/', $j);
      if ($status =~ /open/i) {
        if ($proto =~ /UDP/i) {
          print $ip.":uclnt:".$port."\n";
          print $ip.":userv:".$port."\n";
        }
        elsif ($proto =~ /TCP/i) {
          print $ip.":serv:".$port."\n";
        }
      }
    }
  }
}
```

C.1.3 matchem.java. The source code for *matchem* is provided on a data CD-ROM and can be found under the *matchem* directory at filename *matchem.java*.

C.1.4 syntraf. The source code for *syntraf* is provided on a data CD-ROM and can be found under the *syntraf* directory.

C.1.5 lanmap and collector. The project files and source code for *lanmap* and *collector* are provided on a data CD-ROM and can be found under the *lanmap* directory.

C.1.6 nmap modifications. The differential file describing this investigation's modifications to *nmap* is provided on a data CD-ROM and can be found under the *nmap* directory at filename *nmap-timer.diff*.

C.2 Configuration Files

C.2.1 honeyd configuration. The *honeyd* configuration files for the "Normal" and "Firewall" scenario is provided on a data CD-ROM and can be found under the *configs* directory. *scenario1.travis.cfg* provides the Firewall scenario and *scenario2.travis.cfg* provides the Normal scenario.

C.2.2 syntraf configuration. The *syntraf* configuration file used throughout the experiments is provided on a data CD-ROM and can be found under the *configs* directory, at filename *travis.scn*.

C.3 Experiment System

Figure C.1 describes the experimental cycle.

```
power up all systems (switch, four computers)
login to the collector (the only system with user console)
ensure switch SPAN is set up correctly
  for active, sniffer/collector needs SPAN emitter
  for passive, mapper needs SPAN emitter)
remotely login to the other three systems
ensure each systems secondary network adapter is enabled
for a passive test:
  login a second time to mapper
  on the honeynet: ./honeyd -i eth0 10.0.0.0/8 -f firewall.scenario.cfg
  on the collector: ./collector > collector.log
  on the mapper: tcpdump -i eth0 ip or arp -w sniffer.tcpdump
  on the second mapper login: ./lanmap -i eth0 -s <address of collector> > lanmap.log
  on the generator: java syntraf -t <time> -f traffic.scn
for an active test:
  login a second time to mapper
  on the honeynet: ./honeyd -i eth0 10.0.0.0/8 -f firewall.scenario.cfg
  on the sniffer: tcpdump -i eth0 ip or arp -w snifferlog.tcpdump
  on the mapper: tcpdump -i eth0 ip or arp -w nmaplog.tcpdump
  on the generator: java syntraf -t <time> -f traffic.scn
  immediately on the second mapper login:
     nmap -i eth0 'network addresses' -n -oG nmaplog.grep
```

Figure C.1: Experiment System Configuration

Appendix D. Availability of Source Code and Configuration Files

Availability of source code and projet files for *lanmap*, *collector*, and *syntraf*; differential files describing the modifications to *nmap*; and the configuration files for *honeyd* and *syntraf* are not included as part of this document.

Interested parties should direct their inquiries to:

Dr. Richard Raines
AFIT/ENG 2950 Hobson Way
Wright-Patterson AFB, OH
45433-7765

Bibliography

1. "C-language hash table implementation source code." Author unattributed.
2. "freshmeat.net: Statistics and Top 20." `http://freshmeat.net/stats/`; accessed February 5, 2004.
3. "How does Traceroute work?." `http://www.tek-tips.com/gfaqs.cfm/lev2/5/lev3/60/pid/581/fid/381`; accessed February 11, 2004.
4. "Commanders' NOTAM 00-5: One Air Force, One Network," September 2000. `http://www.issues.af.mil/notams/notam00-5.html`; accessed January 16, 2004.
5. "Debian GNU/Linux 3.0 Released," *Debian News* (2002). `http://www.debian.org/News/2002/`; accessed January 24, 2004.
6. *Chariot Product Documentation*, 2003. `http://www.ixiacom.com/products/performance_applications/pa_display.php?skey=pa_ixchariot`; accessed February 4, 2004.
7. "ettercap Documentation," 2003. `http://ettercap.sourceforge.net/`; accessed July 15, 2003.
8. "The Honeynet Project Briefing," (April 2003). `http://www.honeynet.org/speaking/honeynet_project-2.0.4.ppt.zip`; accessed April 30, 2003.
9. *Merriam-Webster Dictionary, Online Edition*. 2003. `http://www.m-w.com/`; accessed July 15, 2003.
10. "NetFlow Overview Briefing," February 2003. `http://www.cisco.com/application/vnd.ms-powerpoint/en/us/guest/tech/tk362/c1482/ccmigration_09186a0080182b50.ppt`; accessed July 15, 2003.
11. "NetSpec Manual," 2003. `http://www.ittc.ukans.edu/netspec/docs/NetSpecUser.pdf`; accessed September 22, 2003.
12. "NISTNet Documentation," 2003. `http://www-x.antd.nist.gov/nistnet/index.html`; accessed September 22, 2003.
13. "WhatsUp Gold," 2004. `http://www.ipswitch.com/products/whatsup/index.html`; accessed February 4, 2004.
14. Andel, T. R. "(U//FOUO) Traffic Generator." E-Mail between Capt Todd Angel and Dr Rick Raines, September 2003.
15. Arkin, O. and Yarochkin, F. "Xprobe v2.0: A "Fuzzy" Approach to Remote Active Operating System Fingerprinting," (August 2002). `http://www.sys-security.com/archive/papers/Xprobe2.pdf`; accessed July 15, 2003.

16. Beddoe, M. and Abad, C. "The Siphon Project: The Passive Network Mapping Tool," (2000). http://siphon.datanerds.net/; accessed July 15, 2003.

17. Branigan, S., Burch, H., Cheswick, B., and Wojcik, F. "What Can You Do with Traceroute?," *IEEE Internet Computing Online* (2001). http://www.computer.org/internet/v5n5/; accessed June 26, 2003.

18. Brownlee, N., Claffy, K. C., Murray, M., and Nemeth, E. "Methodology for Passive Analysis of a University Internet Link," (2001). http://www.caida.org/outreach/papers/2001/MethodAnalyseLink/passive.pdf; accessed July 15, 2003.

19. Chandran, R. and Pakala, S. "Simulating Networks with Honeyd," (December 2003). http://www.paladion.net; accessed January 24, 2004.

20. Cisco Systems, Inc. *Configuring the Catalyst Switch Port Analyzer (SPAN) Feature*, 2003. http://www.cisco.com/warp/public/473/41.pdf; accessed July 15, 2003.

21. Claffy, K. C., Braun, H.-W., and Polyzos, G. C. "Application of sampling methodologies to wide-area network traffic characterization," (1993). http://www.caida.org/outreach/papers/1993/asmw/sigcomm.sampling.pdf; accessed July 15, 2003.

22. Claffy, K. C., Braun, H.-W., and Polyzos, G. C. "A parameterizable methodology for Internet traffic flow profiling," (1995). http://www.caida.org/outreach/papers/1995/pmi/JSAC-flows.pdf; accessed July 15, 2003.

23. Cohen, F. "Managing Network Security: What's Happening Out There," *Network Security Magazine*, 8–11 (August 1999).

24. Department of the Air Force. *Concept of Operations for Air Force Information Enterprise, DRAFT*, September 2000.

25. Department of the Air Force. *Air Force Instruction 33-112, Computer Systems Management*, February 2001.

26. Department of the Air Force. *Air Force Instruction 33-115v1, Network Management*, November 2002.

27. Diversified Data Resources, Inc. *ACE-SNMP: An Introductory Overview of SNMP*, 1999. http://alpha400.ee.unsw.edu.au/~tele9303/notes/DDRI.pdf; accessed July 15, 2003.

28. Drobisz, J. and Christensen, K. J. "Adaptive Sampling Methods to Determine Network Traffic Statistics including the Hurst Parameter," (1998). http://dlib.computer.org/conferen/lcn/8810/pdf/88100238.pdf; accessed July 15, 2003.

29. Giovanni, C. "Passive Mapping: An Offensive Use of IDS," (March 2000). http://packetstormsecurity.nl/papers/IDS/OffensiveUseofIDS.pdf; accessed June 26, 2003.

30. Giovanni, C. "Passive Mapping: The Importance of Stimuli," (June 2000). http://packetstormsecurity.nl/papers/IDS/PassiveMappingviaStimulus.pdf; accessed June 26, 2003.

31. Hewlett-Packard. *Managing Your Network with HP OpenView Network Node Manager*, 2003.

32. Horstman, C. S. and Cornell, G. *Core Java 2, Volume II - Advanced Features*. Palo Alto, CA: Sun Microsystems Press, 2000.

33. Huffaker, B., Plummer, D., Moore, D., and Claffy, K. C. "Topology discovery by active probing," (2002). http://www.caida.org/outreach/papers/2002/SkitterOverview/skitter_overview.pdf; accessed July 15, 2003.

34. Jacobson, V., Leres, C., and McCanne, S. *pcap Manual Page*, 2003.

35. Jacobson, V., Leres, C., and McCanne, S., "tcpdump Manual Page," 2003. http://www.tcpdump.org/tcpdump_man.html; accessed January 21, 2004.

36. Jain, R. *The Art of Computer Systems Performance Analysis*. New York, New York: John Wiley and Sons, 1991.

37. Kent, K. "Evaluating Network Intrusion Detection Signatures, Part Three," (December 2002). http://www.securityfocus.com/infocus/1651; accessed January 24, 2004.

38. MKS Software. *snmpwalk Manual Page*, 2003. http://www.mkssoftware.com/docs/man1/snmpwalk.1.asp; accessed July 15, 2003.

39. Nazario, J. "Passive System Fingerprinting using Network Client Applications," (January 2001). http://secinf.net/uplarticle/1/passive.pdf; accessed July 15, 2003.

40. Plummer, D. C., "Ethernet Address Resolution Protocol, RFC 826," 1982. http://www.faqs.org/rfcs/rfc826.html; accessed January 24, 2004.

41. Postel, J. B., "Internet Control Message Protocol, RFC 792," September 1981. http://www.faqs.org/rfcs/rfc792.html; accessed February 10, 2004.

42. Postel, J. B., "Internet Protocol, RFC 791," September 1981. http://www.faqs.org/rfcs/rfc791.html; accessed February 10, 2004.

43. Postel, J. B., "Transmission Control Protocol, RFC 793," September 1981. http://www.faqs.org/rfcs/rfc793.html; accessed February 10, 2004.

44. Poulsen, K. "Matrix Sequel has Hacker Cred," *Security Focus* (May 2003). http://www.securityfocus.com/news/4831; accessed January 24, 2004.

45. Reynolds, J. and Postel, J., "Assigned Numbers, RFC 1700," October 1994. http://www.faqs.org/rfcs/rfc1700.html; accessed February 11, 2004.

46. Rossey, L., "LARIAT Overview Briefing," September 2003. PowerPoint Slides.

47. Scambray, J., McClure, S., and Kurtz, G. *Hacking Exposed, 2nd Edition*. Berkely, CA: Osborne/McGraw-Hill, 2001.

48. Shargorodsky, S. "Russian security said to penetrate Internet," *Associated Press* (February 2000). http://www.da.wvu.edu/archives/002202/news/world_nation.html; accessed July 15, 2003.

49. Smith, J., "C-language linked list implementation source code," 2000. http://slicer69.tripod.com/code/index.html; accessed September 21, 2003.

50. Spitzner, L. *Honeypots: Tracking Hackers*. Boston, Massachusetts 02116: Addison-Wesley, 2003.

51. Stephens, R. W. *Unix Network Programming, Volume 1, 2nd Edition*. Upper Saddle River, NJ: Prentice Hall, 1998.

52. Undy, M., Bing, M., and Turner, A., "tcpreplay Home Page," 2003. http://tcpreplay.sourceforge.net; accessed January 21, 2004.

53. Vision, M. "Passive Host Fingerprinting," (February 2001). http://www.whitehats.com/library/passive/index.html; accessed July 15, 2003.

54. Wachowski, A., Wachowski, L., and Silver, J., "The Matrix Reloaded." Motion Picture / DVD, 2003. Performers Keanu Reeves, Laurence Fishburne, Carrie-Ann Moss.

55. Yarochkin, F. "Network Mapping Thesis (Interview)." E-Mail between MSgt J.B. Kuntzelman and Fyodor Yarochkin, September 2003.

56. Yarochkin, F. *nmap Documentation*. Insecure.org, 2003. http://www.insecure.org/nmap/nmap_documentation.html; accessed July 15, 2003.

57. Zalewski, M. and Stearns, W. *p0f Documentation*, 2003. http://lcamtuf.coredump.cx/p0f/README; accessed July 15, 2003.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From – To)*
23-03-2004	Master's Thesis	March 2003 – March 2004

4. TITLE AND SUBTITLE
COMPARATIVE ANALYSIS OF ACTIVE AND PASSIVE MAPPING TECHNIQUES IN AN INTERNET-BASED LOCAL AREA NETWORK

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Kuntzelman, James B., MSgt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way, Building 640
WPAFB OH 45433-8865

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCS/ENG/04-09

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Neal Ziring
TD/C4 NSA
Fort George G. Meade, MD 20755-6000

10. SPONSOR/MONITOR'S ACRONYM(S)

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

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Network mapping technologies allow quick and easy discovery of computer systems throughout a network. Active mapping methods, such as using *nmap*, capitalize on the standard stimulus-response of network systems to probe target systems. In doing so, they create extra traffic on the network, both for the initial probe and for the target system's response. Passive mapping methods work opportunistically, listening for network traffic as it transits the system. As such, passive methods generate minimal network traffic overhead. Active methods are still standard methods for network information gathering; passive techniques are not normally used due to the possibility of missing important information as it passes by the sensor. Configuring the network for passive network mapping also involves more network management.

This research explores the implementation of a prototype passive network mapping system, *lanmap*, designed for use within an Internet Protocol-based local area network. Network traffic is generated by a synthetic traffic generation suite using *honeyd* and *syntraf*, a custom Java program to interact with *honeyd*. *lanmap* is tested against *nmap* to compare the two techniques.

Experimental results show that *lanmap* is quite effective, discovering an average of 76.1% of all configured services (server- and client-side) whereas *nmap* only found 27.6% of all configured services. Conversely, *lanmap* discovered 19.9% of the server services while *nmap* discovered 92.7% of the configured server-side services. *lanmap* discovered 100% of all client-side service consumers while *nmap* found none. *lanmap* generated an average of 200 packets of network overhead while *nmap* generated a minimum of *minimum* 8,600 packets on average--up to 155,000 packets at its maximum average value.

The results show that given the constraints of the test bed, passive network mapping is a viable alternative to active network mapping, unless the mapper is looking for unused server-side services.

15. SUBJECT TERMS
COMPUTER NETWORKS, COMMUNICATIONS PROTOCOLS, NETWORK TOPOLOGY

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Dr. Richard A. Raines
U	**U**	**U**	**UU**	103	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-6565, ext 4278 (richard.raines@afit.edu)

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18