5-2004

A Vulnerability Assessment of the East Tennessee State University Administrative Computer Network.

James Patrick Ashe
East Tennessee State University

A Vulnerability Assessment of the East Tennessee State University Administrative Computer Network

---

A thesis

presented to

the faculty of the Department of Computer and Information Sciences

East Tennessee State University

In partial fulfillment

of the requirements for the degree

Master of Science in Computer Science

---

by

James P. Ashe

May 2004

Dr. Phillip E. Pfeiffer, IV, chair

Dr. Gene Bailey

Dr. Qing Yuan

Keywords: network security, system security, security audit, nmap, nessus

Abstract

A Vulnerability Assessment of the East Tennessee State University Administrative Computer Network

by

James P. Ashe

A three phase audit of East Tennessee State University's administrative computer network was conducted during Fall 2001, Spring 2002, and January 2004. Nmap and Nessus were used to collect the vulnerability data. Analysis discovered an average of 3.065 critical vulnerabilities per host with a low of 2.377 in Spring 2001 to a high of 3.694 in Fall 2001. The number of unpatched Windows operating system vulnerabilities, which accounted for over 75% of these critical vulnerabilities, strongly argues for the need of an automated patch deployment system for the approximately 3,000 Windows-based systems at ETSU.

Contents

List of Tables

Table page

List of Figures

Chapter 1
Introduction

The security of a university's computing network is important to that university's faculty, staff and students. A university's educational processes are hindered and its students, and faculty and staff are inconvenienced when network services or system services become unavailable. In addition, when networked records are not secured, sensitive information such as personal student information can be retrieved and made public.

Network security refers to the application of systems security to a connected network of systems. *System security* is the incorporation of standard security measures into computing. *Security* is defined as measures adopted by organizations to prevent espionage, sabotage, attack or other crime [AmHeritage]. According to Farmer and Spafford,

> *To make a system secure means to protect the information from disclosure; protecting it from alteration; preventing others from denying access to the machine, its services and its data; preventing degradation of services that are present; protecting against unauthorized changes; and protecting against unauthorized access* [Farmer1].

Attacks upon network security include non-technological threats like unwarranted physical proximity and sabotage, and technological threats like Internet worms, hacking, computer viruses, wiretapping, and espionage. Technological threats can be further categorized as *direct network-based attacks* or *automated attacks*. Direct network-based attacks include attempts to learn information about a network or gather information stored in a networked system. Automated attacks include programs or scripts that, once

released into a networked environment, attack vulnerable systems in that environment.

This thesis is concerned with the use of security audits to assess the vulnerability of university computer systems and networks. A security audit examines systems on a local area network to detect "holes" that may be exploited by malicious people. Such holes include physical intrusion, abuse of privilege by legitimate users, and software vulnerabilities.

This work took the form of an investigation of the security of the East Tennessee State University administrative computer network. Motivated by a major system compromise, a previous study of the ETSU administrative network activity sought to determine the *types*, *severity*, and *number* of potential attacks an academic network would experience over a period of time [Cui]. Over a six month period between October 2000 and March 2001, a large percentage of the total incoming network traffic was captured and further analyzed. This research recorded a range of two-hundred thousand to two million attack packets per day, involving port scans, denial of service attacks, and various attempts to exploit known system vulnerabilities. Cui's research raised the question that this research attempted to address: 'If the ETSU network environment is not protected, and attacks happen daily, what is the state of ETSU security?'

This research sought to determine the susceptibility of the ETSU administrative computer network to attack. East Tennessee State University, a moderate size regional university with approximately 11,250 students and 3,000 staff and faculty, controls a single class B Internet address space (151.141.0.0/16). Approximately 3,600 Internet addresses on this network are in active use.

ETSU has two distinct computer networks. One network, which is excluded from this study, connects student dormitories with one another and the Internet. Computers on this network, which receive private internet addresses in the

192.168.x.y range, are only directly reachable by other ETSU student network systems, and therefore not vulnerable to attacks originating from the ETSU administrative network or the Internet. The other network, the administrative network, connects all administrative buildings, computer labs and classrooms with one another and the Internet. Systems on the ETSU administrative network are assigned a 151.141.x.y Internet address and are directly accessible by any computer on the Internet.

The audit was carried out in three phases during the Fall 2001, Spring 2002 and Spring 2004 semesters. A full network audit was conducted during the initial two phases using the Nmap and Nessus security auditing tools. The third phase was a partial network scan in Spring 2004 to gauge the degree to which earlier observations had changed.

Each scan attempted to balance the need to gather data with the need to accommodate normal network operation. Tests that would have involved attempted denial of service attacks were avoided. Tests of systems that were known to have personal firewalls were skipped, to avoid alarming users.

Scans were also done in a way that attempted to account for computer downtime. Under ideal circumstances, all systems would have remained connected to the network at all times. This was true of some computer systems, such as laboratory systems, network equipment and servers. Most systems, such as administrative workstations, are suspected to be shut down on a nightly basis. Accordingly, rescans of unreachable hosts were used to increase the chance of compiling data on all networked hosts.

The key findings of the study found an average host exhibited 3.065 critical vulnerabilities, with a low of 2.377 in Spring 2001 to high a of 3.694 in Fall 2001. Unpatched Windows operating system vulnerabilities were the top security threat to the ETSU administrative network over this 2½ year span accounting for over 80% of all critical vulnerabilities. However, an automated patch management system has yet to be implemented because of concerns

expressed by the user community over operating system patches 'breaking' applications.

The remainder of this thesis is separated into four sections. Section 2, *Related Work*, describes the background research performed for this study. This section also describes the tools used in this study. Section 3, *Research Methodology*, describes the motivation for this study, as well as the strategy used to perform the security audit. Section 4, *Data Analysis*, examines the data gathered during the audit. Section 5, *Conclusion*, includes a summary of the study's findings and potential avenues for research.

Chapter 2
Related Work

The current section reviews tools for detecting security vulnerabilities that are directly related to work in this study. Section 2.1 discusses the two tools used in this study, Nmap (Section 2.1.2.1) and Nessus (Section 2.1.2.2).

The remainder of this discussion assumes a fair understanding of Windows and UNIX networking, the TCP and UDP protocols, network security and different activities normally placed under the 'hacking' umbrella.

2.1. Security Threats and Security Auditing Tools

2.1.1. Security Threats

Security auditing tools were developed to help network and systems administrators gather information about vulnerabilities in their networked environment. These vulnerabilities include unauthorized services, system configuration errors, well-known vulnerabilities and weak user passwords.

Identifying unauthorized services. An *unauthorized service* is an unauthorized host program that listens for and accepts connections from other systems. Legitimate services typically use well-known ports to accept connections. For example, web servers listen on TCP port 80 and telnet hosts listen on TCP port 23. A port that is in the 'listening' state is referred to as an *open* port. Examples of unauthorized services include services running on non-standard ports, services that respond to user requests in non-standard ways, and services that are unnecessary for the system being scanned. These can be detected by scanning a host for all ports actively listening for connection requests and examining how they process incoming messages.

Identifying system configuration errors. A *system configuration* error is a misconfiguration of a service that leaves the service or system susceptible to attack. Unset passwords and shared key files are examples of configuration errors. Operating system installation procedures assign default permissions and passwords to certain files and accounts. In certain cases, the system default creates system vulnerabilities. For example, a UNIX password file readable by the 'world' group can be copied and deciphered, compromising the host system.

Identifying 'well-known' vulnerabilities. Well-known system vulnerabilities are discovered software programming errors or common administrator oversights that allow a system or network to be compromised. These include publicly documented instances of buffer overflows and backdoors in application and operating system software. Finding and patching holes increases network security by minimizing the infiltration paths an attacker can use.

Identifying weak user passwords. User passwords that are easy to guess or crack can allow outsiders to gain network access. Strategies for decreasing the likelihood of compromised passwords include mandatory policies for password selection.

2.1.2. Security Auditing Tools

Security auditing tools are utilities that detect vulnerabilities in systems that could be exploited to compromise a system. The types of vulnerabilities detected vary with each tool. LC4 (http:///www.atstake.com/lc4) is a tool created for the sole purpose of testing password strength. Other tools like the IIS Lockdown Tool (http://www.microsoft.com/windowsserver2003/iis/default.mspx) test a service for configuration errors and make reparations where necessary. Tools such as Nessus (http://www.nessus.org) and the Microsoft Baseline Security

Analyzer (http://www.microsoft.com/technet/security/tools/mbsahome.asp) check for exploits against a database of well-known attacks. The port-scanning tool Nmap (http://www.insecure.org) aids in detecting unauthorized services. Using a combination of tools is the typical way to maximize the number of detected vulnerabilities in a networked environment.

Nmap and Nessus were chosen for this study, in part, because these tools can work in conjunction to minimize audit time and maximize the number of discovered vulnerabilities. Nessus uses a user-configurable database of tests to drive vulnerability analyses. The Nessus database is updated frequently, sometimes daily, by the security community, in response to discoveries of new exploits. Nessus can be configured to use Nmap to determine the open ports on a system before conducting tests, which allows Nessus to focus solely on potentially vulnerable ports. Finally, both tools are free to use, and native to Linux, a free operating system.

2.1.2.1. Nmap

Nmap is a network auditing tool that scans network hosts for open ports. Port scans can determine if a host is offering errant services or failing to offer required services. Examples of errant services are an http daemon on a host not listed as a web server and a backdoor opened by a Trojan horse. Nmap can also determine the operating system running on a scanned host, and scan firewalls to determine the ports a firewall effectively filters [Fyodor].

Nmap can scan network hosts using one of six methods: TCP connect() scans, TCP SYN scans, stealth FIN scans, Xmas tree scans, Null scans, UDP scans, and ping scans.

The TCP connect() scan, the simplest port scanning method, attempts to create a TCP connection between the scanning client and the host. The auditor determines the TCP ports that Nmap will scan. Nmap allows any user to

execute a TCP connect() scan, regardless of privilege. The TCP connect() scan is highly traceable by intrusion detection systems, and an unlikely choice of attackers [Fyodor].

The TCP SYN scan mimics the TCP connect() scan but does not complete the TCP connection. Nmap sends a SYN (synchronize) packet to the host and awaits a reply. A RST (reset) packet from the host indicates there are no services available on the scanned TCP port. An ACK (acknowledgement) packet indicates a service is available on the scanned TCP port. When the host receives an ACK packet, Nmap sends a RST packet to destroy the pending connection. Nmap limits the use of TCP SYN scans to root users, since standard TCP stacks do not track reset connections [Fyodor].

FIN scans, Xmas tree scans, and null scans use irregular TCP packets to discover open ports. The TCP header contains a 6-bit control block with six flags: URG (urgent data), ACK (positive acknowledgement), PSH (push data to receiver), RST (reset connection), SYN (synchronize), and FIN (finish transfer) [RFC793]. The Stealth FIN scan sends a FIN packet to the host and awaits a reply. A closed port that receives a FIN packet should reply with a RST packet as detailed in RFC 793, reset generation, rule 1:

> *If the connection does not exist (CLOSED) then a reset is sent in response to any incoming segment except another reset. In particular, SYNs addressed to a non-existent connection are rejected by this means [RFC793].*

Ports that fail to reply with an RST packet are assumed to be open.

Xmas Tree scans and Null scans are variations of the Stealth FIN scan. The Xmas Tree scan sets the FIN, URG and PSH flags. The Null scan turns off all flags n the 6-bit control block. Stealth FIN, Xmas Tree and Null scans do not work against some operating systems because of TCP protocol modifications. In

particular, the Microsoft Windows TCP protocol will not send an RST packet in response to these scans [Fyodor].

Nmap detects open UDP ports by sending a zero-byte UDP packet to every host UDP port. If the host replies the port is unreachable, the scanned port is closed. No response from the host indicates the UDP port is open [Fyodor].

Using a ping scan, Nmap can determine the existence of systems at specified IP addresses. Nmap ping scans use standard ICMP echo packets and TCP ACK packets sent to port 80. This feature of Nmap is most useful when sweeping a network segment to determine the number of hosts on a network segment. The TCP ACK packet discovers hosts behind a firewall that filters standard ICMP echo packets. TCP ACK packets will only be sent when a root user initiates a ping scan [Fyodor].

2.1.2.2. Nessus

Nessus is a UNIX-based security scanner that checks for well-known vulnerabilities on various platforms. The specific vulnerabilities that Nessus checks are determined as part of an initial configuration step that involves choosing the plug-ins of the associated vulnerability that the user wishes to discover. Tests for vulnerabilities can be created by anyone using the NASL scripting language. The tool's authors determine which plug-ins are distributed with the scanner for use by the Nessus community. Active support from the open source community has ensured a constantly increasing supply of new, downloadable, vulnerability tests.

Nessus emulates the first well-known security scanner, the Security Administrator Tool for Analyzing Networks (SATAN) [Farmer3]. SATAN, like Nessus, attempts to break into a host to determine the level of host security. Unlike Nessus, SATAN cannot detect recently discovered vulnerabilities, and was therefore rejected for this study.

The Nessus program uses two different executables to scan hosts and collect data. The 'server,' called the Nessus daemon (nessusd), is responsible for 'attacking' each host. The Nessus daemon must be executed on a UNIX-based machine. The 'client,' called nessus, collects the data from the attack. Currently, the client portion of the Nessus program is available for UNIX and Windows [Deraison1].

The server portion of Nessus supports various options for multiple situations. Nessus offers multi-user support and the ability to grant different rights to each user. Multi-user support allows security administrators to grant rights to scan selected subsets of a network. Nessus can also use multithreading to conduct concurrent audits. The number of hosts that can effectively be audited simultaneously is determined by available network bandwidth and the processing speed of the Nessus daemon computer [Deraison1]. The Nessus client collects and arranges data received from the vulnerability assessment [Deraison2].

Nessus offers the following twenty-three families of rule sets for use in a security audit [Deraison3].

- *Trojan backdoors*. Trojan backdoor servers are often distributed through e-mail as attachments. The unsuspecting recipient executes the infected program expecting to view something interesting or amusing. Instead, the attachment installs the Trojan backdoor program onto the system. Scanning well-known Trojan ports detects the presence of a Trojan backdoor.
- *CGI script vulnerabilities*. A CGI script is a server-side executable. CGI scripts with programming errors are vulnerable to security flaws. Searching the UNIX \cgi-bin directory detects possible flawed CGI scripts .
- *Cisco network device vulnerabilities.* Nessus, in most cases, uses SNMP to check the version of the system IOS and compare it to version numbers of

the Cisco IOS with known vulnerabilities. Use of these plug-ins requires knowledge of a Cisco device's read-only SNMP community string. This group of plug-ins was added to Nessus too late for inclusion in this study .

- *Default UNIX accounts.* This rule set tests certain well-known UNIX accounts for having a default password, or no password at all. This group of plug-ins was added to Nessus too late for inclusion in this study.
- *Denial of Service (DoS) attacks.* This rule set checks for DoS susceptibility in applications, servers, switches and routers. A denial of service (DoS) attack is launched by issuing commands that crash the target by means of 'confusion' or buffer overflow. Nessus attempts to crash a host using a DoS attack to detect software and hardware vulnerabilities.
- *Finger daemon vulnerabilities.* The UNIX 'finger' command shows user information. Depending on the daemon's configuration, the user being 'fingered' can belong to a local or remote network. Finger daemons provide the names of active accounts, which is a security threat by itself. Some finger daemons have security flaws that offer private information about users, or allow attackers to gain system control.
- *Firewall configuration errors.* This rule set attempts to bypass the firewall and alerts administrators of subsequent firewall configuration errors. Malicious parties can 'hop' a poorly configured firewall to access network resources.
- *FTP vulnerabilities.* FTP (File Transfer Protocol) servers distribute files to users with proper credentials. Some FTP servers exhibit flaws that allow attackers to retrieve arbitrary information or gain control of the system. Detection of FTP server vulnerabilities involves attempting well-known exploits.
- *UNIX shell exploits.* A UNIX shell accepts user commands for execution. Shell access is normally reserved for authorized system users. Gaining shell access allows a user to execute hazardous commands, change a host's

configuration and access possibly sensitive data. Not all possible commands and applications may be executed with just shell access, as root privileges may be necessary.

- *Remote UNIX root exploits.* This rule set tests whether outside parties can gain root access to a system, and with it, the ability to execute any command or application, or examine any data file on the exploited host.
- *General.* This rule set tests for programs and daemons that provide their name and version number. Attackers can use this information to determine if a host might exhibit well-known vulnerabilities. Nessus attempts to gather operating system and version information from all queried hosts.
- *Miscellaneous.* This rule set checks for blank or default passwords within daemons and hardware devices. These plug-ins can determine if an initial password was not changed, or left blank after the daemon or hardware device installation. When a password is blank or unchanged, an attacker can gain administrative access to the daemon or hardware device. The attacker can also change the password to block authorized users. Nessus will test certain hardware devices and daemons for default or blank passwords and alert the administrator if needed.
- *Novell Netware vulnerabilities*. This group of plug-ins was added to Nessus too late for inclusion in this study.
- *NIS (Network Information Service) vulnerabilities*. The NIS server gathers information about network services. Systems query the NIS to locate appropriate network services. NIS servers should not be accessible from outside of the network. An outsider can obtain a network layout by querying an easily accessible NIS server. Network layout information will help an outsider launch an attack with a higher probability of success.
- *Peer-to-peer file sharing.* Some well-known peer-to-peer file sharing programs like Kazaa, Morpheus and Gnutella, if configured incorrectly,

allow anyone to access a computer's file system. This group of plug-ins was added to Nessus too late for inclusion in this study.

- *Nmap port scanning*. Nmap was discussed in Section 2.1.2.1.
- *Sensitive file retrieval vulnerabilities*. Password file retrieval allows an attacker to decipher username/password combinations for use in a consequent break-in. Nessus also uses this rule set's plug-ins to download any file with a known path name and file name.
- *Remote Procedure Call (RPC) vulnerabilities*. Although most remote procedure calls do not pose an immediate security threat, unused remote procedure calls should be disabled in case a vulnerability is discovered in the future.
- *SMTP server vulnerabilities*. An SMTP (Simple Mail Transport Protocol) server delivers e-mails to their destination. Some tests in this rule set attempt to crash the SMTP server. Other tests determine if outsiders can use the SMTP server to send or relay e-mail. Nessus detects and alerts administrators of these security holes and other undesirable SMTP server settings.
- *Simple Network Management Protocol (SNMP) information disclosure vulnerabilities.* The Simple Network Management Protocol allows administrators to gather user, service and executing process information. Nessus issues SNMP queries to obtain private system or network information.
- *Useless daemons and services*. Some useless daemons and services can be exploited to gain user and network information. Other useless daemons and services can be exploited to commandeer network bandwidth.
- *Windows vulnerabilities, including absent hotfixes*. As security holes are detected in Windows operating systems and applications, Microsoft offers patches, or hotfixes, to alleviate the security threat. Good network security practices involve actively patching vulnerable software. Nessus

can detect vulnerabilities that can be patched by applying a hotfix issued by Microsoft.

- *Windows Access Control List (ACL) enumeration.* Determining a membership of a Windows system ACL can provide a starting point for launching attacks against privileged account, such as accounts belonging to Administrators or Domain Administrators. This group of plug-ins was added to Nessus too late for inclusion in this study.

Chapter 3
Research Methodology

The study attempted to achieve two goals: first, to try to determine a 'level' of vulnerability of the networks systems of a representative regional university, and to determine the attractiveness of a representative regional university as a target for exploitation.

East Tennessee State University has approximately 3,600 Internet addresses in active use by a combination of workstations, servers, hubs, switches, routers, printers, print servers and other miscellaneous devices, such as personal digital assistants and wireless access points. Approximately 300 of these network-based systems were networking devices; approximately 100 printers/print servers; and the remainder, standard computer systems.

The study described in this thesis was conducted in three scans. The first two scans featured complete vulnerability checks of the administrative networks. The first sweep began September 23, 2001 and ended December 3, 2001. The second sweep began January 7, 2002 and ended February 21, 2002. The third, partial scan, which gathered data from about a quarter of the hosts on the ETSU network, began January 5, 2004 and ended January 16, 2004.

At the time when the first two scans were conducted, ETSU did not use a firewall, intrusion detection system or patch management system to protect the administrative network. At that time, the task of patching system software on production servers and workstations was the responsibility of systems administrators and workstation users, respectively. Between the second and third scan, a firewall solution was implemented.

Data on potential system vulnerabilities during the first two scans was gathered using Nessus version 1.0.9 through 1.1.13. Nessus plug-ins were updated every morning to ensure the detection of newly discovered vulnerabilities. To detect open TCP ports, Nmap version 2.54 beta 30 was used in

conjunction with Nessus. These programs were hosted on a machine using RedHat Linux versions 7.1, and 7.2. During the third scan, Nessus version 2.0.7, Nmap version 3.0 and RedHat Linux version 9.0 were used.

Nessus was used to find well-known exploits on ETSU administrative network computer systems. Since the ETSU administrative network predominantly consists of Windows machines, unapplied 'hotfixes' were hypothesized to be the top security threat.

Nmap was used to detect unauthorized services hosted on ETSU computer systems. Certain open ports immediately show evidence of a compromised system. For example, a system exhibiting open UDP port 31337 is infected with the Back Orifice Trojan.

The scans of the ETSU class network, a class B network with approximately 3,600 live addresses, were conducted as a series of scans of class C address spaces. The ETSU network is logically partitioned into class C virtual LANs, with a few virtual LANs using two class C address spaces. Each virtual LAN represents a building on the ETSU campus, or a set of related machines: for example, the university's servers, or core routers, or all machines in a particular lab. Accordingly, systems in any given virtual LAN were expected to exhibit common traits, such as common software packages, or similar times of deployment.

Each class C address space was scanned twice per semester with an average of two weeks between scans. The first scan examined all hosts that were 'live,' or available for scanning. The second scan examined hosts with IP addresses not 'live' during the first scan. The class C address spaces were scanned in the same order during each semester, allowing for a smaller deviation of mean time between each complete scan of a class C address space. The number of class C address spaces scanned per day was determined by the number that could be scanned serially during a four hour time frame, between 8am and 12pm, on weekdays. Subnets were not scanned

in parallel because of the limited processing power of the machine used to gather data.

During the third scan, seven subnets with a 255.255.254.0 subnet mask were each scanned once. The only hosts that were scanned were those that were 'live.' Each subnet was scanned during a seven hour time frame, between 9am and 4:30pm on weekdays.

Data was analyzed with a program, nsrparser, which was developed for this work (see Appendix B). nsrparser returns any of the following information from a Nessus output file in the Nessus NSR format:

- Names or IP addresses, and total count of all hosts.
- Best guess of host operating system by examining the information returned in specific plug-ins.
- A listing of all plug-ins, with associated descriptions, that returned information.
- All open ports, sorted by number of vulnerabilities exhibited.
- Names or IP addresses, and total count of hosts with a specific port open.
- Names or IP addresses, and total count of hosts with vulnerabilities on a specific port
- Names or IP addresses, and total count of hosts with warnings on a specific port
- Names or IP addresses, and total count of hosts with notes on a specific port
- Names or IP addresses, and total count of hosts exhibiting a specific vulnerability.
- Names or IP addresses, and total count of hosts exhibiting a specific warning.
- Names or IP addresses, and total count of hosts exhibiting a specific note.

- Names or IP addresses, and total count of hosts with a specified search string located in their plug-in description.
- Names or IP addresses, and total count of hosts with two specified search strings located in plug-in descriptions for that host.

The versions of Nessus used for the first two scans had an important limitation: older versions of Nessus catalogued scan data by host IP address and not by MAC address. Accordingly, any host that changed IP addresses during the scan might have been scanned twice, or bypassed altogether if the host changed IP addresses betweens scans that occurred in a single semester. The standard DHCP lease at the time of the first two scans was three days. The recent versions of Nessus, since 1.1.14, support the option of cataloging hosts by MAC address. The option to use MAC addresses for cataloging would have been used if this cataloging method had been available for the first two scans. This option was not used during the third scan, as the 512 host subnets were only scanned once.

Hosts that were live during the first scan of a semester were not tested for new vulnerabilities by plug-ins downloaded between the two scans. Nessus does have an option to rescan a host for unchecked vulnerabilities. At the time of the first two scans, this option was labeled 'experimental' and was avoided to minimize incorrect data.

Hosts that did not reply to a ping were not scanned. Live hosts utilizing a software firewall such as ZoneAlarm or BlackIce Defender that disallow ping replies were not scanned. Allowing Nessus to scan a computer running a software firewall could have disrupted the practices of the system's user as ZoneAlarm is normally 'chatty' and obtrusive when alerting the system's user of potentially malicious activity.

Nessus plug-ins that emulated denial of service attacks were disabled so as not to disrupt a scanned systems operation. Therefore, vulnerabilities related to some denial of service attacks are absent from the data.

Chapter 4
Data Analysis

The types of vulnerabilities discussed were separated into three main categories, *local*, *network,* and *client.* A *local* vulnerability can only be exploited through an interactive login of the machine. A *remote* vulnerability may be exploited when the susceptible machine is connected to a network. *Client* vulnerabilities may be exploited when the subject machine connects to a malicious host.

Each of these three categories was further separated into five subcategories, listed in order of increasing severity, based on the end result of a successful exploit.

- *Information disclosure* – A machine offers information about itself, its users or the network it is connected to that helps malicious parties learn more about the subject, or other possible subjects.
- *Denial of Service* **–** An exploit renders a service on any machine disabled or unreachable by authorized clients.
- *Readable Service* **–** An exploit offers read access of a service to an unauthorized client.
- *Writeable Service* – An exploit offers read and/or write access of a service to an unauthorized client.
- *Privilege Elevation* – An exploit grants the malicious party 'higher' privileges than a standard guest or user account. In most cases, this refers to an unauthorized party gaining "root" or "administrator" access.

Based on the availability of the vulnerable service, vulnerabilities were classified as *active* and *dormant*. The term *dormant* was chosen over *inactive* because of the uncertainty that a vulnerable service would be available in the

future. The term *total* will refer to the sum of all *active* and *dormant* vulnerabilities.

Nessus categorizes its findings into one of three categories: vulnerability, warning and info. When referencing these categories, the terms *critical vulnerability, warning,* and *informational discovery* are used to refer to what Nessus categorizes as a 'vulnerability', a 'warning,' and an 'info', respectively.

4.1. Most Vulnerable Ports

Each of the two initial Nessus scans identified ports 139 (SMB over NetBIOS), 161 (SNMP), 80 (HTTP), 21 (FTP), and 25 (SMTP) as the five most vulnerable ports, ordered by number of critical vulnerabilities. The top five vulnerable ports remained consistent across the two initial scans.

Port 139, used by SMB over NetBIOS, was the most vulnerable port during both scans. During Fall 2001, the 11,450 port 139 critical vulnerabilities accounted for 83.3% of the total number of discovered critical vulnerabilities. During Spring 2002, the 7,042 port 139 critical vulnerabilities accounted for 75.4% of all discovered critical vulnerabilities. Port 139 vulnerabilities are detailed in Section 4.1.1.

Port 161, used by SNMP, was the second most vulnerable port during both scans. All of these critical vulnerabilities were of the *information disclosure* variety. The information found from these scans was limited to the 'default' SNMP community strings of 'public' and 'private'. During Fall 2001, the 1,391 port 161 critical vulnerabilities accounted for 10.1% of the total number of discovered critical vulnerabilities. During Spring 2002, the 1,218 port 161 critical vulnerabilities accounted for 13.0% of the total number of discovered critical vulnerabilities.

Port 80, used by HTTP servers, was the third most vulnerable port during both scans. During Fall 2001, eighty different critical vulnerabilities were

discovered that involved port 80 services. During Spring 2002, the number of unique critical vulnerabilities fell to sixty-eight. Port 80 vulnerabilities are detailed in Section 4.1.2.

Port 21, used by FTP daemons, was the fourth most vulnerable port during both scans. During Fall 2001, the 137 port 21 critical vulnerabilities accounted for 1.0% of the total number of detected critical vulnerabilities. During Spring 2002, the 193 port 21 critical vulnerabilities accounted for 2.0% of the total number of discovered critical vulnerabilities. Port 21 vulnerabilities are summarized in Section 4.1.3.

Port 25, used by SMTP servers, was the fifth most vulnerable port during both scans. During Fall 2001, the 120 port 25 critical vulnerabilities accounted for slightly less than 1% of the total number of detected critical vulnerabilities. During Spring 2002, the 128 port 25 critical vulnerabilities accounted for 1.3% of the total number of discovered critical vulnerabilities. Port 25 vulnerabilities are summarized in Section 4.1.4.

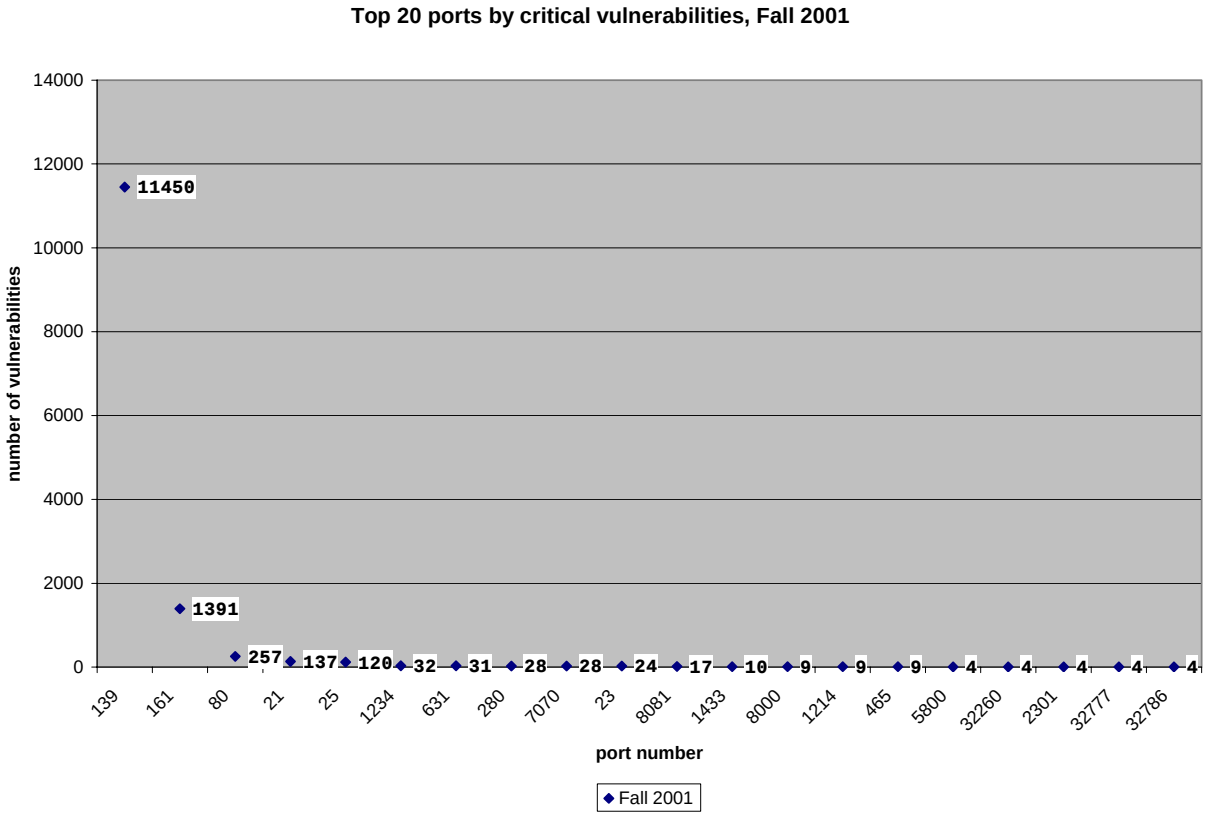

Figure 1. Top twenty ports by critical vulnerabilities, Fall 2001.

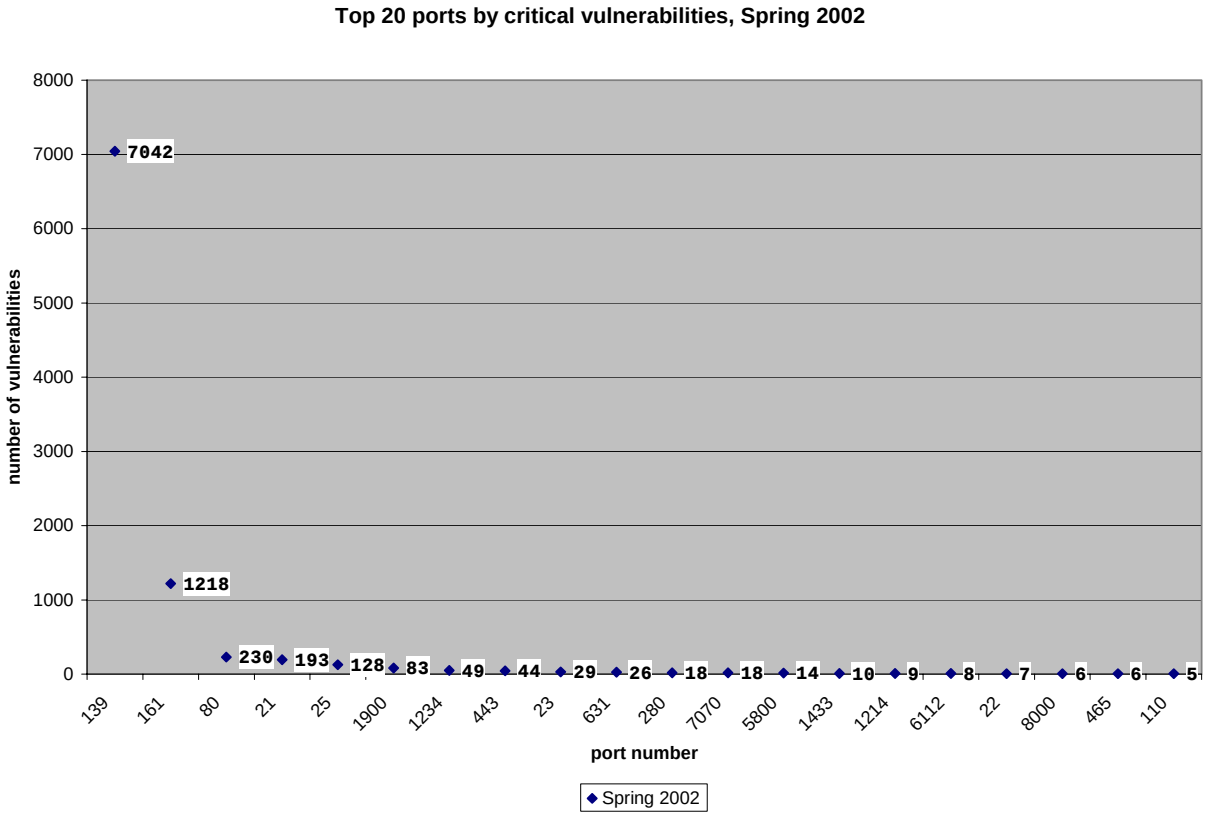

Figure 2. Top twenty ports by critical vulnerabilities, Spring 2002.

Port 1900, used by Universal Plug and Play, showed no critical vulnerabilities during the Fall 2001 scan, but exhibited eighty-three critical vulnerabilities, ranked sixth in total critical vulnerabilities during the Spring 2002 scan. This anomaly can be attributed to the discovery of a buffer overflow vulnerability in the Universal Plug and Play service disclosed December 20, 2001 as outlined in Microsoft Security Bulletin MS01-059.

Table 1. Percent change in critical vulnerabilities, Fall 2001 and Spring 2002.

port	Fall 2001	Spring 2002	% change
all	13735	9339	-32.0
139	11450	7042	-38.5
161	1391	1218	-12.4
80	257	230	-10.5
21	137	193	40.9
25	120	128	6.7
1234	32	49	53.1
631	31	26	-16.1
7070	28	18	-35.7
280	28	18	-35.7
23	24	29	20.8
8081	17	0	-100.0
1433	10	10	0.0
5800	4	14	250.0
443	3	44	1366.7
1900	0	83	N/A

4.1.1. Port 139 vulnerabilities

> *NetBIOS applications employ NetBIOS mechanisms to locate resources, establish connections, send and receive data with an application peer, and terminate connections* [RFC1001].

Windows machines use NetBIOS services for authentication and service sharing. The Windows implementation of NetBIOS uses ports 135, 137, 138 and 139. Port 139, the SMB over NetBIOS session port, was the most widely used of these four ports during the Fall 2001 and Spring 2002 scans.

A NetBIOS null session allows a machine, or a user of that machine, to gather information about another machine's users, shares, and services. Windows processes, for example, use null sessions to browse for network services and relay the findings to the system's user. In these service 'browsing' sessions, a username and password is not used. Null sessions can be considered a security

risk because of the amount of information they can provide about a victim. However, they are a necessary part of the Windows service architecture. Nessus uses null sessions to gather information about the host, including local and domain SIDs, local and domain usernames, file and printer shares and characteristic information about the local user of the host, such as group memberships.

Some of the vulnerabilities discovered through scans of port 139 are not exploitable through a network. Others require a local user account, or the ability to interact with the machine.

4.1.1.1. Privilege Elevation vulnerabilities

Nine different *privilege elevation* vulnerabilities were discovered using information collected from scans of port 139 during both semesters.

A buffer overflow vulnerability in Index Server v2 for Windows NT allows *remote* attackers to run code with system privileges. The Spring 2002 scan showed 515 machines with this vulnerability, down from 1000 machines the previous semester. The information gathered in the scan fails to show how many of these systems utilized Index Server v2. Systems running Windows NT and IIS would have had the greatest probability of running Index Server v2 since both IIS and Index Server v2 are part of Windows NT Option Pack 1 [MS01-025]. Only a single machine during this scan, *ats.etsu.edu*, used Windows NT and IIS and exhibited this vulnerability. *ats.etsu.edu* has since been decommissioned.

A vulnerability in the parsing algorithm for IIS versions 4 and 5 could allow a *remote* attacker to run code with system privileges [MS00-086]. The Spring 2002 scan showed 158 machines with this vulnerability, down from 338 machines the previous semester. None of the seventy machines that offered various IIS services exhibited this vulnerability. In all cases, this vulnerability was classified as *dormant*.

The autologon option in Windows NT based operating systems allows the administrator to be interactively logged onto the machine with a blank password. This option would allow a *local* or *remote* attacker administrative access to the machine. Two machines on campus exhibited this *active* vulnerability, but have since been reconfigured as the result of one being compromised on April 2, 2002, detailed in Section 5.1.

All *local privilege elevation* vulnerabilities are classified as *active* because any user that can interactively login to a machine can exploit these vulnerabilities. Since these vulnerabilities can not be exploited remotely, they are classified *local information disclosure*. A legitimate user of ETSU administrative network system could use this information to determine exploitable workstations. The following summarizes *local privilege elevation* vulnerabilities discovered on the ETSU administrative network:

- Exploitable flaws exist in certain local procedure call (LPC) implementations in the Windows NT and 2000 operating systems. Machines not running Windows 2000 Service Pack 2 and Windows NT machines not patched after the release of Windows NT Service Pack 6a are vulnerable (236 machines as of Spring 2002) [MS00-070].
- Windows NT and Windows 2000 machines not patched against the 'Relative Shell Path' vulnerability allow a *local* user to insert any executable to be run in place of the Windows 'explorer' shell. The privileges gained by the malicious code would reflect those of the person interacting with the machine. (236 machines as of Spring 2002) [MS00-052].
- Vulnerabilities in the 'domain account lockout' policy of Windows 2000 allow users to brute force passwords, including those of the administrator. Only machines running Windows 2000 Service Pack 1 are affected [MS00-089].

- The 'Still Image' service, normally used with scanners and digital cameras, can elevate the privilege of any local user. This service is automatically installed by Windows 2000 and runs as a service after a digital imaging device has been attached to the machine. Machines not patched with Windows 2000 Service Pack 2 are exploitable by this vulnerability (191 machines as of Spring 2002) [MS00-065].
- A defect in the Service Control Manager within Windows 2000 allows malicious users to impersonate elevated privilege services. Machines not patched with Windows 2000 Service Pack 2 are vulnerable (190 machines as of Spring 2002) [MS00-053].
- The Windows NT LAN Manager Security Support Provider allows code to be executed with operating system level rights. Windows NT machines not patched since Service Pack 6a are susceptible to this exploit (40 machines as of Spring 2002) [MS01-008].

4.1.1.2. Writeable Service vulnerabilities

Five *active writeable service* vulnerabilities were discovered on TCP port 139 during the scans of the ETSU administrative network. The four vulnerabilities most often exposed by the scan have many commonalities. The fifth, uncovered on only two machines, is inherent to Linux, an operating system that does not natively support the SMB over NetBIOS interface, but can with packages such as Samba.

The main *writeable service* vulnerabilities involve Windows registry entries that can be edited by system users that lack administrative privileges. In all cases, the registry entries were associated with programs or dynamic link libraries that execute automatically upon startup/logon. A malicious party that could edit these entries could cause a program to execute without a legitimate user's

knowledge. Spyware and backdoors are examples of programs that could exploit this vulnerability.

From the data analysis, these vulnerabilities seem to be limited to Windows NT 4. An attacker with permission to log onto a Windows NT 4 system could edit these values remotely. Most Windows NT 4 machines on the ETSU administrative network use a domain login procedure and support no local accounts, except for those installed by default. The above analysis would suggest that this vulnerability be classified as *local*. However, an unsuspecting user could execute a virus that edits these registry keys to execute the virus' payload upon startup. In this scenario, this vulnerability would be classified as *client.* In this study, this vulnerability was considered *local*.

Samba is an open source program developed for various UNIX and Linux implementations that allows UNIX and Linux machines to interface with Windows file and printer sharing services. An implementation flaw allows for random creation of files on the machine using Samba. Fortunately, the files created may only have the extension '.log', which prevents the overwriting of configuration or other important files. Two machines on the ETSU administrative network exhibited this *remote writeable service* vulnerability.

4.1.1.3. Readable Service vulnerabilities

Five readable service vulnerabilities were identified by the Fall 2001 and Spring 2002 scans of the ETSU administrative network. The most dangerous of these vulnerabilities allows for retrieval of information that could be used to gain complete control over the vulnerable machine.

- During the Spring 2002 scan, 1832 machines allowed Nessus to login remotely, either through the use of a valid username/password combination or through the use of a null session.

- 1088 machines did not have the registry properly "locked down" and were accessible to non-administrators
- 110 machines had file or print shares that were accessible to anyone on the ETSU administrative network.
- Sixty-seven machines exhibited a vulnerability that allows a password protected file or print share to be accessed using a special request containing only the first character of the password [MS00-072].

The most dangerous *readable service* vulnerability allows a *remote* user to access the Windows registry entry containing the password for the VNC service. Virtual Network Computing (VNC) is a service that allows remote desktop control similar to Symantec's pcANYWHERE, Laplink Gold or the Remote Desktop feature found in Windows XP Professional Edition. An attacker who could read this registry entry could leverage the information to take complete control of the victim machine. This vulnerability was found on ten machines during the Fall 2001 scan and seven machines during the Spring 2002 scan.

4.1.1.4. Denial of Service vulnerabilities

A buffer overflow error in the Windows 2000 infrared protocol could cause the *locally* exploitable machine to reboot [MS01-046]. A very high percentage of machines in this study (>99%) probably lack infrared capabilities, leading to a *dormant* classification. During the Spring 2002 scan, 478 machines were not patched to protect against this vulnerability, down from 922 machines the previous semester.

An implementation error in IP fragment reassembly could cause a computer to utilize 100% CPU in attempting to recreate a malformed IP packet, thus denying service to other applications using the CPU [MS00-029]. The Fall

2001 scan showed 920 machines that could be exploited using this vulnerability. The Spring 2002 scan showed 237 machines that were vulnerable to this attack.

A vulnerability in the computer browsing protocol of Windows NT and Windows 2000 machines allows for a malicious machine to become a master browser and *deny service* to those services normally indexed by a master browser [MS00-036]. The Spring 2002 scan discovered 237 machines vulnerable to this attack. During Fall 2001, 920 systems were vulnerable to this attack.

A deficiency in the "industry-standard" NetBIOS interface allows any Windows NT or Windows 2000 machine to act as a malicious WINS server, *denying service* to services on a victim machine. The malicious machine could request a victim machine to release its NETBIOS name, rendering the services on the victim machine unreachable [MS00-047]. 917 machines were vulnerable to this attack during Fall 2001. 236 were vulnerable to this attack during the Spring 2002 scan.

A remote procedure call (RPC) vulnerability in Windows 2000 machines allows a malformed RPC request to crash the service, thereby *denying access* to the RPC service. Although all Windows 2000 machines are susceptible to this vulnerability, machines running Windows 2000 Server are at the greatest risk [MS00-066]. Only four machines of the 190 marked 'vulnerable' were fingerprinted as running Windows 2000 Server, and categorized as machines for which this vulnerability was *active*.

A second well-known RPC vulnerability allows an attacker to remove a victim machine from a domain, leaving it unable to attach to domain resources. The Microsoft Security Bulletin MS00-062 also states an exploited domain controller would not allow domain logins, disallowing all domain users from domain resources. All versions of Windows 2000 Gold are vulnerable except Windows 2000 Datacenter Server. Machines patched with Windows 2000 Service Pack 1 are immune to this exploit [MS00-062]. As of the Spring 2002 scan, 111 machines were not patched against this vulnerability. Two of these 111

machines used Windows 2000 Server: neither, however, was a Windows 2000 Datacenter Server system.

A third RPC vulnerability *denial of service* vulnerability affects only Windows NT Server based machines. Sending a malformed request to the service causes the service to halt. Machines not patched since the release of Windows NT Service Pack 6a are vulnerable to this exploit [MS01-048]. Of the 48 systems that showed evidence of this vulnerability during the Spring 2002 scan, eleven use Windows NT Server.

Incomplete TCP/IP packets sent to port 139 of vulnerable Windows 9x or Windows NT machine could cause the victim machine to stop responding to all network traffic. The Windows 9x machines are only vulnerable if file and printer sharing is enabled. Windows NT machines are vulnerable if the 'server' service is running, which is *active* by default [MS00-091]. All forty systems found with this potential vulnerability during the Spring 2002 scan used a variant of Windows NT.

A programming error in the Windows NT Point-To-Point Tunneling Protocol (PPTP) causes a memory leak that could be used to exhaust available memory on the victim machine, rendering all services useless [MS01-009]. Of the forty machines with this potential vulnerability, none are believed to offer any PPTP services. Since this vulnerability is not exploitable on any machines attached to the ETSU administrative network, this vulnerability was classified as *dormant*.

Windows NT Server and Windows NT Terminal Server are vulnerable to a *local* exploit that could cause access to all services to be denied. The vulnerability allows the malicious user to dominate the usage of the TCP/IP stack, making it unusable by other services [MS01-003]. Two of the forty machines that were not patched against this vulnerability use Windows NT Server.

The scan detected exactly one vulnerability in the Fall 2001 scan missing during the Spring 2002 scan. This vulnerability, when exploited, causes a domain controller to devote a large percentage of its CPU time to processing misinformation. During the Fall 2001 scan, 757 machines were potentially

vulnerable, but none of these 757 machines were domain controllers. This vulnerability was not discovered during the Spring 2002 scan because the Microsoft Security Bulletin that originally disclosed information about this vulnerability was superseded by another Microsoft Security Bulletin, which caused this plug-in to become obsolete [MS01-011].

4.1.1.5. Information Disclosure vulnerabilities

The following information disclosure vulnerabilities were detected during the Spring 2002 scan.

- 1,681 machines disclosed information about their native LAN Manager, operating system and workgroup/domain.
- 1,671 machines disclosed their domain SID.
- 1,579 machines disclosed users of the domain when presented with their acquired domain SID.
- 1,150 machines allowed remote registry access.
- 348 machines disclosed their browse list of other hosts on the network.
- 299 machines disclosed their host SID.
- 297 machines disclosed users of the host when presented with their acquired host SID.
- 198 machines disclosed all of their available file and printer shares.
- 191 machines allowed for retrieval of a hashed password through a *client* vulnerability for an offline brute-force attack [MS00-067].
- 191 machines were Windows 2000 machines not patched with Service Pack 2.
- 168 machines disclosed all services currently running on the machine.
- Eight machines disclosed they were a primary domain controller or a backup domain controller.

- Six machines were Windows NT machines not patched with Service Pack 6a.

4.1.1.6. Port 139 Vulnerabilities summary

Table 2. Summary of port 139 vulnerabilities by classification.

	Active & Remote	*Active & Local*	*Dormant & Remote*	*Dormant & Local*
Privilege elevation	2	849	673	0
Writeable service	511	0	0	0
Readable service	3104	0	0	0
Denial of Service	861	0	278	518
Information Disclosure	7787	0	0	0

The most dangerous vulnerabilities are classified as *active, remote* and *privilege elevation*. Two of these vulnerabilities existed in over 3,900 machines scanned during the Spring 2002 semester. One of these vulnerabilities was leveraged on April 2, 2002 as described in Section 5.1. Both machines with these potentially destructive vulnerabilities were administered by the same person, and their vulnerabilities have since been remedied.

4.1.2. Port 80 vulnerabilities

Port 80 is used by the HTTP protocol, mainly in conjunction with web access. Many different web server software packages are in use on the Internet, but in an environment where a very large percentage of the systems use the Windows operating system, it can be assumed that Internet Information Services (IIS) is the most widely used web server.

Correlations between the operating system in use and the version of IIS used were used -- Windows NT uses IIS version 4 and Windows 2000 uses IIS version 5 -- to determine false positives in the collected data. In certain cases, a vulnerability may have been detected by reading the host registry and determining that a patch had not been applied. This vulnerability may only be exploitable through certain versions of IIS.

4.1.2.1. Privilege Elevation vulnerabilities

Forty-six different *privilege elevation* vulnerabilities were discovered using information collected from scans of port 80 during both semesters. Only the top six vulnerabilities are reviewed, since a possible anomaly in Nessus' analysis of a host (see Section 4.1.2.6) may have increased the number of reported vulnerabilities. All data in this section and the following section include the data recovered and categorized as a possible anomaly.

A buffer overflow vulnerability within the Windows Indexing Service used by IIS allows *remote* attackers to run code with system privileges [MS01-033]. The Spring 2002 scan showed fifty-four machines with this vulnerability, up from forty machines the previous semester. This vulnerability was the one exploited by the Code Red worm, launched July 12, 2001. In the Fall 2001 scan, seven machines exhibited remnants of a Code Red infection, proving this can be classified as an *active* and *remote* vulnerability.

Forty-six machines exhibited a buffer overflow vulnerability in the Internet Printing Protocol, installed and activated by default in IIS 5. This vulnerability is technically similar to the previously discussed Windows Indexing Service vulnerability, except the exploit uses a different protocol. With the transition to Windows 2000 from Windows NT, the number of machines that exhibited this vulnerability nearly doubled from twenty-seven in the Fall 2001 semester. All machines that exhibited this vulnerability reported as a Windows 2000 Machine with port 80 available. This vulnerability was classified as *active* and *remote*.

A component of the Microsoft Data Access Components contains a vulnerability that allows code execution, and the ability to read secured files [MS99-025]. Only IIS 3 and 4 are vulnerable to this exploit, and of the thirty-one machines labeled by Nessus to be vulnerable, eleven were using Windows NT and IIS 4.

Social engineering is needed to fully exploit a cross-site scripting vulnerability that plagued nineteen machines during the Fall 2001 scan. In cross-site scripting, a user of malicious site B is tricked into thinking they are browsing desired site A. Data entered on site B is passed to site A, along with malicious code that is unchecked by site A and executed on the users client machine [Securiteam]. Since this vulnerability takes such elaborate engineering and is not a readily available exploit, it was classified as *dormant* and *client*.

Microsoft offers a tool that hardens an IIS installation that "[turns] off unnecessary features, thus reducing attack surface available to attackers." [Microsoft1] Fifteen machines still exhibited services normally disabled by the IIS lockdown tool. This number is up from seven machines with this vulnerability during Fall 2001. Since vulnerabilities were found with these services, this vulnerability was classified *active* and *remote*.

A CGI script associated with SQL administration allows an attacker to send arbitrary commands to the script's host system because a variable the script accepts is not checked for correct syntax. To execute this exploit, the attacker

must have administrative access to the SQL server. However, there is a well-known exploit in which the default administrator account for the SQL server, the 'sa' account, is created with a blank password. A combination of a blank 'sa' password and this CGI makes the system vulnerable to a *remote* exploit. Since none of the systems checked had a blank 'sa' password this vulnerability was classified as *dormant*. Nine systems during the Spring 2002 scan showed the availability of this CGI, down from fourteen the previous semester.

Findings concerning other privilege elevation vulnerabilities and those listed above are summarized in Section 4.1.2.7.

4.1.2.2. Writeable Service vulnerabilities

Eight different *writeable service* vulnerabilities were discovered during the Spring 2002 scan. Only the top two of these vulnerabilities are reviewed because of the Nessus anomaly.

A web-accessible executable for IIS version 3 named 'newdsn' allows an attacker to create arbitrary files on the vulnerable host. The 'newdsn' script, like any web-accessible sample executable or script, should be removed in a production environment. Removing vulnerable sample applications is one purpose of the IIS Lockdown Tool discussed in the previous section. Five systems during the Spring 2002 scan were potentially vulnerable to this exploit. Since three of these systems used the Windows NT 4 operating system, this may be classified as an *active* and *remote* vulnerability [xforce3].

A Sambar web server allows an attacker to use a Perl script for sending e-mail without proper credentials. This was classified as a *writeable service* vulnerability because it allows the attacker to use the service in an undesired manner, but does not allow for an elevation of privileges. Because none of the servers involved truly used the Sambar web server, this has been classified as a *dormant remote* vulnerability.

4.1.2.3. Readable Service vulnerabilities

Sixteen different *readable service* vulnerabilities were discovered during the Spring 2002 scan, of which only the top three vulnerabilities are reviewed.

IIS versions 4 and 5 include an administration page that allows any user to change their domain password by entering the account name and current password. This page and its password change option may be accessed repeatedly without the attacker being locked out. Therefore, the attacker could be able to discern a password by brute force. Twenty systems, all using the Windows NT4 or Windows 2000 server operating systems, had this administration page available to the anonymous browser. Twelve systems had this administration page available during the Fall 2001 scan. This vulnerability was classified as *active* and *remote*.

Twelve systems running IIS had a sample application installed with a vulnerability that allows an attacker to list the contents of any directory on a server; nine more than the previous semester. This sample application would normally be removed when the IIS lockdown tool was used to harden the web service. This vulnerability was also classified as *active* and *remote*.

Four systems running IIS were configured to support unrestricted administrative access from remote websites. This allows attackers that know or can decipher the username and password of an IIS administrator to use the administration applications. Since these administration pages can be reached from *remote* machines, and the URLs of these pages are well known, this was classified as an *active* vulnerability.

4.1.2.4. Denial of Service vulnerabilities

Six vulnerabilities, when exploited, would deny access to a system service. Two of these are examined in this section. The remaining vulnerabilities are not

reviewed because of the possible anomaly that may have skewed the count of total vulnerabilities.

The first vulnerability was labeled by Nessus as a denial of service vulnerability, but the description given by Nessus was too vague for further analysis:

> *It may be possible to make the web server execute arbitrary code or crash by sending it a too authorization. Risk factor: High*

Of the seventeen devices listed with this vulnerability, fifteen of them were a variety of print device. The remaining two were MacOS platforms. One of these two systems was a known web server. The other system, whose IP address did not resolve to a hostname, was probably not an advertised web server. Since this vulnerability could not be studied, but was reported by Nessus, it was categorized as a *remote, dormant* vulnerability.

A vulnerability discovered on four systems causes a denial of service to the web server when exploited.

> *A denial of service vulnerability (exists) that could enable an attacker to temporarily disrupt service on an IIS 5.0 web server. WebDAV doesn't correctly handle particular type of very long, invalid request. Such a request would cause the IIS 5.0 service to fail* [MS01-044].

This vulnerability can be categorized as an *active, remote* vulnerability.

4.1.2.5. Information Disclosure vulnerabilities

The following information about machines on the ETSU administrative network was determined during the Spring 2002 scan.

- 174 machines advertised their web server software and version number.
- Fifty-three machines used Microsoft FrontPage Extensions.
- Twenty-one machines had FrontPage related, world-readable files that contain server configuration information.
- Eighteen machines had a robots.txt file that contains information about the web site directory structure.
- Fourteen machines used a version of Perl that returns the directory of the virtual web server when confronted with a request to a non-existent Perl script.
- Ten machines using Apache would have disclose legitimate usernames of system users.
- Four machines offered an IIS administration file which would have allowed anyone to view the directory structure of the machine.
- Three machines used a CGI script that returns environment variables

4.1.2.6. Port 80 anomalies

Data from security audits should be verified when possible since tools like Nessus may generate false positives or fail to detect existing vulnerabilities. The author of this study lacked the authority to verify the data reported from Nessus and relied largely on the tool's ability to validate the data, and the plausibility of the data. One implausible situation involved a system named *forbesm.etsu.edu* that allegedly exhibited forty-five different port 80 vulnerabilities. In many cases, this system was the only system to display any of these vulnerabilities. The

validity of the scan was investigated by attempting to exploit some of the *readable service* vulnerabilities using information returned by Nessus plug-ins. None of the attempted exploits resulted in a positive result. Although it could not be determined if all detected vulnerabilities were false positives for reasons of web server integrity, it seemed 'likely' that most (>90%) of the Nessus reported data on this system was false.

Since no other anomalies were discovered during the analysis phase of this study, the author regards all other data from Nessus as reliable.

4.1.2.7. Port 80 Vulnerabilities Summary

Table 3. Summary of port 80 total vulnerabilities by classification.

	Active & Remote	*Active & Local*	*Dormant & Remote*	*Dormant & Local*
Privilege elevation	202	0	49	0
Writeable service	28	0	5	0
Readable service	55	0	0	0
Denial of Service	8	0	17	0
Information Disclosure	314	0	0	0

During the Fall 2001 scan Nessus discovered 164 critical vulnerabilities, 171 warnings and 174 informational discoveries that have been categorized above.

4.1.3. Port 21 Vulnerabilities

Port 21, is the TCP port most widely used by File Transfer Protocol (FTP). During the Spring 2002 scan, 189 devices had services available on port 21. Eleven different vulnerabilities were discovered, all of the *remote* and *active*

variety. Sixty-nine of the vulnerabilities allowed for some type of privilege elevation. No vulnerabilities were discovered that allowed an attacker to upload files to arbitrary locations on the FTP site. Eighty-four vulnerabilities were discovered that allowed anonymous users to browse the FTP site. No vulnerabilities that would disclose information about the system were discovered.

The most notable of these vulnerabilities were the eighty-four vulnerabilities related to anonymous FTP access. Nmap operating system fingerprint data showed all of the devices to be IP printers or print server devices. No file data is stored on any of these devices. It is unknown what the FTP services on these machines are used for. These systems probably receive print data on port 21, or use this port for downloading firmware or software systems upgrades.

In fifteen cases, a vulnerability in the Washington University FTP daemon that is found on many Linux distributions allows an attacker with a valid account or anonymous access to exploit the vulnerability to run arbitrary code as root. This vulnerability was found on seven Linux systems, five AIX systems and one Solaris system. Two print devices were also listed as having this vulnerability. None of the systems with the vulnerability allowed anonymous access. If it was assumed that the FTP servers had at least one standard user on them, then the server could be considered *active* against this *remote* exploit [xforce1].

Table 4. Summary of Port 21 vulnerabilities by classification.

	Active & Remote	*Active & Local*	*Dormant & Remote*	*Dormant & Local*
Privilege elevation	69	0	0	0
Writeable service	0	0	0	0
Readable service	84	0	0	0
Denial of Service	42	0	0	0
Information Disclosure	0	0	0	0

4.1.4. Port 25 vulnerabilities

Port 25 is most widely used by the Simple Mail Transport Protocol (SMTP) for sending electronic mail. During the Spring 2002 scan, seventy-two devices had services available on port 25. Twelve of the vulnerabilities allowed for an elevation in privileges. Eleven of these vulnerabilities, however, required an interactive session with the system. Thirty-one vulnerabilities allowed for an attacker to use the service for *writing*. Fifty-two vulnerabilities allowed for information disclosure.

The single *remote, active* privilege elevation vulnerability was discovered on an AIX system. The Nessus plug-in information states that a HELO command with an argument over 12,000 characters causes a buffer overflow that can allow arbitrary code execution. The remainder of the privilege elevation vulnerabilities can only be exploited from the local host since the buffer overflow that causes the vulnerability involves the processing of the sendmail '–bt' switch argument.

The writable service vulnerability discovered allows the attacker to write arbitrary files to the SMTP server system by forging the receipt field with an absolute file path and file name. The SMTP server does not validate the

information in this fieldreceipts. The Nessus plug-in warns that this information could be a false positive because of the ways in which different mail servers react to such a request. Some servers respond positively, while others may ignore the message without notifying the sender. To verify the validity of each reported vulnerability would require file system access.

The information disclosure vulnerabilities were "banner" responses that allow the attacker to retrieve the name and sometimes the version of the mail server being used. The attacker could use this information to see if well-known vulnerabilities exist for the server being examined.

Table 5. Summary of port 25 vulnerabilities by classification.

	Active & Remote	*Active & Local*	*Dormant & Remote*	*Dormant & Local*
Privilege elevation	1	11	0	0
Writeable service	31	0	0	0
Readable service	0	0	0	0
Denial of Service	0	0	0	0
Information Disclosure	52	0	0	0

4.2. Spring 2004 and Spring 2002

Since the initial scan during the Fall 2001 and Spring 2002 semesters, and the follow-up scan during the Spring 2004 semester, there was one major change to the ETSU administrative network. A firewall protecting the entire campus administrative network from the Internet was implemented. Almost all (>99%) of the 65,535 TCP ports are blocked globally, and requests for open ports in the firewall to a single host must be registered. This change has helped reduce the vulnerabilitiy of the campus network to outside attacks. However, if

an attacker has availability to the ETSU administrative network, their attacks will not be stopped by a firewall or intrusion detection system.

The administrative network's vulnerability to internal attack was exposed in 2003 when an ETSU employee inadvertently plugged a laptop infected with the MSBlast worm into the ETSU administrative network. Since ETSU does not automate their management of patching systems, the worm penetrated a significant portion of the network, disrupting services, and, at its peak, passing more than 50,000 infected messages a day through the ETSU mail server.

A final partial sweep of the ETSU administrative network conducted during the Spring semester of 2004 scanned 909 of the estimated 3,600 hosts. This scan was expected to show similar numbers in the number of critical vulnerabilities per host, and the number of critical vulnerabilities per port. Once the analysis of the data was completed, it was discovered that the number of critical vulnerabilities per host increased, whereas the top three vulnerable ports registered zero critical vulnerabilities during the Spring 2002 audit.

4.2.1. Spring 2004 Summary

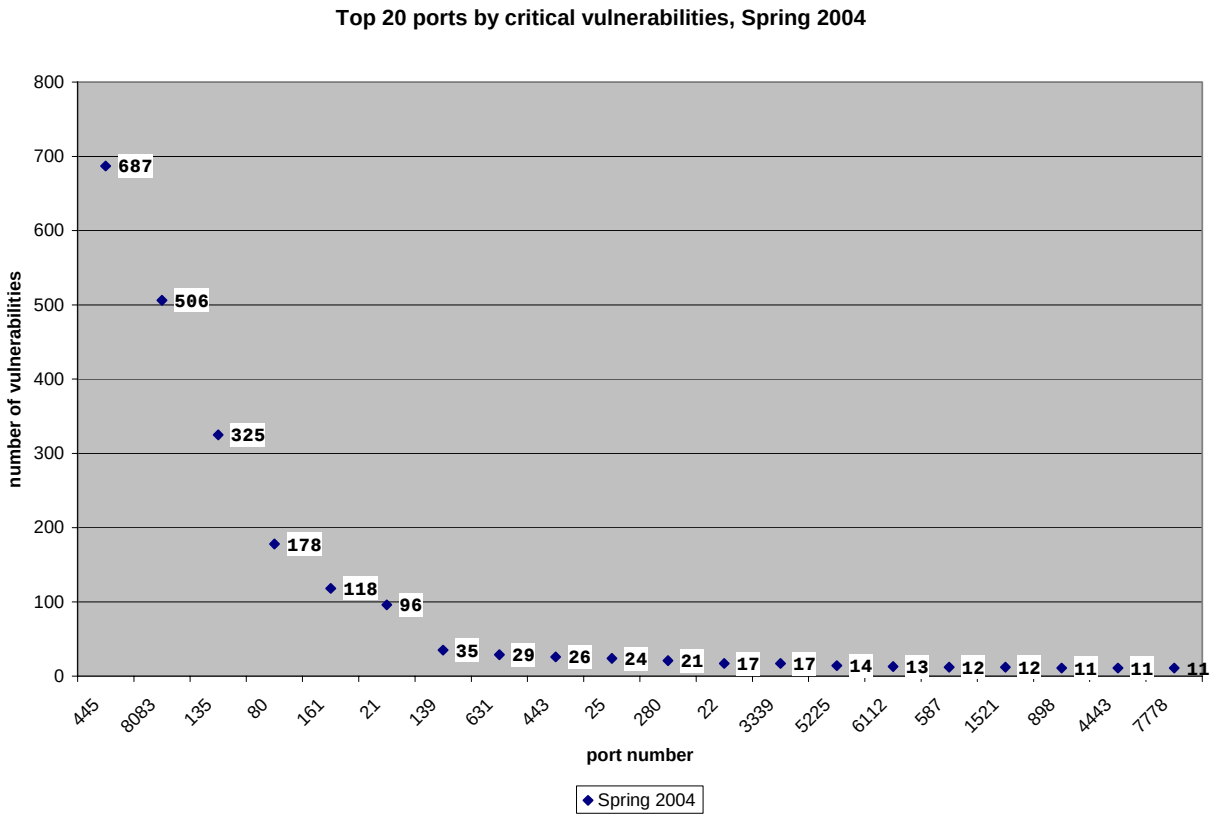

Figure 3. Top twenty ports by critical vulnerabilities, Spring 2004.

Port 445, the port most widely used by SMB over TCP/IP, exhibited the most vulnerabilities during the Spring 2004 semester. Port 8083 vulnerabilities which numbered 506 during the Spring 2004 semester are discussed in the following section. The NetBIOS RPC end point mapper, port 135, rose to the third most vulnerable port. The next four most vulnerable ports were all in the top five most vulnerable during the 2002 scan: 80 (HTTP), 161 (SNMP), 21 (FTP), and SMB over NetBIOS (139).

4.2.1.1. Port 8083

Port 8083 is a non-standard port chosen for use by the McAfee Framework Service. ETSU uses this service to deploy and manage the McAfee VirusScan Enterprise product on ETSU workstations using ePolicy Orchestrator as its centralized console. Five different vulnerabilities were discovered during the Spring 2004 scan.

One of these port 8083 vulnerabilities was related to a buffer overflow in a .dll used by FrontPage Extensions. This vulnerability was only discovered on three machines using a Windows operating system. It is hypothesized that the systems in question used another software package that also used port 8083.

A second port 8083 vulnerability can be exploited utilizing an unchecked buffer in one of the core operating system components in Windows 2000 and Windows XP. To exploit this vulnerability, the vulnerable system must be using IIS version 5 or 6. This vulnerability allows the attacker to run arbitrary code. Only one Windows XP machine was located with this vulnerability [MS03-007].

The remaining three vulnerabilities involve a directory traversal vulnerability. To exploit this vulnerability, a client sends a correctly formulated request to a server that contains character sequences normally reserved for changing directories, such as '../'. Hosts that are vulnerable to this exploit allow these requests to access directories outside the scope of the available web root. The exploit allows an attacker to access directories normally denied access to anonymous web browsers.

Through testing, it was determined that the ePolicy Orchestrator Agent (ePOAgent) client is susceptible to a simple directory traversal exploit using '.../' as the character sequence to access a parent directory. The ETSU configuration for this service places the home directory for these services six folders below the root folder of the system partition. The attacker can retrieve a file by forming a request that begins with 'http//hostname:8083/.../.../.../.../.../.../' and

appending the directories and name of a known any file. The files available to the attacker are any files that are not hidden and readable by the NTFS 'everyone' group. This exploit can only read the system partition, and can not access other partitions. Also, the exact name and location of the file must be known since this attack method does not offer directory listings. A quick Internet search uncovered no information regarding this vulnerability. One listing on the ISS X-Force database had an entry that was similar to the exploit described above:

> *McAfee AsaP VirusScan is a Web-based AntiVirus service for remote clients that uses the myCIO HTTP server to transfer virus definitions. McAfee AsaP VirusScan could allow a remote attacker to traverse directories on the Web server. A remote attacker can send a specially-crafted URL request containing modified "dot dot" sequences that use three dots instead of two (/.../) to traverse directories and view or download any file outside of the Web root directory.* [xforce2]

Although developed by the same company, these two products are marketed separately.

At the time of this writing it was unclear if Network Associates Incorporated was notified about this vulnerability in their enterprise level virus protection product.

4.2.2. Spring 2004 and Spring 2002 comparison

To compensate for the Spring 2004 semester partial scan, the number of vulnerabilities per port was adjusted by a common multiplier (hosts scanned

during Spring 2002 / hosts scanned during Spring 2004) to estimate the total number of detectable vulnerabilities.

Table 6. Percent change in critical vulnerabilities, Spring 2002 and Spring 2004.

port	Spring 2002	Spring 2004	Spring 2004 adjusted (x4.309)	difference (adjusted)	% change (adjusted)
all	9311	3159	13613	4302	46.2
445	0	687	2960	2960	N/A
8083	0	506	2180	2180	N/A
135	0	325	1400	1400	N/A
80	202	178	767	565	279.7
161	1218	118	508	-710	-58.3
21	193	96	414	221	114.3
139	7042	35	151	-6891	-97.9
631	26	29	125	99	380.6
443	44	26	112	68	154.6
25	128	24	103	-25	-19.2
280	18	21	90	72	402.7
22	7	17	73	66	946.5
3339	0	17	73	73	N/A
5225	0	14	60	60	N/A
6112	8	13	56	48	600.2

The number of vulnerabilities increased 46.2% between the time period of Spring 2002 and Spring 2004. The number of vulnerabilities during the Spring 2004 scan was at a level similar to Fall 2001.

The order of the most vulnerable ports between the two scans changed significantly. Port 139 was the most vulnerable port in the 2002 scan showing 7,042 discovered abilities. In the Spring 2004 scan, it ranked seventh, showing only 151 vulnerabilities. The most vulnerable port during the most recent scan was port 445, which had a total of 2,960 vulnerabilities. During the second scan, this port registered zero vulnerabilities. This change can be attributed to a change in the SMB communications architecture starting in Windows 2000. SMB is an acronym for 'server message block' and is described as "a message

format used by DOS and Windows to share files, directories and devices" [Webopedia]. In versions of Windows previous to Windows 2000, SMB was partnered with the NetBIOS protocol for communication purposes. This communication used TCP port 139. Since Windows 2000, NetBIOS was no longer used to transmit SMB information; the server message blocks were transmitted using the TCP/IP protocol. Nessus data indicated that only systems running the Windows 9x operating systems and some Linux systems, assumed to be running Samba, have vulnerabilities on port 135. All machines that reported vulnerabilities on port 445 were either Windows 2000 professional or Windows XP.

Port 135, used by the Windows RPC end point mapper, was the third most vulnerable port during the Spring 2004 scan. Remote Procedure Call is a protocol that supports the remote execution of code on a host system. Port 135 is consistently at the top of the most attacked ports, as reported by the Internet Storm Center, a service run by the SANS Institute that receives and analyzes logs from millions of intrusion detection systems. The MS03-026 Microsoft security advisory warned of a buffer overflow in the RPC interface that could allow arbitrary code execution [MS03-026]. This vulnerability was exploited by the MSBlast worm. It is theorized that the steadily decreasing level of port 135 attacks from unique sources is related to the discovery and cleansing of this worm.

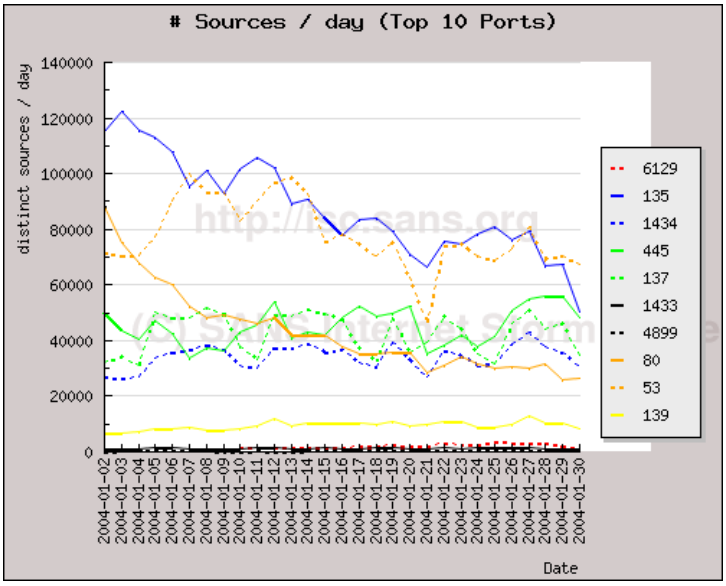

Figure 4. Top attacked ports per day by number of distinct attack sources. (source: Internet Storm Center – http://www.incidents.org, January 30, 2004)

Port 80 vulnerabilities increased by 279.7% from Spring 2002 to Spring 2004. The vulnerabilities were detected in systems running various Windows operating systems, Linux operating systems and print devices. Sixty-two different vulnerabilities were discovered on the 89 machines that reported a port 80 vulnerability. The class C subnet that contains the main campus web servers was not scanned during this third scan, and therefore the 178 vulnerabilities Nessus discovered during the Spring 2004 scan were on systems running possible unauthorized services.

SNMP information disclosure vulnerabilities dropped by 58.3% between the Spring 2002 and Spring 2004 scans. FTP vulnerabilities increased 114.3% during the two year span.

The number of average critical vulnerabilities, warnings and informational discoveries per host found by Nessus increased between the Spring 2002 and Spring 2004 semesters. The average number of critical vulnerabilities per host rose 46.1% from 2.377 average critical vulnerabilities to 3.475 average critical vulnerabilities. The average number of warnings per host rose 46.1% from 6.682

average warnings to 9.764 average warnings. The average number of information discoveries per host rose 16.3% from 8.832 average informational discoveries to 10.273 average informational discoveries.

Table 7. Average vulnerabilities by type, Fall 2001, Spring 2002 and Spring 2004.

	Fall 2001	*Spring 2002*	*Spring 2004*	*Total*
critical vulnerabilities	13642	9311	3159	26112
avg. critical / host	3.694	2.377	3.475	3.065
warnings	18588	26176	8876	53640
avg. warn. / host	5.033	6.682	9.764	6.043
informational disclosure	30828	34597	9339	74764
avg. info / host	8.347	8.832	10.273	8.776

Chapter 5
Conclusion

5.1. Conclusions

Unpatched Windows operating system vulnerabilities are the top threat to security on the ETSU administrative network. Port 139 critical vulnerabilities accounted for 80.1% of all critical vulnerabilities discovered during the initial two scans. The major problem facing campus security is the absence of a patch management and deployment system for the over 3,000 Windows based computer systems. Under the current security model, users are responsible for applying operating system and application patches to their workstation. Attempts for the Office of Information Technology to automate patch distribution to workstations have been squelched by members of the faculty community. Concerns exist that automated patch installation may disable specialized programs that are installed and used on faculty workstations.

Allowing 'academic freedom' grants liberties to workstation users at ETSU that do not follow preferred security guidelines. Users are made a local administrator on their primary workstation and may install any program, or run any executable. Users commonly make unwise choices such as installing programs laced with spyware or running unknown executables received through e-mail. The network is also open to anyone with access to a network jack, and systems that are not members of the ETSU domain can receive an IP address and access ETSU resources and the Internet. The ETSU firewall protects against exploits by outside sources, but when an infected source enters the ETSU network, the probability that the network could suffer a large number of compromises increases.

In some cases, server class operating systems are installed on workstations and server class hardware not administered by the Office of Information

Technology. On April 2, 2002, a Windows NT server was compromised and turned into a web server of pornographic material. The intruders were able to compromise this Windows NT Server because the administrator password was left blank. On April 4, 2002, a Windows 2000 machine was compromised and turned into a 'warez' server holding applications, games and pornographic movies. The administrator password on this machine was also left blank. On April 14, 2002, a machine was used as a mail server to send out e-mails advertising pornography. The *network readable* SMTP *service* allowed anonymous relaying of e-mail and was used by the intruders to send over 23,000 solicitations. The machine was configured to offload the duty of sending the mail to the former ETSU mail server, *access.etsu.edu*. Under the unusual load of outgoing e-mail, and the 13,000 solicitations that were 'bounced back,' *access* folded and *denied service* to ETSU faculty, staff and students.

Systems such as *ats.etsu.edu*, mentioned in Section 4.1.1.1, and the three systems mentioned here illustrate how allowing university employees to administer their workstations or servers has decreased the security on the ETSU administrative network. Changing this policy of open administration would be difficult and may not be desired in an educational environment. Security policies set by an information technology department, foremost, should not interfere with this educational mission. However, until security policies are developed and enforced, it is theorized that the number of vulnerabilities will remain consistent with the data discovered in this study.

Continued security audits at ETSU would not be practical until a system for patch deployment was implemented. The analysis found most of the critical vulnerabilities (>75%) were related to Windows operating systems being unpatched or misconfigured. Without a patch deployment system to create a security baseline, an audit would only discover the obvious; that operating system patches were missing on many systems.

5.2. Avenues for Research

5.2.1. ETSU Domain Password Policy

East Tennessee State University does not enforce a strict password policy on the ETSU Windows 2000 Active Directory Domain. The current policy allows for any password that is at least six characters in length and may consist of any combinations of letters, numbers or symbols. A user's password must be changed once every ninety days, and the same password can not be used twice in succession. A possible extension of this thesis would involve using the LC4 program described in the following section to analyze the passwords used on the ETSU domain to determine the exploitability or 'crackability' or these passwords. From this information, it could be determined if a stricter password policy for the domain should be suggested.

5.2.1.1 LC4

LC4 is a Windows-based utility that evaluates the strength of Windows user account passwords. Password strength is determined by the time it takes a third party to guess or decipher a user password. A password easy for an attacker to guess or decipher increases network vulnerability. Weak passwords are generally common words, names of people or pets, birthdays, and passwords six characters or less in length. Strong passwords are generally eight characters or longer in length, and contain a combination of letters, numbers and symbols. LC4 can only decipher passwords used in Windows NT/2000 network environments [LC4].

LC4 first must gather the encrypted passwords before the (sometimes long) deciphering process. LC4 can retrieve encrypted passwords in from a local registry; from a remote registry; from a SAM file; and through sniffing.

Administrative rights are required for many encrypted password retrieval tasks, for obvious security reasons.

LC4 can 'dump' passwords from the local machine registry. LC4 will not retrieve locally stored password hashes if a non-privileged user makes the 'dump' request.

Encrypted passwords may also be retrieved from a remote computer registry. The user requesting the retrieval must have administrative rights on the host machine. To retrieve remote password hashes, the host machine must allow remote registry access also. If the remote machine protects the encrypted passwords with Microsoft SYSKEY (Windows NT 4.0 SP3 and later), a utility exists that can be used to bypass SYSKEY encryption and retrieve the encrypted passwords for use by LC4 [LC4].

Passwords can be retrieved from a 'SAM file.' The SAM file is a database of encrypted passwords held by the Security Account Manager of Windows NT based systems [Microsoft3]. A SAM file can only be retrieved in specific ways since the Security Account Manager holds a lock on the SAM file. The first way to retrieve a SAM file involves booting the system from a floppy disk that contains DOS or a DOS-based version of Windows. On this boot disk, a utility to view NTFS partitions under DOS must exist. The SAM file may be copied since the SAM file is not locked in DOS. Second, Windows NT 4 systems store the SAM file on repair disks and the WINNT\repair directory. The SAM file, lastly, can be recovered from a backup tape [LC4].

The final method of password retrieval is 'sniffing' and collecting network packets that contain encrypted passwords. Sniffing network packets with LC4 requires a packet capture driver to be installed. A computer on an unswitched network will receive packets destined for all systems attached to a common hub, section of network or broadcast domain. The network interface device normally discards packets destined for other machines. A packet capture driver accepts all packets received by the network interface device [LC4].

LC4 attempts to decipher a password using a four phase algorithm [LC4]:

- Test the password for equality with the username.
- Test the password against common words or alphanumeric combinations. The newest release of LC4 contains a smaller dictionary of approximately twenty-five thousand words and a larger dictionary of approximately two-hundred fifty thousand words.
- Test the password against combinations of dictionary words with different combinations of numeric and symbolic characters. The default setting appends two non-alphabetic characters to the end of each dictionary word. The 'hybrid' phase deciphers such passwords as 'apple50' and 'secret!!'.
- Attempt a brute force attempt of all character combinations. The brute force phase of decryption will take the longest, in some cases in excess of twenty-four or forty-eight hours.

Password strength can be measured by the time needed to decipher a password. LC4 lists the audit time for each deciphered password. If desired, the deciphered password may remain hidden [LC4].

Windows NT passwords can be brute-forced by LC4 in two pieces. Windows NT has a password character-length limit of fourteen characters. The encryption methods employed by Windows NT allow the password to be decrypted in two seven-character segments. Decrypting two seven-character password fragments is far faster than decrypting one fourteen-character password. Windows 2000 sets no limit on password length. Passwords fifteen characters or longer must be deciphered in full [LC4].

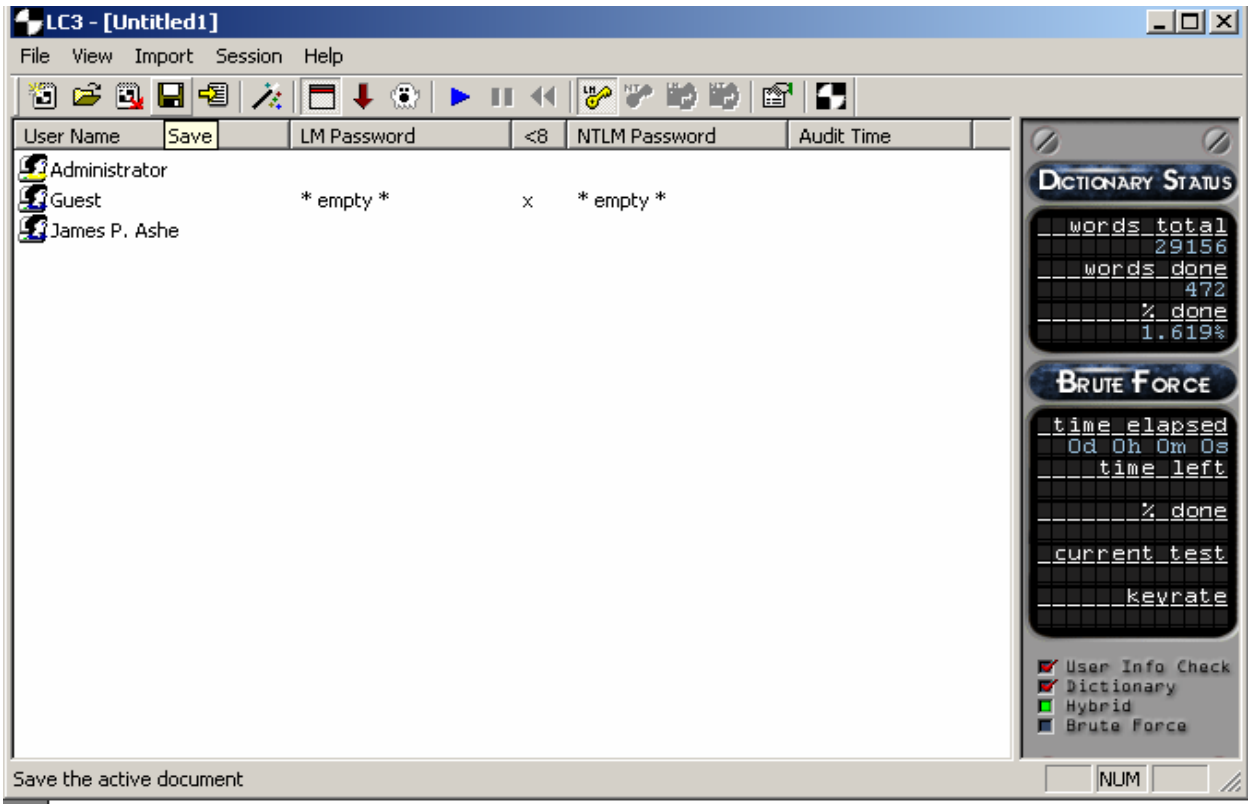

Figure 5. LC4 (not reporting cracked password).

LC4 could be used to determine the average strength of ETSU user passwords. With the absence of a password policy, a large percentage of user passwords (30%) are expected to be deciphered in the first three deciphering phases.

5.2.2. Software Update Services

East Tennessee State University does not use a software package for the automated delivery of system patches to Windows workstations and servers on the ETSU administrative network. One such software package for delivering system patches is Microsoft Software Update Services. Software Update Services (SUS) is a tool that allows domain administrators to control the 'Automatic

Updates' control panel object of Windows 2000, Windows XP and Windows Server 2003 systems on their domain.

A Software Update Services server is deployed on a site to be used as the repository for all workstations and servers use to receive critical updates. The critical updates are cached locally on the SUS server through a synchronization process with windowsupdate.microsoft.com. Once the update is downloaded and accepted for distribution by the administrator, the 'Automatic Updates' object within the client operating system is directed to process the update through the SUS server.

An active directory group policy controls how the Automatic Updates client processes updates. SUS supports for three options for managing updates. The first option notifies the user of a patch's availability and requires user intervention to both download the update from the SUS server and to install the patch. The second option downloads the patch automatically to the client system, but will not attempt to install it without user intervention. The third choice forces the Automatic Updates client to automatically download and install the update. If the final choice is chosen, the administrator chooses the time at which the patch installation occurs. Supplementary options exist that allow for an administrator to handle situations where a system was powered down during a scheduled installation time, and situations where a user is interactively logged into a machine during a scheduled installation time.

Currently, a Software Update Services server is being tested at ETSU for use on the administrative network. However, global use of the server has been denied by committees that are concerned about the automatic patching of systems causing changes that could disable certain software packages. An extension of this thesis could involve studying the risks and rewards of automatically patching system software on an enterprise level.

5.2.3. Systems Management Server

Microsoft Systems Management Server allows administration of an entire network from a centralized point. Deploying updates, new software, and on-site troubleshooting can be costly, in terms of money and labor, especially with a large number of systems. Rolling out a new software program normally involves a privileged user visiting each system and manually installing the software package, or allowing the user of the system to have administrative access over their system. Deploying a system-wide software update, such as a service pack, involves similar labor. Systems Management Server allows administrators to deploy software and updates without visiting each system [Microsoft2].

Systems Management Server maintains an inventory of hardware devices in a SQL database. Knowledge of hardware devices helps to develop plans to upgrade software and hardware for older machines. Hardware inventory aids troubleshooting cases where certain devices with known 'problem' software can be updated to a newer and less problematic software package. Systems Management Server can eliminate human error in inventory upkeep. Using SQL, administrators can query the hardware device database for desired hardware characteristics [Microsoft2].

Systems Management Server maintains an inventory of software programs. It is possible to query a listing of software packages installed on a system, or a listing of all systems with a specific software package installed. When the number of software licenses for an application is limited, Systems Management Server can detect the number of loaned licenses and block users from exceeding the license limit [Microsoft2].

Administrators can control clients remotely using SMS remote desktop abilities to determine, and remedy problems encountered by a user. Systems Management Server allows for full or partial control of a remote client [Microsoft2].

5.2.3.1. Inventory

Systems Management Server gathers data about network resources during 'discovery' events. First, Systems Management Server gathers information about a computer system when the system is added to an SMS 'site.' A 'site' in SMS can refer to any logical partitioning of resources, including an entire enterprise or a single subnet. Second, SMS can gather information from a DHCP server to discover systems that are not present on the network at a given time. Third, different logon methods into Novell NetWare systems can trigger Systems Management Server to gather information. Finally, SMS periodically checks known resources for updates. SMS polls computer systems that do not regularly activate conventional discovery triggers, such as systems that are infrequently rebooted, to keep its information about these resources current [Microsoft2].

After a resource is discovered, the Systems Management Server client is automatically installed on the newly discovered resource. The SMS client software then collects resource data collection and transfers it to the Systems Management Server upon request. The software tracks data on more than 200 different attributes of each resource system. Client attributes tracked are reminiscent of data listed by the Windows Device Manager and Windows System Information. The Windows Device Manager lists hardware devices, such as drives, video and network adapters, disk controllers, ports and other peripherals. Microsoft System Information lists hardware device attributes like processor speed, amount of memory, operating system, and computer manufacturer, as well as hardware resources, such as memory address and interrupt requests.

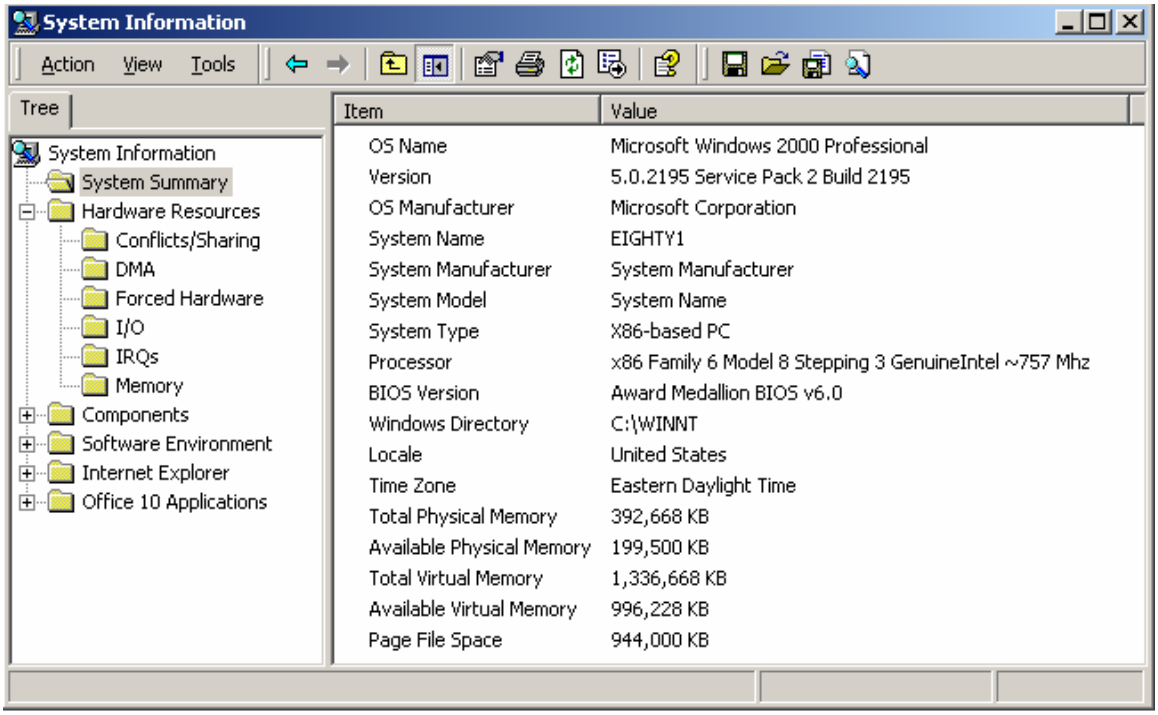

Figure 6. Microsoft System Information MMC snap-in

If a system's hardware configuration is changed, only changes and not the complete hardware inventory are transmitted to the Systems Management Server. Transmitting only changes saves bandwidth on slower connections and reduces overhead on the Systems Management Server. The SMS client installed on each resource system collects the amendment data [Microsoft2].

The SMS client performs a software inventory in a similar manner to a hardware inventory. The client searches the system for installed software and transmits the compiled list to the Systems Management Server [Microsoft2].

In addition to cataloguing software applications installed on each resource system, Systems Management Server can restrict access to certain applications based on user, time and usage. For example, an administrator can block access to Solitaire on all systems during normal business hours. Also, administrators can also deny access to certain applications with limited licenses. A user wishing to use an application with all available licenses loaned enters a line for a 'callback' when a license becomes free for use. Mobile users can also

check out a license for the use of a program on a laptop computer so the mobile user can execute the problem disconnected from the network without exceeding available software licenses [Microsoft2].

5.2.3.2. Deployment

Deployment of a single application or application update over a network of hundreds or thousands of systems is time consuming and very costly. Systems Management Server can ease a 'rollout' of an application or an update over a network by eliminating the 'footwork' involved in a classic rollout. The Systems Management Server first alerts each SMS client of an applicable update. After acceptance by the user, the SMS client then installs the update. Systems Management Server can install single files, applications or operating systems [Microsoft2].

Through the SMS Software Inventory, an administrator can assess systems in need of certain updates. Updates can be advertised to single or multiple users, to a network segment, or to a specific machine. Once a user receives an advertisement from the SMS client, that user can choose to install the software immediately or schedule an unattended install. Conventionally, an administrator would need to visit each computer, log in as a privileged user and install the software manually. The SMS client acts as a system administrator, so any user can install advertised software, regardless of privilege.

To prepare a software update for SMS distribution, an administrator creates a script to automate the installation process on the client system. The administrator must create a 'package' that details the software information and location for distribution on the network. Multiple distribution points can be created to alleviate a bottleneck at one server hosting an update for an entire network. Once the application is hosted on the appropriate distribution points, and the script and package are created, the administrator may advertise the

new application to a single user/machine or multiple users/machines. The SMS client on each appropriate machine then alerts the user that new software is available for installation [Microsoft2].

Programs may be automatically uninstalled from client machines by SMS. When a program is no longer needed or when an unattended installation fails, programs can be uninstalled by the SMS client. Administrators can reduce installation-scripting errors by using the SMS 'dry-run' function. An application package can be test-installed to determine the integrity of the installation script and application files [Microsoft2].

5.2.3.3. Diagnostics and Troubleshooting

Application and hardware problems can be diagnosed remotely using Systems Management Server. An administrator can take full control over the client computer, send files when needed or chat with the user to gather more information about a problem with the client computer.

Remote control allows an administrator to view a client system remotely and issue commands to the client via mouse or keyboard. The remote control functionality is only accessible to individuals with administrative rights [Microsoft2].

Administrators can also reboot a remote machine to finish a software update, or to revive a system that has failed. Administrators can also execute programs remotely. Using all these features of SMS, an administrator can send an update in executable form to a client computer, execute the update and then reboot the machine to finish the installation process [Microsoft2].

In addition to monitoring the health of a single system, SMS permits administrators to monitor their network's health. Potential network trouble includes broken links and bottlenecks. A network can be mapped and graphically displayed for the administrator. The mapping can show systems

within a subnet and how subnets connect to form a compete network. Mapping information can be used to find bottlenecks in traffic patterns or software distribution paths.

References

[AmHeritage] The American Heritage Dictionary of the English Language, Fourth Edition.

[Axelsson1] Axelsson, Stefan. *Intrusion Detection Systems: A Survey and Taxonomy*. 14 March 2000.

[Axelsson2] Axelsson, Stefan. *The Base-Rate Fallacy and the Difficulty of Intrusion Detection*. ACM Transactions on Information and System Security, Vol. 3, No. 3, August 2000, pages 186-205.

[Cui] Cui, Zhiqiang. *Security Incidents in an Academic Setting: A Case Study.* East Tennessee State University: Electronic Theses and Dissertations. <http://etd-submit.etsu.edu/etd/theses/available/etd-0330102-212624/> accessed January 20, 2004

[Deraison1] Deraison, Renaud. *Manpage of NESSUSD*. <http://www.nessus.org/doc/nessusd.html> September 17, 2000

[Deraison2] Deraison, Renaud. *Manpage of NESSUS*. <http://www.nessus.org/doc/nessus.html> September 17, 2000

[Deraison3] Deraison, Renaud. *Plugins*. <http://cgi.nessus.org/plugins/dump.php3> accessed July 15, 2001.

[Farmer1] Farmer, Dan and Eugene H. Spafford. *The COPS Security Checker System.* July 10, 1992.

[Farmer2] Farmer, Dan. *COPS Overview.* <http://www.fish.com/cops/overview.html> May 18, 1993.

[Farmer3] Farmer, Dan. *What SATAN Is.* <http://www.fish.com/satan/summary.html> accessed July 23, 2001.

[Fyodor] Fyodor. *Nmap Network Security Scanner MAN Page*. <http://www.insecure.org/nmap/nmap_manpage.html> accessed July 12, 2001.

[LC4] *LC4 Documentation*. <http://www.atstake.com/research/LC4/documentation/help.htm> accessed July 13, 2001.

[Microsoft1] *IIS Lockdown Tool* <http://www.microsoft.com/windows2000/downloads/recommended/iislockdown/default.asp> accessed April 13, 2003.

[Microsoft2] *Systems Management Server Version 2.0: Scalable Management for Windows Based Systems.* 1998.

[Microsoft3] *Windows NT System Key Permits Strong Encryption of the SAM*. <http://support.microsoft.com/support/kb/articles/Q143/4/75.asp> February 22, 2001.

[MS99-025] *Microsoft Security Bulletin (MS99-025).* <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms99-025.asp> accessed April 13, 2003.

[MS00-029] *Microsoft Security Bulletin (MS00-029).* <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms00-029.asp> accessed April 26, 2002.

[MS00-036] *Microsoft Security Bulletin (MS00-029).* <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms00-029.asp> accessed April 26, 2002.

[MS00-047] *Microsoft Security Bulletin (MS00-047).* <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms00-029.asp> accessed April 29, 2002.

[MS00-052] *Microsoft Security Bulletin (MS00-052).* <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms00-052.asp> accessed April 29, 2002.

[MS00-053] *Microsoft Security Bulletin (MS00-053).* <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms00-053.asp> accessed April 24, 2002.

[MS00-062] *Microsoft Security Bulletin (MS00-062).* <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms00-062.asp> accessed April 29, 2002.

[MS00-065] *Microsoft Security Bulletin (MS00-065).* <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms00-065.asp> accessed April 24, 2002.

[MS00-066] *Microsoft Security Bulletin (MS00-066).* <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms00-066.asp> accessed April 29, 2002.

[MS00-067] *Microsoft Security Bulletin (MS00-067).* <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms00-067.asp> accessed April 25, 2002.

[MS00-070] *Microsoft Security Bulletin (MS00-070).* <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms00-070.asp> accessed April 23, 2002.

[MS00-072] *Microsoft Security Bulletin (MS00-072).* <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms00-072.asp> accessed April 30, 2002.

[MS00-086] *Microsoft Security Bulletin (MS00-086).* <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms00-086.asp> accessed April 23, 2002.

[MS00-089] *Microsoft Security Bulletin (MS00-089).* <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms00-089.asp> accessed April 23, 2002.

[MS00-091] *Microsoft Security Bulletin (MS00-091).* <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms00-091.asp> accessed April 29, 2002.

[MS01-003] *Microsoft Security Bulletin (MS01-003).* <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms01-003.asp > accessed April 29, 2002.

[MS01-008] *Microsoft Security Bulletin (MS01-008).* <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms01-008.asp > accessed April 24, 2002.

[MS01-009] *Microsoft Security Bulletin (MS01-009).* <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms01-009.asp > accessed April 29, 2002.

[MS01-011] *Microsoft Security Bulletin (MS01-011).* <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms01-009.asp > accessed April 30, 2002.

[MS01-025] *Microsoft Security Bulletin (MS01-025).* <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms01-025.asp > accessed April 23, 2002.

[MS01-033] *Microsoft Security Bulletin (MS01-033).* <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms01-033.asp > accessed April 13, 2003.

[MS01-044] *Microsoft Security Bulletin (MS01-044).* <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms01-044.asp > accessed April 25, 2002.

[MS01-046] *Microsoft Security Bulletin (MS01-046).* <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms01-046.asp > accessed April 25, 2002.

[MS01-048] *Microsoft Security Bulletin (MS01-048).* <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms01-048.asp> accessed April 29, 2002.

[MS03-007] *Microsoft Security Bulletin (MS03-007).* <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms03-007.asp> accessed January 29, 2004.

[MS03-026] *Microsoft Security Bulletin (MS03-026).* <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms03-026.asp> accessed January 30, 2004.

[RFC793] *RFC 793 - Transmission Control Protocol: DARPA Internet Program Protocol Specification*. <http://www.faqs.org/rfcs/rfc793.html> September 1981.

[RFC1001] *RFC 1001 – Protocol Standard for a NETBIOS Service on a TCP/UDP Transport: Concepts and Methods.* <http://www.faqs.org/rfcs/rfc1001.html> March 1987.

[Roesch] Roesch, Martin. *Snort - Lightweight Intrusion Detection for Networks.* Proceedings of USENIX LISA 99 conference. November, 1999.

[Securiteam] *IIS Cross-Site Scripting Vulnerability.* <http://www.securiteam.com/windowsntfocus/IIS_Cross-Site_scripting_vulnerability__Patch_available_.html> accessed April 13, 2003

[Webopedia] *SMB*. <*http://www.webopedia.com/TERM/S/SMB.html*> accessed January 30, 2004.

[xforce1] *WU-FTPD glob() function error handling heap corruption.* < http://xforce.iss.net/xforce/xfdb/7611> accessed January 28, 2004.

[xforce2] *McAfee myCIO HTTP server directory traversal.* <http://xforce.iss.net/xforce/xfdb/6834> accessed January 30, 2004.

[xforce3] *IIS 3.0 newdsn.exe sample application allows remote creation of arbitrary files.* <http://xforce.iss.net/xforce/xfdb/1530> accessed January 27, 2004.

Appendix A

Terms and Definitions

some definitions adapted from http://www.webopedia.com

Term or abbreviation	Definition
ACK	ACKnowledge – A type of TCP/IP packet sent to acknowledge the host and client are attempting to create a connection.
bootpc	A utility that allows a client to receive network information from a bootp server.
bootps	A service that allows clients to network boot using a bootp server.
chargen	A UNIX utility that sends random characters over a network.
CGI	Common Gateway Interface – A specification that allows a server-side executable to accept and manipulate data given from a web page.
COPS	Computer Oracle and Password System - A freeware UNIX tool that discovers computer system misconfigurations that pose a risk to system and network security.
CRC	Cyclic Redundancy Check – a technique for determining transmission errors.
DDoS	Distributed Denial of Service (attack) – an attack launched by multiple computers against a host that blocks a large percentage of legitimate network traffic from entering the host network or system.
DHCP	Dynamic Host Configuration Protocol – a protocol used for automatically assigning dynamic IP addresses to network clients.
DoS	Denial of Service – an attack launched by a single computer against a host that causes a failure in the host that denies services.
DOS	Disk Operating System – a computer operating system developed by Microsoft prevalent in many computers for over a decade.
echo	A utility used to display a line of text on a screen.
ETSU	East Tennessee State University - a regional university located in Johnson City, Tennessee.
exec	A UNIX command that executes applications.
FIN	FINish – a type of TCP/IP packet sent to terminate the connection between a host and a client.

finger	A utility used to retrieve information about a remote user.
FTP	File Transfer Protocol – a protocol used for transferring files across the Internet.
gopher	A system of organizing information for remote browsing that predates the World Wide Web.
HTTP	HyperText Transfer Protocol – The protocol used by the World Wide Web for transferring data between web servers and web browsers.
ICMP	Internet Control Message Protocol – A protocol used for transmitting error, control and informational messages.
IDS	Intrusion Detection System – a system, normally placed at the entrance of a network, that examines network traffic and logs packets sent with possible malicious intent.
IP	Internet Protocol – a connectionless protocol used to deliver packets of information between two, or more, systems on a network. Paired with the Transport Control Protocol to create TCP/IP, the main protocol used on the Internet and local area networks.
klogin	A utility used to remotely login to a UNIX machine
kshell	A utility used to remotely open a Kerberos remote shell
LC4	A Windows tool that audits password strength of Windows user accounts.
login	A UNIX command used to sign onto a system.
mask-request	A remote request for a network mask.
Nessus	A freeware UNIX tool that scans systems on a network to discover “well-known” security vulnerabilities.
NIS	Network Information Service/Server – A server that continuously browses a network to discover services and lists network services for lookup.
Nmap	A freeware UNIX tool that determines open ports within a network protocol stack.
NTFS	New Technology File System – A file system used by Windows NT machines that increases reliability and security when compared to FAT.
OSPF	Open Shortest Path First – a routing protocol that offers a more efficient method of router intercommunication than the Routing Information Protocol.
PUSH	A type of TCP/IP packet that prioritizes the packet.
RFC	Request For Comments – Documents submitted to be considered for Internet standards.
RIP	Routing Information Protocol – a protocol used by routers to exchange internet topography information between routers

RPC	Remote Procedure Call – a protocol that allows a client to execute a program on a host
RST	ReSeT – a type of TCP/IP packet that resets the connection between a client and host.
SATAN	Security Administrator Tool for Analyzing Networks – the first well-known, network security scanner.
SMB	Server Message Block – a message format used by DOS and Windows to share files, directories and devices.
SMS	Systems Management Server – An application developed by Microsoft for administering a network from a centralized point.
SMTP	Simple Mail Transfer Protocol – a protocol used for sending e-mail messages between e-mail servers
SNMP	Simple Network Management Protocol – a protocol used for managing networks. Mainly for remote system diagnostics.
Snort	A freeware cross-platform "lightweight intrustion detection system"
SQL	Structured Query Language – A language for database data manipulation.
SYN	SYNchronize – A type of TCP/IP packet that synchronizes a data stream between a client and host.
talk	A UNIX program used to communicate between two
TCP	Transport Control Protocol – a connection-based protocol that enables two system to communicate with streams of information. Paired with the Internet Protocol to create TCP/IP, the main protocol used on the Internet and local area networks.
telnet	A program/protocol used to connect and interact with a remote host to perform operations as if the remote host were local.
TFTP	Trivial File Transfer Protocol – A FTP service without security features that uses the UDP protocol to transfer files between systems.
timestamp-request	A remote request for a timestamp.
traceroute	A utility that tracks a packet between the client and receiving host, noting all intermediate systems traversed.
UDP	User Datagram Protocol – a connectionless protocol used mainly in broadcast delivery of packets.
UNIX	A computer operating system with many variants built for multi-user, multi-process operation.
URG	URGent – a type of TCP/IP packet that flags the packet as containing urgent data.

who	A UNIX program used to list users currently signed into a system.
whois	A utility used to find information about a domain name or an IP address.
xdmcp	X Display Manager Control Protocol – used by X terminals to set up an X session with a remote system.

Appendix B

nsrparser source code

```
// --- main.cpp ---
//
// purpose : this program parses a Nessus .nsr file (or a cat of multiple
//                                 .nsr files) to return desired statistics.  hex edit and remove
//                                 all 1A entries from files before using.
//
// ----------------

// declarations
#pragma warning(disable:4786)
#include <algorithm>                                                    // vector sorting
#include <fstream>                                                      // file streaming
#include <iostream>                                                     // user input and screen output streams
#include <string>                                                       // string container
#include <vector>                                                       // vector container
using namespace std;

// struct declarations
struct SELECTION // a single line of a .nsr file separated into its pieces
{

        string                  strHost;                        // host name/ip
        string                  strPort;                        // port number
        string                  strPlugin;                      // plugin number
        string                  strSeverity;                    // warning type
        string                  strDescription;                 // vulnerability description

};

struct PLUGDESC // plugin number and associated description
{

        string                  strPlugin;                      // plugin number
        string                  strDescription;                 // vulnerability description

};

struct OPERSYS // operating system information gathered from scan
{

        string                  strHost;                        // host name/ip
        string                  strnmap;                        // nmap findings
        string                  strQueso;                       // queso findings
        string                  strOS;                          // host OS (inferred)

};

struct HOST // host names
{

        string                  strHost;                        // host name/ip

};


struct PORTSEV // an open port and its associated occurrences of notes, info and report flags
{

        string                  strPort;                        // port number
```

```
        int                     nNOTE;                  // number of information entries
        int                     nINFO;                  // number of warning entries
        int                     nREPORT;                // number of vulnerability entries

};

//variable declarations
        fstream                 fstrIn;                 // default incoming stream
        fstream                 fstrOut;                // default outgoing stream
        SELECTION*              SELCurrent;             // pointer for NEW SELECTION creation
        vector<HOST>*           pvecHosts;              // pointer for NEW vector<HOST> creation
        vector<OPERSYS>         vecOS;                  // operating system information vector
        vector<PORTSEV>*        pvecSEV;                // pointer for NEW vector<PORTSEV> creation
        vector<PLUGDESC>        vecPLUGDESC;            // plugins and associated descriptions vector
        vector<PLUGDESC>*       pvecPLG;                // pointer for NEW vector<PLUGDESC>
                                                                creation
        vector<SELECTION>       vecMainData;            // main data vector of SELECTIONS

//function declarations
        void                    findDescription(string);        // finds a part of a description
        void                    findDescription2(string, string);
                                                        // finds a part of a description
        void                    findHoles(string);
                                                        // finds holes on a specified port
        void                    findPlugin(string, string);     // finds hosts with specific information
        void                    makeSelection(string&);         // make selection struct from NSR line
        string                  parseString(string&);           // retrieves information from NSR line
        void                    retrieveData();                 // opens and parses file, filling vecMainData
        void                    openPorts(string);              // counts number of host with a port open
        void                    operatingSystems();             // find operating systems for hosts
        void                    outputPlugin();                 // outputs list of plugins found to file
        void                    portInfo();                     // returns port information
        void                    portVulnerabilities(string, string);    // finds all vulnerabilities listed for a port
        string                  removeSemicolons(string);       // removes semicolons from plugin description
        string                  returnOS(string);
        void                    mainMenu();                     // offers user parsing options
        void                    uniqueHosts();                  // counts unique hosts
        bool                    vecsort(PORTSEV, PORTSEV);

// --- main() ---
//
// --------------
void main()
{

        cout << ".nsr parser v0.50" << endl;

// load information
        retrieveData();

// manipulate information
        mainMenu();

        cout << "[quit]" << endl;

} // end main

// --- retrieveData() ---
//
// called by : main
// calls to : makeSelection, operatingSystems
// purpose : opens a file in .nsr format and parses the file, filling vecMainData,
//                              a vector of SELECTION objects.
//
// -------------------
void retrieveData()
```

```
{
        bool                                    bBadFile = true;        // bad filename flag
        bool                                    bFoundValue = false;    // item found in vector flag
        char                                    cDelimiter = 10;        // delimiter for .nsr SELECTION information
        PLUGDESC*                               PTemp;                  // pointer for NEW PLUGDESC creation
        string                                  strFileName = "",       // name of file to parse
                                                strLine = "",           // a single line of the data file
                                                strTemp = "";           // string container for temporary information
        vector<PLUGDESC>::reverse_iterator      iPLG;                   // vecPLUGDESC iterator
        vector<SELECTION>::iterator             iSEL;                   // vecMainData iterator

        do
        {
// get filename to be used in processing
                cout << "[enter name of data file]: ";
                cin >> strFileName;
                cin.ignore(80, '\n');

// attempt to open the file, ask again on error, continue until acceptable
                fstrIn.open(strFileName.c_str());
                bBadFile = fstrIn.fail();
                if (bBadFile)
                {
                        cout << "!! error - can not find file " << strFileName << endl;
                        fstrIn.clear();
                }
        } while (bBadFile);

// get a line of text, and process each line using the hex 0A delimiter
        cout << "[processing main data vector] ";

        do
        {
                getline(fstrIn, strLine, cDelimiter);
                makeSelection(strLine);
        } while (!fstrIn.eof());

        cout << " [done @ " << vecMainData.size() << " Nessus entries]" << endl;

// close the file
        fstrIn.close();

// create vector of plugin numbers and associated descriptions
        cout << "[processing plugin/description vector] ";

        for (iSEL = vecMainData.begin(); iSEL != vecMainData.end(); iSEL++)
        {
                strTemp = (*iSEL).strPlugin;

                for (iPLG = vecPLUGDESC.rbegin();
                        iPLG != vecPLUGDESC.rend() && bFoundValue == false;
                        iPLG++)
                {
                        if (strTemp == (*iPLG).strPlugin)
                        {
                                bFoundValue = true;
                        }
                }

// if value is not found in host vector, add hostname to host vector
                if (bFoundValue == false)
                {
                        PTemp = new PLUGDESC;
                        (*PTemp).strPlugin = (*iSEL).strPlugin;
                        (*PTemp).strDescription = (*iSEL).strDescription;
```

```
                        vecPLUGDESC.push_back(*PTemp);
                        delete PTemp;
                }

// reset bFoundValue
                bFoundValue = false;
        }

        cout << " [done @ " << vecPLUGDESC.size() << " Nessus plugins]" << endl;

// fill operating systems vector
        cout << "[processing operating system information] ";

        operatingSystems();

        cout << " [done @ " << vecOS.size() << " OS inferences]" << endl;

} // retrieveData()

// --- makeSelection() ---
//
// called by: retrieveData
// calls to: parseString
// inputs: strLine - a single line from the data file
// purpose: fill a SELECTION object from a line of the data file
//
// ----------------------
void makeSelection(string& strLine)
{

// create a NEW SELECTION
        SELCurrent = new SELECTION;

// get host name from first section of data file line
        (*SELCurrent).strHost = parseString(strLine);

// check for bogus host name/ip entries, else fill vector
        if ((*SELCurrent).strHost == "<empty>")
        {
                delete SELCurrent;
        }
        else
        {
                (*SELCurrent).strPort = parseString(strLine);
                (*SELCurrent).strPlugin = parseString(strLine);
                (*SELCurrent).strSeverity = parseString(strLine);
                (*SELCurrent).strDescription = removeSemicolons(parseString(strLine));

// add the nSelection to vecMainData and destroy SELCurrent
                vecMainData.push_back(*SELCurrent);
                delete SELCurrent;
        }

} //makeStruct()

// --- parseString() ---
//
// called by: makeSelection
// inputs: strLine - a single line from the data file
// purpose: parses a single line of a .nsr file using the | delimiter
// returns: portion of strLine or <empty> is line is empty
//
// --------------------
string parseString(string& strLine)
{
```

```
	int			nCol = 0;		// column of desired character
	string			strReturn = "";		// string to return

// return <empty> if the length of strLine = 0
	if (strLine.length() == 0)
		return ("<empty>");

// find first | in the line, and return the portion of the string before the delimiter
// if no delimiter, return the entire string.  trim strLine to remove used portion
	nCol = strLine.find_first_of("|");

	if (nCol != string::npos)
	{
		strReturn = strLine.substr(0,nCol);
		strLine = strLine.substr(nCol+1, strLine.length() - nCol);
	}
	else
	{
		strReturn = strLine;
		strLine = "";
	}

	return strReturn;

} // parseString()

// --- removeSemicolons() ---
//
// called by: makeSelection
// inputs: strLine - description returned from plugin
// purpose: replaces ; from plugin descriptions with newline characters for easier output
// returns: strTemp - the plugin description with replaced ;
//
// -------------------------
string removeSemicolons(string strLine)
{

	int			nCol = 0;		// column of desired character
	string			strTemp = "";		// string container for temporary information

// if no ; exist, return
	if (strLine.find_first_of(";") == string::npos)
	{
		return (strLine);
	}

// loop until no ; remain
	do
	{
		nCol = strLine.find_first_of(";");

//change the next line to choose the ; replacement
		strTemp = strTemp + strLine.substr(0, nCol) + "\n";
		strLine = strLine.substr(nCol+1, strLine.length() - nCol);
	} while (strLine.find_first_of(";") != string::npos);

	strTemp = strTemp + strLine;

	return (strTemp);

} // removeSemicolons()

// --- operatingSystems() ---
//
// called by: retrieveData
// purpose: infer operating system of host from information gathered during the scan
```

```
//
// -------------------------
void operatingSystems()
{

        bool                                    bFoundValue = false;        // host found in vector flag

// operating system clues found in nmap and queso plugins
        const char*                             aix = "AIX";
        const char*                             cisco = "Cisco";
        const char*                             early = "NT4 / Win95 / Win98";
        const char*                             early2 = "95/98/NT";
        const char*                             hpux = "HP-UX";
        const char*                             irix = "Irix";
        const char*                             irix2 = "IRIX";
        const char*                             linux = "Linux";
        const char*                             ljet1 = "LaserJet";
        const char*                             ljet2 = "JETdirect";
        const char*                             ljet3 = "JetDirect";
        const char*                             ltx         = "Lantronix";
        const char*                             mac         = "Mac";
        const char*                             me = "Me";
        const char*                             me2 = "Millenium";
        const char*                             nov = "Novell";
        const char*                             nts = "NT 4.0 Server";
        const char*                             sol = "Solaris";
        const char*                             three1 = "3Com";
        const char*                             three2 = "3COM";
        const char*                             un = "UNIX";
        const char*                             vms = "VMS";
        const char*                             wnt = "WindowsNT";
        const char*                             w2k = "Windows 2000";
        const char*                             w2ks = "Advance Server";
        const char*                             w31 = "3.11";
        const char*                             xp = "XP";

        OPERSYS*                                OTemp;                      // pointer for NEW OPERSYS objects
        string                                  strnmap = "",
                                                strQueso = "",              // OS information findings
                                                strTemp = "";               // string container for temporary information
        string                                  NMAP_PLUGIN_NUMBER = "10336";
                                                                            // plugin number for nmap OS information
        string                                  QUESO_PLUGIN_NUMBER = "10337";
                                                                            // plugin number for queso OS information
        vector<OPERSYS>::reverse_iterator iOS;                              // vecOS iterator
        vector<SELECTION>::iterator             iSEL;                       // vecMainData iterator

// crawl main data vector for hostnames
        for(iSEL = vecMainData.begin(); iSEL != vecMainData.end(); iSEL++)
        {

// get plugin number
                strTemp = (*iSEL).strPlugin;

// compare plugin numbers
                if ((strTemp == NMAP_PLUGIN_NUMBER) || (strTemp == QUESO_PLUGIN_NUMBER))
                {
// check host vector for hostname found in vecMainData
// reverse iterator chosen for "rule of proximity"
                        for(iOS = vecOS.rbegin();
                                iOS != vecOS.rend() && bFoundValue == false;
                                iOS++)
                        {
// if hostname is already listed in vecOS, add new data to vecOS
                                if ((*iSEL).strHost == (*iOS).strHost)
                                {
```

```
                                        bFoundValue = true;

                                        if ((*iSEL).strPlugin == NMAP_PLUGIN_NUMBER)
                                                (*iOS).strnmap = (*iSEL).strDescription;
                                        else
                                                (*iOS).strQueso = (*iSEL).strDescription;
                                }
                        }

// if value is not found in host vector, add hostname to host vector
                        if (bFoundValue == false)
                        {
                                OTemp = new OPERSYS;
                                (*OTemp).strHost = (*iSEL).strHost;
                                (*OTemp).strnmap = "<empty>";
                                (*OTemp).strQueso = "<empty>";
                                (*OTemp).strOS = "<no information>";

                                if ((*iSEL).strPlugin == NMAP_PLUGIN_NUMBER)
                                        (*OTemp).strnmap = (*iSEL).strDescription;
                                else
                                        (*OTemp).strQueso = (*iSEL).strDescription;

                                (vecOS).push_back(*OTemp);
                                delete OTemp;
                        }

// reset bFoundValue
                        bFoundValue = false;
                }
        }

// attempt to determine OS from nmap and queso findings and update vecOS
        for (iOS = vecOS.rbegin(); iOS != vecOS.rend(); iOS++)
        {

                strnmap = (*iOS).strnmap;
                strQueso = (*iOS).strQueso;

                if ((strnmap.find(sol) != string::npos) && (strQueso.find(sol) != string::npos))
                        (*iOS).strOS = "Solaris";
                else if ((strnmap.find(w2ks) != string::npos) && (strQueso.find(wnt) != string::npos))
                        (*iOS).strOS = "Windows 2000 Server";
                else if ((strnmap.find(w2k) != string::npos) && (strQueso.find(wnt) != string::npos))
                        (*iOS).strOS = "Windows 2000 Professional";
                else if ((strnmap.find(nts) != string::npos) && (strQueso.find(wnt) != string::npos))
                        (*iOS).strOS = "Windows NT4 Server";
                else if ((strnmap.find(early) != string::npos) && (strQueso.find(wnt) != string::npos))
                        (*iOS).strOS = "Windows NT4 Workstation";
                else if ((strnmap.find(early) != string::npos) && (strQueso.find(early2) != string::npos))
                        (*iOS).strOS = "Windows 95 or 98";
                else if ((strnmap.find(cisco) != string::npos) && (strQueso.find(cisco) != string::npos))
                        (*iOS).strOS = "Cisco IOS";
                else if (strnmap.find(me2) != string::npos)
                        (*iOS).strOS = "Windows Millennium Edition";
                else if (strnmap.find(w31) != string::npos)
                        (*iOS).strOS = "Windows for Workgroups v3.11";
                else if ((strnmap.find(ljet1) != string::npos) || (strQueso.find(ljet2) != string::npos) || (strnmap.find(ljet3)
!= string::npos))
                        (*iOS).strOS = "HP JetDirect Printer";
                else if ((strnmap.find(xp) != string::npos) || (strQueso.find(xp) != string::npos))
                        (*iOS).strOS = "Windows XP";
                else if ((strnmap.find(nov) != string::npos) || (strQueso.find(nov) != string::npos))
                        (*iOS).strOS = "Novell Netware";
                else if ((strnmap.find(three1) != string::npos) || (strQueso.find(three2) != string::npos))
                        (*iOS).strOS = "3Com Device";
```

```
                else if ((strnmap.find(hpux) != string::npos) || (strQueso.find(hpux) != string::npos))
                        (*iOS).strOS = "HP-UX";
                else if ((strnmap.find(linux) != string::npos) || (strQueso.find(linux) != string::npos))
                        (*iOS).strOS = "Linux";
                else if ((strnmap.find(irix) != string::npos) || (strQueso.find(irix2) != string::npos))
                        (*iOS).strOS = "Irix";
                else if ((strnmap.find(aix) != string::npos) || (strQueso.find(aix) != string::npos))
                        (*iOS).strOS = "AIX";
                else if ((strnmap.find(vms) != string::npos) || (strQueso.find(vms) != string::npos))
                        (*iOS).strOS = "VMS";
                else if ((strnmap.find(mac) != string::npos) ||    (strQueso.find(mac) != string::npos))
                        (*iOS).strOS = "MacOS";
                else if ((strnmap.find(un) != string::npos) || (strQueso.find(un) != string::npos))
                        (*iOS).strOS = "[?]UNIX";
                else if ((strnmap.find(w2k) != string::npos) && (strnmap.find(me) != string::npos))
                        (*iOS).strOS = "[?]Windows 2000 Professional";
                else if (strnmap.find(w2ks) != string::npos)
                        (*iOS).strOS = "[?]Windows 2000 Server";
                else if (strnmap.find(nts) != string::npos)
                        (*iOS).strOS = "[?]Windows NT4 Server";
                else if (strQueso.find(wnt) != string::npos)
                        (*iOS).strOS = "[?]Windows NT4 Workstation / Server / Cisco IOS";
                else if ((strnmap.find(early) != string::npos) || (strQueso.find(early2) != string::npos) ||
(strnmap.find(early2) != string::npos))
                        (*iOS).strOS = "[?]Windows 95 / 98 / NT4";
                else
                {
                        (*iOS).strOS = "<unknown>";
                }
        }
} // operatingSystems()

// --- mainMenu() ---
//
// called by: main
// purpose: main menu for program.  switch on users choice.
//
// ------------------
void mainMenu()
{

        bool                            bGoodValue = false;             // acceptable menu selection flag
        char                            YN = 'n';                       // yes/no answer
        int                             nSelection;                     // menu choice of user
        string                          strInfo;                        // user given information
        string                          strInfo2;                       // user given information
        vector<PLUGDESC>::iterator iPLG;                                // vecPLUGDESC iterator

// loop until quit
        do
        {
//loop until acceptable choice
                do
                {
// reset yes/no flag
                        YN = 'n';

                        cout << endl << "[main menu]" << endl << endl;
                        cout << "    [ 1] hosts by name" << endl;
                        cout << "    [ 2] hosts by single port" << endl;
                        cout << "    [ 3] hosts by single vulnerability" << endl;
                        cout << "    [ 4] hosts by single port vulnerabilities" << endl;
                        cout << "    [ 5] ports by # vulnerabilities" << endl;
                        cout << "    [ 6] vulnerabilities by single port" << endl;
                        cout << "    [ 7] hosts by description" << endl;
                        cout << "    [ 8] hosts by two descriptions" << endl;
```

```
		cout << "     [ 9] warnings by single port" << endl;
		cout << "     [10] notes by single port" << endl;
		cout << "     [11] hosts by single warning" << endl;
		cout << "     [12] hosts by single note" << endl;
		cout << "     [97] output plugin information to file" << endl;
		cout << "     [98] change information file" << endl;
		cout << "     [99] quit" << endl;
		cout << endl << "     [1-10, 98, 99]: ";
		cin >> nSelection;
		cin.ignore(80, '\n');

		if (((nSelection >= 1) && (nSelection <= 12)) || ((nSelection >= 97) && (nSelection <= 99)))
			bGoodValue = true;

		if (bGoodValue == false)
			cout << endl << "!! error - invalid menu choice" << endl << endl;

} while (bGoodValue == false);

switch(nSelection)
{

		case 1:
			uniqueHosts();
			break;

		case 2:
			cout << endl << "[enter port number to search for] : ";
			cin >> strInfo;
			cin.ignore(80, '\n');
			cout << endl;

			openPorts(strInfo);
			break;

		case 3:
			cout << endl << "[enter plugin number to search for] : ";
			cin >> strInfo;
			cin.ignore(80, '\n');
			cout << endl;

			for (iPLG = vecPLUGDESC.begin(); iPLG != vecPLUGDESC.end(); iPLG++)
			{

				if ((*iPLG).strPlugin == strInfo)
				{
					cout << endl << (*iPLG).strDescription << endl;
					cout << endl << "[find this information/vulnerability (y/n)] : ";
					cin >> YN;
					cout << endl;
					cin.ignore(80, '\n');

					if (YN == 'y')
					{
						findPlugin(strInfo, "REPORT");
						break;
					}
					else
					{
						cout << endl <<"!! error - user intervention" << endl;
						break;
					}

				}

				if (YN == 'y')
```

```
                        break;

        }

        if (YN == 'y')
                break;

        cout << "!! error - plugin not found" << endl << endl;
        break;

case 4:
        cout << endl << "[enter port number to search for] : ";
        cin >> strInfo;
        cin.ignore(80, '\n');
        cout << endl;

        findHoles(strInfo);
        break;

case 5:
        portInfo();
        break;

case 6:
        cout << endl << "[enter port number to search for] : ";
        cin >> strInfo;
        cin.ignore(80, '\n');
        cout << endl;

        portVulnerabilities(strInfo, "REPORT");
        break;

case 7:
        cout << endl << "[enter description string] : ";
        cin >> strInfo;
        cout << endl;

        findDescription(strInfo);
        cin.ignore(80, '\n');
        break;

case 8:
        cout << endl << "[enter description string 1] : ";
        cin >> strInfo;
        cout << endl;


        cout << endl << "[enter description string 2] : ";
        cin >> strInfo2;
        cout << endl;

        findDescription2(strInfo, strInfo2);
        cin.ignore(80, '\n');
        break;

case 9:
        cout << endl << "[enter port number to search for] : ";
        cin >> strInfo;
        cin.ignore(80, '\n');
        cout << endl;

        portVulnerabilities(strInfo, "INFO");
        break;


case 10:
```

```
		cout << endl << "[enter port number to search for] : ";
		cin >> strInfo;
		cin.ignore(80, '\n');
		cout << endl;

		portVulnerabilities(strInfo, "NOTE");
		break;

case 11:
		cout << endl << "[enter plugin number to search for] : ";
		cin >> strInfo;
		cin.ignore(80, '\n');
		cout << endl;

		for (iPLG = vecPLUGDESC.begin(); iPLG != vecPLUGDESC.end(); iPLG++)
		{

			if ((*iPLG).strPlugin == strInfo)
			{
				cout << endl << (*iPLG).strDescription << endl;
				cout << endl << "[find this information/vulnerability (y/n)] : ";
				cin >> YN;
				cout << endl;
				cin.ignore(80, '\n');

				if (YN == 'y')
				{
					findPlugin(strInfo, "INFO");
					break;
				}
				else
				{
					cout << endl <<"!! error - user intervention" << endl;
					break;
				}

			}

			if (YN == 'y')
				break;

		}

		if (YN == 'y')
			break;

		cout << "!! error - plugin not found" << endl << endl;
		break;


case 12:
		cout << endl << "[enter plugin number to search for] : ";
		cin >> strInfo;
		cin.ignore(80, '\n');
		cout << endl;

		for (iPLG = vecPLUGDESC.begin(); iPLG != vecPLUGDESC.end(); iPLG++)
		{

			if ((*iPLG).strPlugin == strInfo)
			{
				cout << endl << (*iPLG).strDescription << endl;
				cout << endl << "[find this information/vulnerability (y/n)] : ";
				cin >> YN;
				cout << endl;
				cin.ignore(80, '\n');
```

```
                                        if (YN == 'y')
                                        {
                                                findPlugin(strInfo, "NOTE");
                                                break;
                                        }
                                        else
                                        {
                                                cout << endl <<"!! error - user intervention" << endl;
                                                break;
                                        }

                                }

                                if (YN == 'y')
                                        break;

                        }

                        if (YN == 'y')
                                break;

                        cout << "!! error - plugin not found" << endl << endl;
                        break;


                case 97:
                        outputPlugin();
                        break;

                case 98:
                        cout << endl;
                        fstrIn.clear();
                        vecMainData.clear();
                        retrieveData();
                        break;

                case 99:
                        break;

                default:
                        break;

        } // switch

        bGoodValue = false;

    } while (nSelection != 99);

} // mainMenu()

// -- uniqueHosts() --
//
// ------------------

void uniqueHosts()
{
    bool                                bFoundValue = false;    // value found flag
    HOST*                               HTemp;                  // temporary host
    string                              strTemp;                // temporary string container

    pvecHosts = new vector<HOST>;

    vector<SELECTION>::iterator iSEL;
    vector<HOST>::reverse_iterator iHOST;
```

```
// crawl main data vector for hostnames
	for(iSEL = vecMainData.begin(); iSEL != vecMainData.end(); iSEL++)
	{
		strTemp = (*iSEL).strHost;

// check host vector for hostname found in vecMainData
// reverse iterator chosen for "rule of proximity"
		for(iHOST = (*pvecHosts).rbegin();
			iHOST != (*pvecHosts).rend() && bFoundValue == false;
			iHOST++)
		{

			if (strTemp == (*iHOST).strHost)
			{
				bFoundValue = true;
			}

		}

// if value is not found in host vector, add hostname to host vector
		if (bFoundValue == false)
		{

			HTemp = new HOST;
			(*HTemp).strHost = (*iSEL).strHost;
			(*pvecHosts).push_back(*HTemp);
			delete HTemp;

		}

// reset bFoundValue
		bFoundValue = false;

	}

// output
	cout << endl << "<list>" << endl;

	for(iHOST = (*pvecHosts).rbegin(); iHOST != (*pvecHosts).rend(); iHOST++)
	{
		cout << (*iHOST).strHost;
		cout << ": (";
		cout << returnOS((*iHOST).strHost);
		cout << ")" << endl;

	}

	cout << "</list>" << endl;

	cout << endl << "[done @ " << (*pvecHosts).size() << " hosts]" << endl << endl;

	delete pvecHosts;

} // uniqueHosts

// --- openPorts() ---
//
// -------------------

void openPorts(string strPortNumber)
{

	bool		bFoundValue = false;	// value found flag
	HOST*		HTemp;			// temporary host
	int		nCol1, nCol2;		// delimiter columns
```

```
	string				strTemp;			// temporary string container

	pvecHosts = new vector<HOST>;

	vector<SELECTION>::iterator iSEL;
	vector<HOST>::reverse_iterator iHOST;

// crawl main data vector for hostnames
	for(iSEL = vecMainData.begin(); iSEL != vecMainData.end(); iSEL++)
	{

// get port number
		strTemp = (*iSEL).strPort;
		nCol1 = strTemp.find_first_of("(");
		nCol2 = strTemp.find_first_of("/");
		strTemp = strTemp.substr(nCol1 + 1, nCol2 - nCol1 - 1);

// compare port numbers
		if (strTemp == strPortNumber)
		{

// check host vector for hostname found in vecMainData
// reverse iterator chosen for "rule of proximity"
			for(iHOST = (*pvecHosts).rbegin();
				iHOST != (*pvecHosts).rend() && bFoundValue == false;
				iHOST++)
			{

				if ((*iSEL).strHost == (*iHOST).strHost)
				{
					bFoundValue = true;
				}

			}

// if value is not found in host vector, add hostname to host vector
			if (bFoundValue == false)
			{

				HTemp = new HOST;
				(*HTemp).strHost = (*iSEL).strHost;
				(*pvecHosts).push_back(*HTemp);
				delete HTemp;

			}

// reset bFoundValue
			bFoundValue = false;

		}

	}

// output
	cout << endl << "<list>" << endl;

	for(iHOST = (*pvecHosts).rbegin(); iHOST != (*pvecHosts).rend(); iHOST++)
	{
		cout << (*iHOST).strHost;
		cout << ": (";
		cout << returnOS((*iHOST).strHost);
		cout << ")" << endl;
	}

	cout << "</list>" << endl;
```

```
	cout << endl << "[done @ " << (*pvecHosts).size() << " systems with port "
		<< strPortNumber << " open]" << endl << endl;

	delete pvecHosts;

} // openPorts()

// --- findPlugin() ---
//
// --------------------

void findPlugin(string strPluginNumber, string strType)
{

	bool		bFoundValue = false;	// value found flag
	HOST*		HTemp;			// temporary host
	string		strTemp;		// temporary string container

	pvecHosts = new vector<HOST>;

	vector<SELECTION>::iterator iSEL;
	vector<HOST>::reverse_iterator iHOST;

// crawl main data vector for hostnames
	for(iSEL = vecMainData.begin(); iSEL != vecMainData.end(); iSEL++)
	{

// get port number
		strTemp = (*iSEL).strPlugin;

// compare port numbers
		if ((strTemp == strPluginNumber) && ((*iSEL).strSeverity == strType))
		{

// check host vector for hostname found in vecMainData
// reverse iterator chosen for "rule of proximity"
			for(iHOST = (*pvecHosts).rbegin();
				iHOST != (*pvecHosts).rend() && bFoundValue == false;
				iHOST++)
			{

				if ((*iSEL).strHost == (*iHOST).strHost)
				{
					bFoundValue = true;
				}

			}

// if value is not found in host vector, add hostname to host vector
			if (bFoundValue == false)
			{

				HTemp = new HOST;
				(*HTemp).strHost = (*iSEL).strHost;
				(*pvecHosts).push_back(*HTemp);
				delete HTemp;

			}

// reset bFoundValue
			bFoundValue = false;

		}

	}
```

```
// output
	cout << endl << "<list>" << endl;

	for(iHOST = (*pvecHosts).rbegin(); iHOST != (*pvecHosts).rend(); iHOST++)
	{
		cout << (*iHOST).strHost;
		cout << ": (";
		cout << returnOS((*iHOST).strHost);
		cout << ")" << endl;

	}

	cout << "</list>" << endl;

	cout << endl << "[done @ " << (*pvecHosts).size() << " systems with specified "
		<< "information plugin " << strPluginNumber << "]" << endl << endl;

	delete pvecHosts;

} // findPlugin

// --- findHoles() ---
//
// -------------------

void findHoles(string strPortNumber)
{
	bool		bFoundValue = false;	// value found flag
	HOST*		HTemp;			// temporary host
	int		nCol1, nCol2;		// delimiter columns
	string		strTemp;		// temporary string container

	pvecHosts = new vector<HOST>;

	vector<SELECTION>::iterator iSEL;
	vector<HOST>::reverse_iterator iHOST;

// crawl main data vector for hostnames
	for(iSEL = vecMainData.begin(); iSEL != vecMainData.end(); iSEL++)
	{

// get port number
		strTemp = (*iSEL).strPort;
		nCol1 = strTemp.find_first_of("(");
		nCol2 = strTemp.find_first_of("/");
		strTemp = strTemp.substr(nCol1 + 1, nCol2 - nCol1 - 1);

// compare port numbers
		if (strTemp == strPortNumber)
		{

			if ((*iSEL).strSeverity == "REPORT")
			{

// check host vector for hostname found in vecMainData
// reverse iterator chosen for "rule of proximity"
				for(iHOST = (*pvecHosts).rbegin();
				   iHOST != (*pvecHosts).rend() && bFoundValue == false;
					iHOST++)
				{

					if ((*iSEL).strHost == (*iHOST).strHost)
					{
						bFoundValue = true;
					}
```

```
                                        }

// if value is not found in host vector, add hostname to host vector
                                        if (bFoundValue == false)
                                        {

                                                HTemp = new HOST;
                                                (*HTemp).strHost = (*iSEL).strHost;
                                                (*pvecHosts).push_back(*HTemp);
                                                delete HTemp;

                                        }

// reset bFoundValue
                                        bFoundValue = false;

                                }

                        }

                }

// output
        cout << endl << "<list>" << endl;

        for(iHOST = (*pvecHosts).rbegin(); iHOST != (*pvecHosts).rend(); iHOST++)
        {
                cout << (*iHOST).strHost;
                cout << ": (";
                cout << returnOS((*iHOST).strHost);
                cout << ")" << endl;
        }

        cout << "</list>" << endl;

        cout << endl << "[done @ " << (*pvecHosts).size() << " systems with vulnerabilities "
                << "on port " << strPortNumber << "]" << endl << endl;

        delete pvecHosts;

} // findHoles



// --- portInfo() ---
//
// -----------------

void portInfo()
{

        bool                    bFoundValue = false;    // value found flag
        PORTSEV*                PTemp;                  // temporary host
        int                     nCol1, nCol2;           // delimiter columns
        int                     INF = 0,
                                NOT = 0,
                                REP = 0;                // counters
        string                  strTemp;                // temporary string container

        pvecSEV = new vector<PORTSEV>;

        vector<SELECTION>::iterator iSEL;
        vector<PORTSEV>::reverse_iterator iSEV;

// crawl main data vector for hostnames
        for(iSEL = vecMainData.begin(); iSEL != vecMainData.end(); iSEL++)
```

```
	{

// get port number
			strTemp = (*iSEL).strPort;
			nCol1 = strTemp.find_first_of("(");
			nCol2 = strTemp.find_first_of("/");
			strTemp = strTemp.substr(nCol1 + 1, nCol2 - nCol1 - 1);

// check severity vector for port found in vecMainData
// reverse iterator chosen for "rule of proximity"
			for(iSEV = (*pvecSEV).rbegin();
				iSEV != (*pvecSEV).rend() && bFoundValue == false;
				iSEV++)
			{

				if ((*iSEL).strPort == (*iSEV).strPort)
				{

					if ((*iSEL).strSeverity == "NOTE")
						(*iSEV).nNOTE++;
					if ((*iSEL).strSeverity == "INFO")
						(*iSEV).nINFO++;
					if ((*iSEL).strSeverity == "REPORT")
						(*iSEV).nREPORT++;

					bFoundValue = true;

				}

			}

// if value is not found in severity vector, add port number to severity vector
			if (bFoundValue == false)
			{

				PTemp = new PORTSEV;
				(*PTemp).strPort = (*iSEL).strPort;
				(*PTemp).nINFO = 0;
				(*PTemp).nNOTE = 0;
				(*PTemp).nREPORT = 0;

				if ((*iSEL).strSeverity == "NOTE")
					(*PTemp).nNOTE++;
				if ((*iSEL).strSeverity == "INFO")
					(*PTemp).nINFO++;
				if ((*iSEL).strSeverity == "REPORT")
					(*PTemp).nREPORT++;

				(*pvecSEV).push_back(*PTemp);
				delete PTemp;

			}
// reset bFoundValue
			bFoundValue = false;

	}

// output
	char		c;
	char		lclFileName[100];		// file name to output to

	do
	{
		cout << endl << "[print to *S*creen or *F*ile] : ";
		cin >> c;
		cin.ignore(80, '\n');
```

```
		cout << endl;
	} while (c != 'S' && c != 's' && c != 'F' && c != 'f');

	sort ((*pvecSEV).rbegin(), (*pvecSEV).rend(), vecsort);

	if (c == 'S' || c == 's')
	{

		cout << endl << "<list>" << endl;

		for(iSEV = (*pvecSEV).rbegin(); iSEV != (*pvecSEV).rend(); iSEV++)
		{
			cout << (*iSEV).strPort << " Vulnerabilities: " << (*iSEV).nREPORT
				<< " | Warnings: " << (*iSEV).nINFO << " | Notes: "
				<< (*iSEV).nNOTE << endl;
			NOT = NOT + (*iSEV).nNOTE;
			INF = INF + (*iSEV).nINFO;
			REP = REP + (*iSEV).nREPORT;

		};

		cout << "</list>" << endl;

		cout << endl << "[done @ " << (*pvecSEV).size() << " unique ports open]" << endl
			<< "[ " << REP << " vulnerabilities, " << INF << " warnings, "
			<< NOT << " notes ]" << endl;

	}
	else
	{

		cout << "[enter filename for report] : ";
		cin >> lclFileName;
		cin.ignore(80, '\n');
		cout << endl;

		fstrOut.open(lclFileName, ios::out);

		fstrOut << "<list>\n";

		for(iSEV = (*pvecSEV).rbegin(); iSEV != (*pvecSEV).rend(); iSEV++)
		{
			fstrOut << (*iSEV).strPort << " Vulnerabilities: " << (*iSEV).nREPORT
				<< " | Warnings: " << (*iSEV).nINFO << " | Notes: "
				<< (*iSEV).nNOTE << "\n";
			NOT = NOT + (*iSEV).nNOTE;
			INF = INF + (*iSEV).nINFO;
			REP = REP + (*iSEV).nREPORT;

		};

		fstrOut << "</list>\n\n";

		fstrOut << "[done @ " << (*pvecSEV).size() << " unique ports open]\n"
			<< "[ " << REP << " vulnerabilities, " << INF << " warnings, "
			<< NOT << " notes ]\n";

		fstrOut.close();

	}

	delete pvecSEV;

} // portInfo

bool vecsort(PORTSEV p1, PORTSEV p2)
```

```
{

	if (p1.nREPORT != p2.nREPORT)
		return p1.nREPORT > p2.nREPORT;
	else if (p1.nINFO != p2.nREPORT)
		return p1.nINFO > p2.nINFO;
	else
		return p1.nNOTE > p2.nNOTE;

}

// --- outputPlugin() ---
//
// ----------------------

void outputPlugin()
{

	char 					lclFileName[100];		// file name to output to
	vector<PLUGDESC>::iterator iPLG;				// iterator for PLUGDESC vector

	cout << endl << "[enter filename for report] : ";
	cin >> lclFileName;
	cin.ignore(80, '\n');
	cout << endl;

	fstrOut.open(lclFileName, ios::out);

	for (iPLG = vecPLUGDESC.begin(); iPLG != vecPLUGDESC.end(); iPLG++)
	{

		fstrOut << "\n>>>>>[" << (*iPLG).strPlugin << "]" << endl;
		fstrOut << (*iPLG).strDescription << "]" << endl;

	}

	fstrOut.close();

} // outputPlugin()

// --- portVulnerabilities ---
//
// ---------------------------

void portVulnerabilities(string strPortNumber, string strType)
{

	bool 					bFoundValue = false;	// value found flag
	PLUGDESC* 				PTemp;			// temporary host
	int 					nCol1, nCol2;		// delimiter columns
	string 					strTemp, strTemp2;	// temporary string container

	pvecPLG = new vector<PLUGDESC>;

	vector<SELECTION>::iterator iSEL;
	vector<PLUGDESC>::reverse_iterator iPLG;

// crawl main data vector for hostnames
	for(iSEL = vecMainData.begin(); iSEL != vecMainData.end(); iSEL++)
	{

// get port number
		strTemp = (*iSEL).strPort;
		nCol1 = strTemp.find_first_of("(");
		nCol2 = strTemp.find_first_of("/");
		strTemp = strTemp.substr(nCol1 + 1, nCol2 - nCol1 - 1);
```

```
                    strTemp2 = (*iSEL).strSeverity;

// compare port numbers
                    if (strTemp == strPortNumber && strTemp2 == strType)
                    {

// find plugin in plugin vector
                              for(iPLG = (*pvecPLG).rbegin();
                                        iPLG != (*pvecPLG).rend() && bFoundValue == false;
                                        iPLG++)
                              {

                                        if ((*iSEL).strPlugin == (*iPLG).strPlugin)
                                        {
                                                  bFoundValue = true;
                                        }

                              }

// if value is not found in host vector, add hostname to host vector
                              if (bFoundValue == false)
                              {

                                        PTemp = new PLUGDESC;
                                        (*PTemp).strPlugin = (*iSEL).strPlugin;
                                        (*PTemp).strDescription = (*iSEL).strDescription;
                                        (*pvecPLG).push_back(*PTemp);
                                        delete PTemp;

                              }
// reset bFoundValue
                              bFoundValue = false;

                    }

          }

// output
          cout << endl << "<list>" << endl;

          for(iPLG = (*pvecPLG).rbegin(); iPLG != (*pvecPLG).rend(); iPLG++)
          {
                    cout << (*iPLG).strPlugin << endl;
          }

          cout << "</list>" << endl;

          cout << endl << "[done @ " << (*pvecPLG).size() << " vulnerabilities on port "
                    << strPortNumber << "]" << endl << endl;

          delete pvecPLG;

} // portVulnerabilities

// --- returnOS ---
//
// ----------------

string returnOS(string strHost)
{

          bool                              bFoundValue = false;
          vector<OPERSYS>::iterator iOS;

          for (iOS = vecOS.begin(); iOS != vecOS.end() && bFoundValue == false; iOS++)
```

```
	{
		if ((*iOS).strHost == strHost)
			return (*iOS).strOS;
	}

	return ("<no information>");

}

//

void findDescription(string strDescription)
{

	bool		bFoundValue = false;	// value found flag
	HOST*		HTemp;			// temporary host
	string		strTemp;		// temporary string container

	pvecHosts = new vector<HOST>;

	vector<SELECTION>::iterator iSEL;
	vector<HOST>::reverse_iterator iHOST;

// crawl main data vector for hostnames
	for(iSEL = vecMainData.begin(); iSEL != vecMainData.end(); iSEL++)
	{

// get port number
		strTemp = (*iSEL).strDescription;

// compare port numbers
		if (strTemp.find(strDescription) != string::npos)
		{

// check host vector for hostname found in vecMainData
// reverse iterator chosen for "rule of proximity"
			for(iHOST = (*pvecHosts).rbegin();
				iHOST != (*pvecHosts).rend() && bFoundValue == false;
				iHOST++)
			{

				if ((*iSEL).strHost == (*iHOST).strHost)
				{
					bFoundValue = true;
				}

			}

// if value is not found in host vector, add hostname to host vector
			if (bFoundValue == false)
			{

				HTemp = new HOST;
				(*HTemp).strHost = (*iSEL).strHost;
				(*pvecHosts).push_back(*HTemp);
				delete HTemp;

			}

// reset bFoundValue
			bFoundValue = false;

		}

	}
```

```
// output
	cout << endl << "<list>" << endl;

	for(iHOST = (*pvecHosts).rbegin(); iHOST != (*pvecHosts).rend(); iHOST++)
	{
		cout << (*iHOST).strHost;
		cout << ": (";
		cout << returnOS((*iHOST).strHost);
		cout << ")" << endl;

	}

	cout << "</list>" << endl;

	cout << endl << "[done @ " << (*pvecHosts).size() << " systems with specified "
		 << "description: " << strDescription << "]" << endl << endl;

	delete pvecHosts;
}

//

void findDescription2(string strInfo, string strInfo2)
{

	bool		bFoundValue = false;	// value found flag
	HOST*		HTemp;			// temporary host
	string		strTemp;		// temporary string container

	pvecHosts = new vector<HOST>;
	vector<HOST>	vecBoth;

	vector<SELECTION>::iterator iSEL;
	vector<HOST>::reverse_iterator iHOST;

// crawl main data vector for hostnames
	for(iSEL = vecMainData.begin(); iSEL != vecMainData.end(); iSEL++)
	{

// get port number
		strTemp = (*iSEL).strDescription;

// compare port numbers
		if (strTemp.find(strInfo) != string::npos)
		{

// check host vector for hostname found in vecMainData
// reverse iterator chosen for "rule of proximity"
			for(iHOST = (*pvecHosts).rbegin();
				iHOST != (*pvecHosts).rend() && bFoundValue == false;
				iHOST++)
			{

				if ((*iSEL).strHost == (*iHOST).strHost)
				{
					bFoundValue = true;
				}

			}

// if value is not found in host vector, add hostname to host vector
			if (bFoundValue == false)
			{

				HTemp = new HOST;
				(*HTemp).strHost = (*iSEL).strHost;
```

```
                                        (*pvecHosts).push_back(*HTemp);
                                        delete HTemp;

                                }

// reset bFoundValue
                                bFoundValue = false;

                        }

                }

                for(iSEL = vecMainData.begin(); iSEL != vecMainData.end(); iSEL++)
                {

// get port number
                        strTemp = (*iSEL).strDescription;

// compare port numbers
                        if (strTemp.find(strInfo2) != string::npos)
                        {

// check host vector for hostname found in vecMainData
// reverse iterator chosen for "rule of proximity"
                                for(iHOST = (*pvecHosts).rbegin();
                                        iHOST != (*pvecHosts).rend() && bFoundValue == false;
                                        iHOST++)
                                {

                                        if ((*iSEL).strHost == (*iHOST).strHost)
                                        {
                                                bFoundValue = true;
                                                HTemp = new HOST;
                                                (*HTemp).strHost = (*iSEL).strHost;
                                                (vecBoth).push_back(*HTemp);
                                                delete HTemp;

                                        }

                                }
// reset bFoundValue
                                bFoundValue = false;

                        }

                }

// output
                cout << endl << "<list>" << endl;

                for(iHOST = (vecBoth).rbegin(); iHOST != (vecBoth).rend(); iHOST++)
                {
                        cout << (*iHOST).strHost;
                        cout << ": (";
                        cout << returnOS((*iHOST).strHost);
                        cout << ")" << endl;

                }

                cout << "</list>" << endl;

                cout << endl << "[done @ " << (vecBoth).size() << " systems with specified "
                         << "description: " << strInfo << " & " << strInfo2 << "]" << endl << endl;

                delete pvecHosts;
}
```

Vita

James P. Ashe

personal data: date of birth – October 18, 1975
place of birth – Gloversville, New York

education: Gloversville High School, Gloversville, New York, 1994
State University of New York at Plattsburgh; Plattsburgh, NY
Secondary Education / Mathematics, B.A., 1998
East Tennessee State University; Johnson City, TN
Computer Science, M.S., 2004

professional: Systems Analyst, East Tennessee State University, 2002-2004
Graduate Assistant, Office of Information Technology, East Tennessee State University, 2000-2002
PC Analyst Assistant, Citizens Communications; Johnstown, New York, 1998

honors: Upsilon Pi Epsilon, Computer Science honor society, 2002