

2008

SnoScan: An iterative functionality service scanner for large scale networks

Jonathan William Murphy
Iowa State University

Follow this and additional works at: <https://lib.dr.iastate.edu/etd>

 Part of the [Electrical and Computer Engineering Commons](#)

Recommended Citation

Murphy, Jonathan William, "SnoScan: An iterative functionality service scanner for large scale networks" (2008). *Graduate Theses and Dissertations*. 11147.

<https://lib.dr.iastate.edu/etd/11147>

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

SnoScan: An iterative functionality service scanner for large scale networks

by

Jonathan William Murphy

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE

Co-Majors: Information Assurance; Computer Engineering

Program of Study Committee:
Douglas Jacobson, Major Professor
Thomas Daniels
Clifford Bergman

Iowa State University

Ames, Iowa

2008

Copyright © Jonathan William Murphy, 2008. All rights reserved.

TABLE OF CONTENTS

LIST OF FIGURES	iii
LIST OF TABLES	iv
ABSTRACT	v
CHAPTER 1. Introduction	1
CHAPTER 2. Background	3
2.1 Service Scanners	3
2.1.1 Nagios	4
2.1.2 Zabbix	7
2.2 ISEAGE	9
2.2.1 Cyber Defense Competitions	11
2.2.2 IT ADVENTURES	12
2.2.3 Information Assurance Capstone Design Course	13
2.3 SnoScan	15
CHAPTER 3. IMPLEMENTATION	16
3.1 Overview	16
3.2 Baseline	19
3.3 Configuration Creator and Database	20
3.4 Nagios Plug-in and Result Table	23
3.5 Result Interface	25
3.6 SnoScan Internals and Functionality	29
3.6.1 Secure Shell	30
3.6.2 File Transfer Protocol	32
3.6.3 Internet Message Access Protocol and Simple Mail Transfer Protocol	33
3.6.4 Remote Desktop Protocol	35
3.6.5 HTTP	36
CHAPTER 4. RESULTS	37
4.1 Findings	37
4.2 Limitations	38
4.2.1 SSH Limitations	38
3.6.5 FTP Limitations	39
3.6.5 IMAP and SMTP Limitations	39
3.6.5 RDP Limitations	39
3.6.5 HTTP Limitations	40
CHAPTER 5. conclusion	41
5.1 Future Work	41
BIBLIOGRAPHY	43
ACKNOWLEDGEMENTS	45

LIST OF FIGURES

Figure 1. Nagios status window.....	6
Figure 2. Zabbix auto-discovery interface.....	8
Figure 3. ISEAGE Architecture.....	11
Figure 4. SnoScan Flow chart.....	17
Figure 5. SnoScan internal layout.....	18
Figure 6. Nagios configuration files from the 2008 ISU CDC.....	19
Figure 7. ISU CDC Service Scanner Setup Page	21
Figure 8. Backend Database for SnoScan.....	22
Figure 9. Result Database for SnoScan	24
Figure 10. Result File from Testing.....	25
Figure 11. Result interface for SnoScan	26
Figure 12. In-depth view of Result File from SnoScan Interface.....	28

LIST OF TABLES

Table 1: Secure Shell test cases	31
Table 2: FTP test cases	32
Table 3: IMAP and SMTP test cases	34

ABSTRACT

Every day we rely on various integral parts of infrastructure to function as a society. These infrastructures, such as power, water, and roads, are all abstract representations of networks. Each of these networks is monitored so that their usability and functionality remain intact. SnoScan is a service scanner for computer networks to allow the constant monitoring and notification of their status with regards to their functionality. While there are many open source service scanners available, they will simply tell you if the service is present or not, but can not recognize an error in functionality. By testing for functionality, an in-depth picture of the status of the network can be revealed and present errors to administrators that would otherwise go unnoticed. A functionality service scanner is only useful when the relevance and importance of knowing the functionality is significant to the service in question. If it is enough to say that if the service is present, the service is intrinsic and requires no further testing. However, if knowing the service is present does not give sufficient evidence that the service is working as intended, further testing of the system is required. SnoScan is intended to monitor computer networks by emulating user behavior and initiating status checks to various services typically hosted in a home or business setting. Knowing that the service is functionally sound will provide more relevant information than a standard service scanner and ultimately a more stable network.

CHAPTER 1. INTRODUCTION

With networks becoming larger and more interconnected, the increasing points of failure grow exponentially. Services are provided as a product to customers and as such must be held accountable for their availability, reliability, and security. Each user must be able to put their trust into the service, relying on it to perform various tasks. Most of these systems are passively monitored so that there is reliability of service and metric data to support claims of stability. If a corporation relies on its database systems to function properly, they put effort into making sure that system is as robust as possible. It is simply not enough to know if the system is present on the network, but to tell if it is working as functionally expected. It is useful to know a simple binary explanation of whether or not the service is available, but a deeper view gives a better diagnosis as to what is happening on the network. A good example would be if someone has a pulse, it means their heart is beating, but it does not tell you if it is functioning properly. The same can be said for a web server. If the web server returns an ACK packet from port 80, it does not necessarily mean that it is functioning as expected. While if the pulse or the ACK packets are absent, that is a problem and very important deduction, but it does not identify the underlying cause.

SnoScan was created to deal with the increasing difficulty of monitoring functionality of services instead of status of services. The first implementation was developed for the Internet Scale Event and Attack Generation Environment (ISEAGE) at Iowa State University. ISEAGE allows us to host multiple networks within a contained environment for testing purposes of various software and hardware configurations. This allows us to focus on testing these machines with no external interference. This paper will

detail the pitfalls of common service scanners, detail how SnoScan performs functionality tests, model the various types of tests and how the data is represented, and the limitations of a functionality service scanner. First, I will go into detail about what a service scanner is and does, comparing current open source implementations of service scanners, their effectiveness and ultimately their downfalls. It is important to note the importance of their limitations help differentiate the difference between functionality service scanning and status service scanning.

CHAPTER 2. BACKGROUND

Computer functionality is a difficult problem to tackle in its inherent nature that each set of functionality tests are subjective to whom the machine is important to. If a web server hosts an SQL database, the system administrator would say that the database is more important than the underlying functionality of the web server that hosts the content. The web developer would argue that keeping content available at all times is most important. With these subjective views, it is best left to the client to monitor what they feel is the most appropriate functionality set and keeps the tests as anonymous as possible with respects to which service is integral.

2.1 Service Scanners

Network monitoring and service scanning describes the use of a system that constantly monitors a computer network for slow or failing components and that notifies the network administrator incase of outages.[1] The usefulness of these network monitors is to tell the base state of functionality of said machine. For example, to determine the status of a web server, monitoring software may periodically send an HTTP request to fetch the page. The inherent problem with this is a faulty or compromised system may still a response to the query, but the underlying data or functionality is gone. Most of the open source scanners will be able to passively monitor services as well as actively monitor them.

Passive monitoring is a semi-timed query for the service to see if it returns an expected response. Typically passive monitoring is easier to do but provides many more false positives. This is partially due to the inherent nature of discreet testing techniques.

Many passive scanners cannot cope with delay, time out, redirects, or obscure protocol behavior. Passive monitors do not interact with the system other than making the initial connection and testing whether or not that connection was valid.

Active monitoring is when a client agent is installed on the computer that you are trying to monitor. This client will either connect to or be connected to a server that controls the monitoring functions. These functions can range from system level information, such as disk space and memory usage, to service information, such as web server load and SQL database entries. Active monitoring is much more accurate than passive in the exact nature of the monitoring. The scanner is having a greater role; it is having a back and forth participatory conversation instead of a one-sided diagnosis. Active monitoring has a major downfall in that the agent installed must be configured and compiled for each and every machine monitored. While many hosts will be similar in functionality, very few will be exactly alike. This causes much more overhead than what should be required. For the ISEAGE project, since it is reconfigurable and needs to be able to be adapted to on-the-fly, a solution is needed that provides the activeness of agent level scanning but the passiveness to remain un-reliant on underlying management issues. This is a major issue when the size of a network scales to something much larger. ISEAGE is able to emulate the entire IP range, and being able to monitor a sliver of that range needs something new entirely.

2.1.1 Nagios

Nagios is an open source host and service monitoring system designed to inform you of network problems before your clients, end-users, and managers do. It has been designed to run under the Linux operating system, but works fine under most UNIX variants as well.

The monitoring daemon runs intermittent checks on hosts and services you specify by using external “plug-ins” which return status information to Nagios. When problems are encountered, the daemon can send notifications to administrative contacts in a variety of ways. [2]

Nagios is one of the largest open source monitoring tools. It has a vast community following with many conferences and web forums devoted to its development. It is managed by a variety of scripts that build a system baseline in which you can view your network at any given time for a snapshot of its status. Nagios itself is very complex, mainly because of the diversity of its functionality. Almost every aspect of Nagios can be customized to fit the needs of the network; there are still aspects that require a personal touch to be able to do. While there are plug-ins to keep track of certain things within Nagios, none of which have been adapted to perform functionality tests instead of status tests.

In the work done by Stefan Worm on monitoring of large-scale computer clusters, he details out how Nagios monitors such services, he uses what is known as the agent module of Nagios ,or Nagios Remote Plugins Executor (NRPE) which is installed on the target machine and executes commands given by the server architecture. This, unfortunately, is unable to act as a third party user accessing the system.

Each host that is being monitored needs its’ own definition file as well as a definition for each service to be monitored. Each host then needs to be grouped into a major *hostgroup* which ties common hosts together based on their role within the network or tie the whole network in as one. The screenshot below is an example of the Nagios display of a network of host being monitored with multiple services on each host.

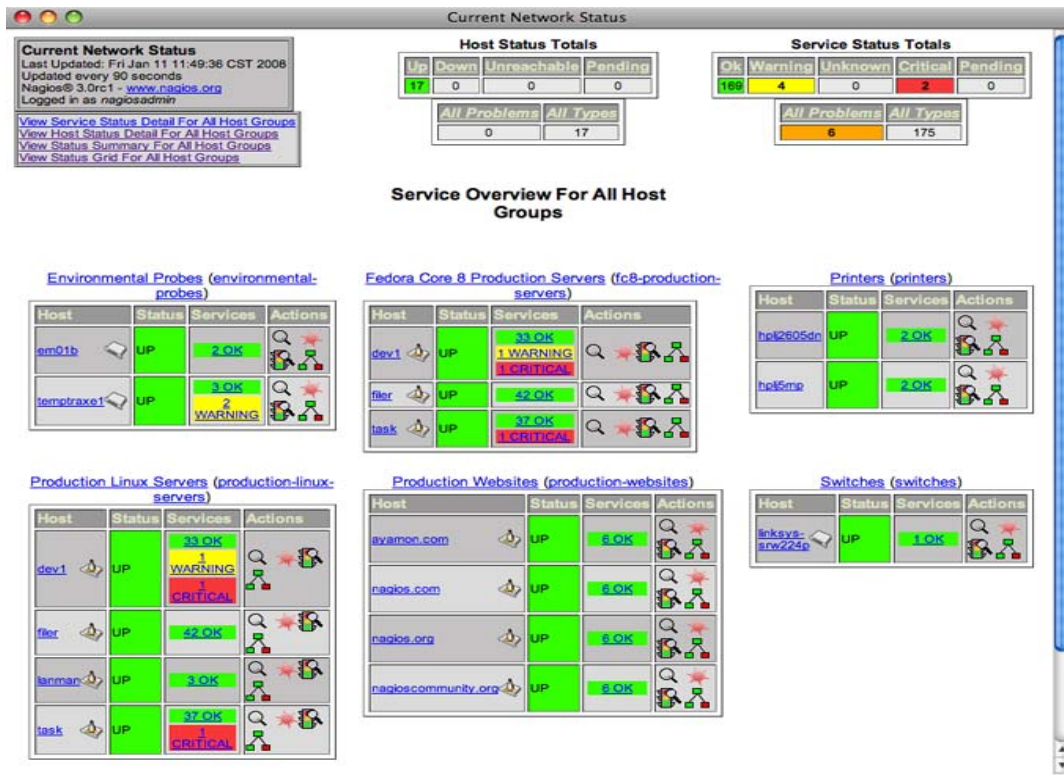


Figure 1. Nagios status window

The window displays a plethora of useful information, with more information about each host by clicking on them. Each service being monitored will give the administrator knowledge about the last test that it ran and the results. It is simple to read and effective in displaying the health of the overall network.

2.1.1.1 Limitations

Nagios has two major parts, a passive scanner and an active scanner. Without the ability to utilize the agent component of Nagios, it is infeasible to monitor the active functionality of each machine. It is not practical to rely on an active scanner due to the invasive nature of active scanners, providing unrealistic results.

To keep a fully functional Nagios configuration for multiple networks going and changing over time requires constant monitoring and rewriting of tedious configuration files. For networks that continue to grow, such as ISEAGE, it becomes infeasible to expect the constant attention to configuration files when the time can be better spent reviewing service scanning logs and statistical data to prevent future outages.

Keeping track of the history of a service is just as important as knowing the status of a service. It gives a window into what has happened and can predict what might happen in the future. If the system administrator can see when a server fails and knows what triggered the failure, it is possible to remedy the situation before it causes further complications. The simplest way of monitoring is, if the monitoring server actively checks the monitoring client directly via a network connection – for this, the server sends a request message to the client which sends an answer and finally the server processes the answer. Hereupon the server performs an adequate reaction to it which can be also caused by a missing answer, depending on the configuration of the monitoring application. With this approach of monitoring, for example, it can be checked if the client is up. [11] While it is true to check whether the service is up as detailed by Stefan Worm, unfortunately it does not tell us if the function is working as intended.

2.1.2 Zabbix

Zabbix is an enterprise-class open source distributed monitoring solution. Zabbix is software that monitors numerous parameters of a network and the health and integrity of servers. It uses a flexible notification mechanism that allows users to configure e-mail based alerts for multiple events. [6]

One of the major features of Zabbix that Nagios does not provide is event over time tracking. This is extremely important when trying to monitor trends over time and report a “six sigma” type of service availability to customers. The other major feature is the auto-discovery which does exactly as it sounds, finds servers on network and populates them into the system. Auto-discovery makes possible the evolution of networks without extensive overhead of manually reconfiguring the scanner to keep track of said hosts. Below is a screenshot of the auto-discovery feature built into Zabbix monitoring two separate IP ranges.

The screenshot shows the Zabbix web interface for the 'Status of discovery' page. The interface includes a navigation menu with options like Monitoring, Inventory, Reports, Configuration, and Administration. The main content area displays a table of discovered hosts, categorized into 'ieage2' and 'Local network'. Each host entry shows its IP address, uptime/downtime, and connectivity status for FTP, HTTP, ICMP Ping, SSH, and TCP. The status is indicated by red (down) or green (up) squares.

Host	Uptime/Downtime	FTP	HTTP	ICMP Ping	SSH	TCP
ieage2 (7 Items)						
199.100.16.100	15 days, 21:57:05					
199.100.16.142	15 days, 21:57:04					
199.100.16.150	15 days, 21:57:04					
199.100.16.16	15 days, 21:57:06					
199.100.16.20	15 days, 21:57:06					
199.100.16.250	15 days, 21:57:04					
199.100.16.50	15 days, 21:57:05					
Local network (5 Items)						
192.168.1.1	15 days, 19:18:57					
192.168.1.100	15 days, 16:16:24					
192.168.1.102	15 days, 19:08:37					
192.168.1.103	10 days, 21:25:43					
192.168.1.104	15 days, 19:08:20					

Figure 2. Zabbix auto-discovery interface

2.1.2.1 Limitations

Zabbix was the primary replacement choice for Nagios when originally comparing the two, but came under harsh criticism when trying to add customized commands into its interface. While the scanner will auto-discover hosts and monitor them with event over time tracking, it is not accurate to any scale in which would be effective. In preliminary testing of Zabbix, it took between 5 and 25 minutes for it to ‘discover’ that the hosts were no longer present on the network. This large gap in time can be serious concern for any system administrator. Without the ability to keep a strict rule set in place, it became unpractical to use Zabbix as a baseline for SnoScan. The other major hindrance was the inability to automate the input of hosts directly into the system without the auto-discovery interface. To add a host to the system required the use of a web interface manually entering in each system parameters, which is far slower than typing them out for Nagios.

2.2 ISEAGE

The Internet-Scale Event and Attack Generation Environment (ISEAGE) is an internet test bed developed at the Iowa State University Information Assurance Center. The goal of ISEAGE is to provide a world-class research and education facility to enhance the current state of the art in information assurance. Dedicated to creating a virtual Internet for the purpose of researching, designing, and testing cyber defense mechanisms, the one-of-a-kind facility will be the catalyst for bringing together top researchers from several disciplines for a common goal of making computing safer. Unlike computer-based simulations, real attacks will be played out against real equipment. [3]

Many researchers and vendors are working hard to provide products and services to help defend against cyber attacks. Users of these technologies often do not have any mechanisms to test or even try out these defenses. Researchers and vendors need a facility to test ideas against real-world attacks. The ISEAGE will provide a controlled environment where real-world attacks can be played out against different configurations of equipment. It will contain a vast warehouse of attack tools that will be able to simulate point-to-point and distributed attacks against test configurations. [4]

ISEAGE will provide an integrated environment for them to work on synergistic research projects in information assurance. The information assurance research work at Iowa State deals with overlapping problems and can benefit from a common laboratory. For example, the group working in intrusion detection and the group working in survivable networks often utilize the same attack data and are concerned about the same types of attacks. The proposed laboratory is a critical element needed to elevate research efforts and help provide solutions to these complex problems. [4]

Because of the complexity of many large scale networks and monitoring the status of each, ISEAGE proved to be the perfect test bed for SnoScan. Each test case was enacted as a third party connecting from a separate network to each target host.

ISEAGE Architecture

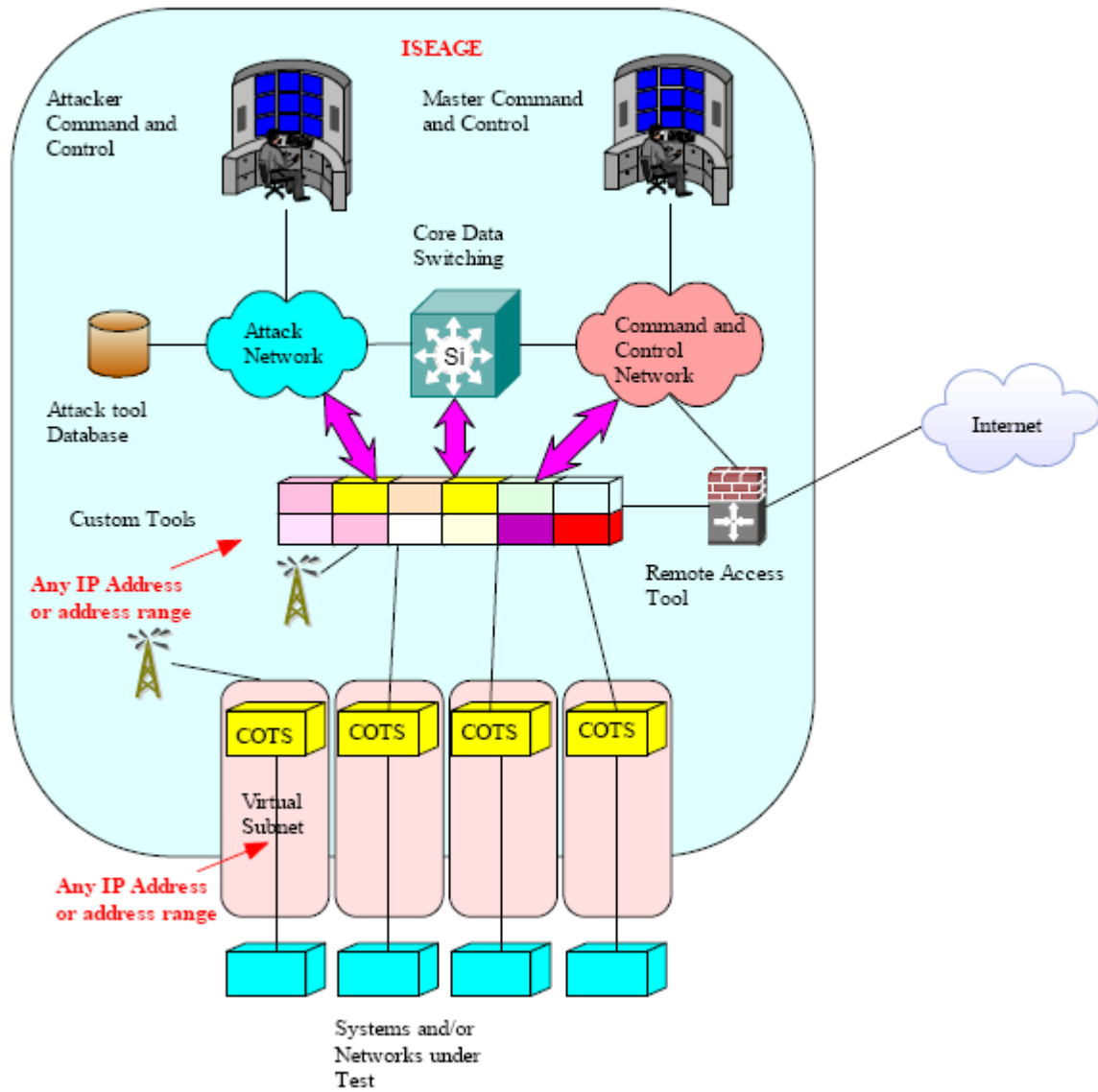


Figure 3. ISEAGE Architecture

2.2.1 Cyber Defense Competitions

Iowa State University plays host to multiple Cyber Defense Competitions each year. CDC's are a chance to explore and demonstrate practical applications of Information Assurance defense and attack techniques in a secure environment. Teams are given

resources to build and secure a network isolated in the ISEAGE environment. These networks are then attacked during the night by a team of experienced intrusion specialists whose sole goal is to bring down the networks in the most entertaining way possible. Teams are scored for security of their networks, quality of their documentation, usability of their systems, and their participation in other fun events during the course of the competition. [5]

The CDC's must be monitored and scored with strict consistency. While each test would be considered subjective based on the response generated from a test, if the tests are uniformly scored on a binary scale, then it is possible to remove the subjectivity involved. Each team is required to build and monitor the stability and usability of each system as well as have it comply with the guidelines set in place for the contest. These guidelines are meant to emulate real world conditions in which a user would encounter on any given day. If the functionality were disrupted, then the user will be unable to do what they need to.

Agent monitors on each client would be a best-case solution if we wanted an artificial status monitor. Agent monitoring daemons would be subjected to different sets of credentials than a user logging into the server. The agent could have altered permissions to do various functions that a normal user would not and obfuscate results. Each functionality test will be uniformly similar regardless of underlying software or operating systems on the host server.

2.2.2 IT ADVENTURES

IT Adventures is an innovative program that engages Iowa high school students in exploration and experimentation with Information Technology (IT) through events and service learning projects. [7] One of the programs in which the high school students

participate is a cyber defense competition. The high school teams (the Blue Teams) play the role of IT support staff. They configure a network of four computers and provide services to the end users of their network (the Green Team) throughout the event. They also must defend their network for an extended period of time from hackers (the Red Team). In addition to configuring and protecting their network, the Blue Team will be asked to participate in anomalies introduced by the Green Team. By successfully completing each of the tasks they better their score for the competition. The Green Team anomalies and defending from the Red Team attacks comprise the real-time portion of the competition.

The high school teams are provided a scenario of a company and the types of services they are required to run on their networks in a separate document. Generally, they are required to provide domain name service (DNS), email (SMTP and POP/IMAP), file transfer protocol (FTP), remote programming and web services. [7]

IT Adventures takes the ISU CDC to a new level. The scale of the event grows from just a handful of teams to 25+, increasing the overhead of scoring and monitoring significantly. By providing a baseline monitoring, and subsequently scoring, systems, the ability to expand IT Adventures to a larger scale becomes easier from a management perspective.

2.2.3 Information Assurance Capstone Design Course

The initial concept of SnoScan was for the IT-Olympics and CDC's held at Iowa State, but not limited to these areas. Iowa State created the Information Assurance Capstone Design course as an experimental course that emulates all aspects of a CDC but takes them a step further. The capstone design course is broken into four major sections: Security design

process and security plan creation, Implementation of a security plan, attack and defend the implementations, and reporting and analysis. The students will attack each other's secure environments in an effort to defeat the security systems to ultimately learn what security means. Students then evaluate the security plans and the performance of the plans. Other topics include social, political and ethics issues, journaling, written reports, and an oral report.

The capstone course will encompass all focal points of the degree, culminating in a three part capstone course. The three parts include the planning & implementing, defense & attack, and assessment of various parts of a network.

Students would be required to plan a network to include many of the services that are provided in an average corporation, including mail, web, programming environments, etc. While on a much smaller scale, the intent is to give an idea of why certain security aspects are important in their implementation. Each student would be required to give cost/benefit analysis on why certain elements were chosen and what other alternatives are available if the options presented were not applicable.

The next portion of the course would be actual implementation of the networks they described after their networks were approved by the course instructor and/or teaching assistant. It is important that security professionals get hands on approach for the systems and devices that they are securing. Following the setup and preparation phase, students would then be able to do penetration testing and network analysis against their own and other student's networks.

Risk assessment in the security field is a vital part. As a final for the course, each student would have a written assessment of the security and vulnerability of their networks,

what can be improved and what must be changed. Being a distance course, students would be required to create a paper and then submit an oral/video presentation of their risk assessment analysis.

The capstone design course is a vital part of learning information assurance. By getting a hands-on approach to security and seeing the entire process from the initial security plan to implementation and ultimately reporting, it culminates the knowledge garnered while in the Information Assurance masters program.

2.3 SnoScan

The purpose for a monitoring tool that is able to play both a passive and active role without an agent present on each machine allows us to keep all test cases consistent between the uniqueness of each machine and focus on the functionality of the service in question. Each service is expected to act in a specific manner. If we are able to eliminate other noise and focus on the exact functionality of a service, we are able to objectively rate the functionality of it. While SnoScan is not able to monitor every service available, I chose to focus on the services that are typically common on a given network with regards to a CDC. If the service can fit inside of a uniform framework, it is possible to expand SnoScan to monitor them as well.

CHAPTER 3. IMPLEMENTATION

This chapter is geared to the implementation of SnoScan. It focuses on defining the problem and arriving at a solution that is suitable for the needs of the system. Each problem defined has a range of possible solutions, contrasting the differences between each and arriving at a suitable baseline. The chapter ends with a short depiction of the tool in operation and what results are gathered.

3.1 Overview

SnoScan is meant to meet the challenge of a specific problem. Testing functionality is subjective in the sense that each service is used differently. By creating a common baseline functionality test, one can assume that the functionality is working as intended. There are multiple parts of a service scanner and each play an important role. The initial setup and configuration requires deep knowledge on what the final implementation will be. Below is a flow diagram of the overall process of SnoScan, from creation to result data sets.

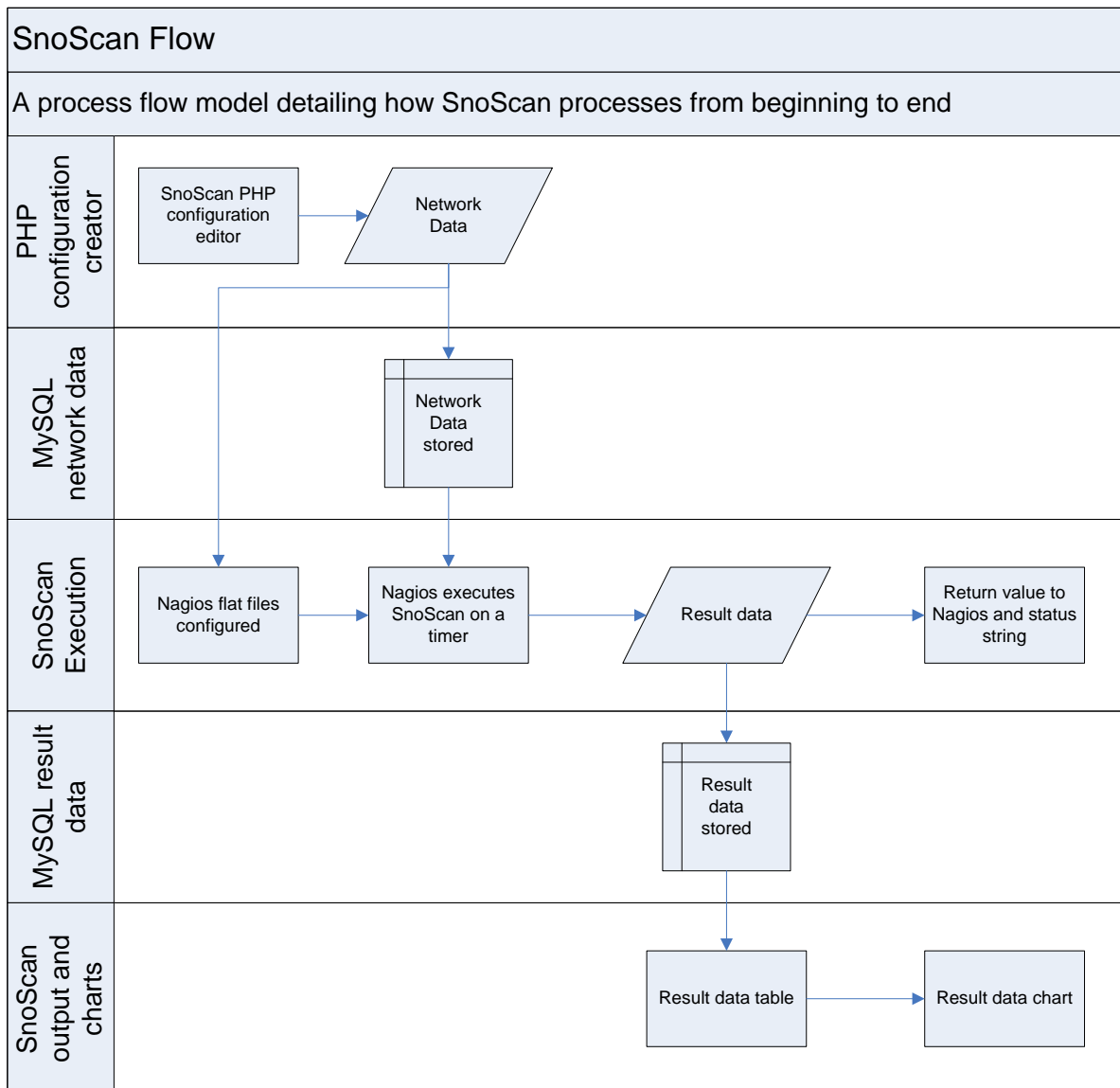


Figure 4. SnoScan Flow chart

The above diagram lays out how the process and data flows between the different sections of the overall process of SnoScan. The data originates from creation in a PHP web page by the user or administrator. The data gets passed into both a MySQL table as well as flat text files for Nagios to manage the network properly. Once the information is stored in both Nagios and a MySQL database, Nagios will execute SnoScan. SnoScan will then be

executed with the settings provided by the administrator on the target systems. This test will require a conditional statement to verify the results executed by the command. Once the conditional statement is executed, the resulting value is passed to both a MySQL database and Nagios for an instantaneous snapshot. Once the data enters the MySQL database, a dynamic PHP page is used to display the network data, and as such provide a graph to display a graphical representation of network uptime. Below is a snapshot of the flow within the execution of SnoScan itself.

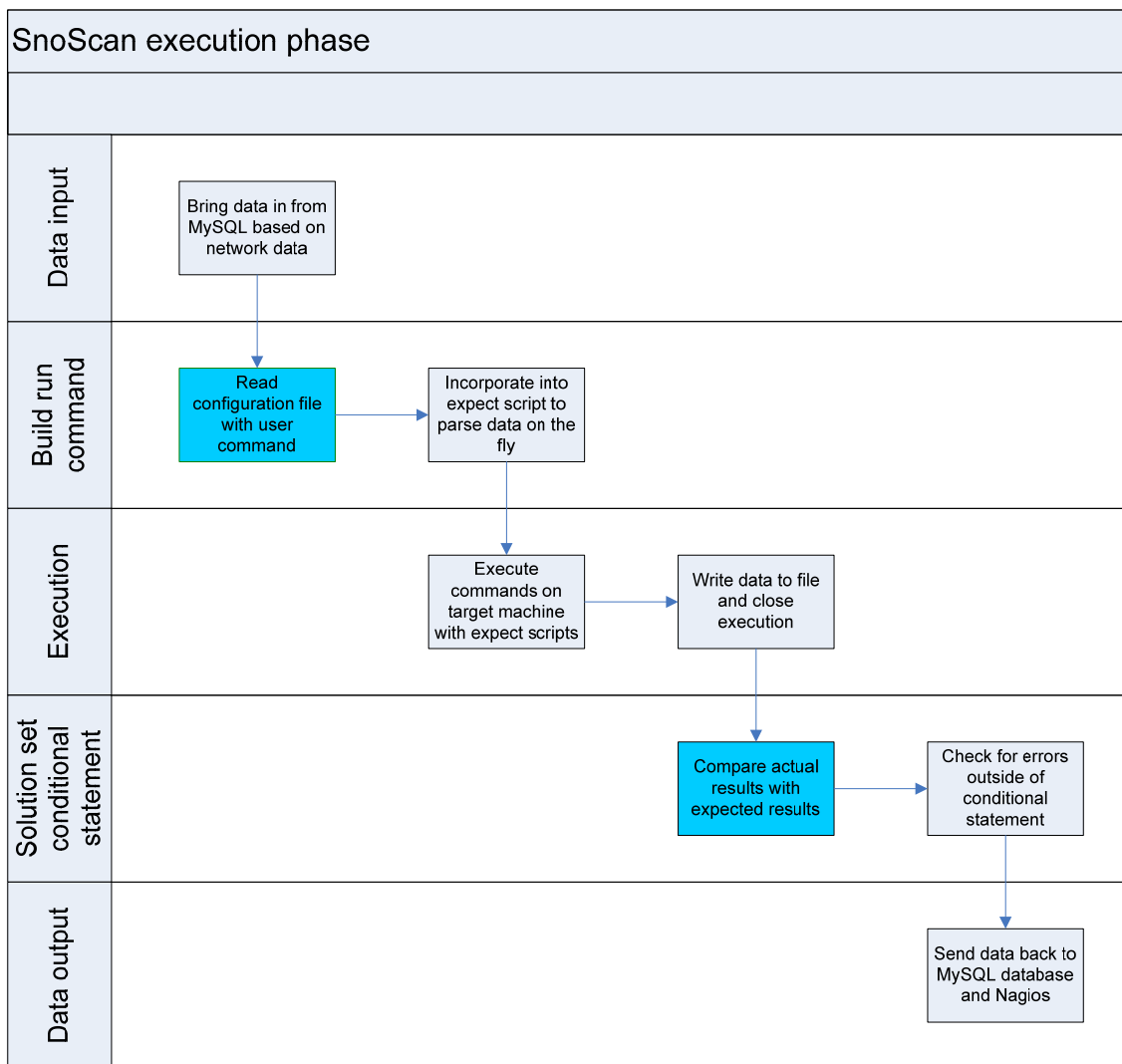


Figure 5. SnoScan internal layout

3.2 Baseline

After evaluating many of the open source service scanners for their effectiveness and reliability, I have selected Nagios as a baseline for SnoScan. While it is entirely possible to develop a full functionality service scanner without Nagios, it does provide a few features that are useful without re-inventing the wheel. One of the inherent difficulties as stated before was the initial setup management of Nagios. To add or modify services, the administrator needs to modify configuration files much like the one below:

```

root@Scanner: /usr/local/nagios/etc/scanner
-rw-r--r-- 1 nagios nagios 143 2008-10-17 14:46 Team7_IMAP_service_by_hostname.cfg
-rw-r--r-- 1 nagios nagios 187 2008-10-17 14:46 Team7_SMTP_host_by_hostname.cfg
-rw-r--r-- 1 nagios nagios 143 2008-10-17 14:46 Team7_SMTP_service_by_hostname.cfg
-rw-r--r-- 1 nagios nagios 189 2008-10-17 14:46 Team7_SSH_host_by_hostname.cfg
-rw-r--r-- 1 nagios nagios 142 2008-10-17 14:46 Team7_SSH_service_by_hostname.cfg
-rw-r--r-- 1 nagios nagios 184 2008-10-17 14:46 Team7_WWW_host_by_hostname.cfg
-rw-r--r-- 1 nagios nagios 141 2008-10-17 14:46 Team7_WWW_service_by_hostname.cfg
-rw-r--r-- 1 nagios nagios 199 2008-10-17 17:33 Team8_DNS_host_by_hostname.cfg
-rw-r--r-- 1 nagios nagios 148 2008-10-17 17:33 Team8_DNS_service_by_hostname.cfg
-rw-r--r-- 1 nagios nagios 199 2008-10-17 17:33 Team8_FTP_host_by_hostname.cfg
-rw-r--r-- 1 nagios nagios 148 2008-10-17 17:33 Team8_FTP_service_by_hostname.cfg
-rw-r--r-- 1 nagios nagios 191 2008-10-17 17:49 Team8_hostgroup.cfg
-rw-r--r-- 1 nagios nagios 151 2008-10-17 17:33 Team8_IMAP_service_by_hostname.cfg
-rw-r--r-- 1 nagios nagios 203 2008-10-17 17:33 Team8_SMTP_host_by_hostname.cfg
-rw-r--r-- 1 nagios nagios 151 2008-10-17 17:33 Team8_SMTP_service_by_hostname.cfg
-rw-r--r-- 1 nagios nagios 200 2008-11-03 18:35 Team8_SSH_host_by_hostname.cfg
-rw-r--r-- 1 nagios nagios 148 2008-10-17 17:33 Team8_SSH_service_by_hostname.cfg
-rw-r--r-- 1 nagios nagios 200 2008-10-17 17:33 Team8_WWW_host_by_hostname.cfg
-rw-r--r-- 1 nagios nagios 149 2008-10-17 17:33 Team8_WWW_service_by_hostname.cfg
root@Scanner:/usr/local/nagios/etc/scanner# cat Team8_SSH_host_by_hostname.cfg
define host{
    use                generic-host
    host_name          ssh.networkkinja.com
    alias              Team8 Shell Server
    address            ssh.networkkinja.com
    check_command      check_ssh
    contact_groups     admins
    max_check_attempts 2
}
root@Scanner:/usr/local/nagios/etc/scanner# cat Team8_hostgroup.cfg
define hostgroup {
    hostgroup_name    Team8
    alias              Team 8: Network Ninjas
    members           ssh.networkkinja.com,ftp.networkkinja.com,www.networkkinja.com,mail.networkkinja.com
,dns.networkkinja.com
}
root@Scanner:/usr/local/nagios/etc/scanner#
root@Scanner:/usr/local/nagios/etc/scanner#

```

Figure 6. Nagios configuration files from the 2008 ISU CDC

As you can see from the above screenshot, there are multiple factors that Nagios needs to know about. Each host that is being monitored needs to have its' own unique host definition file. This is where one of the difficult particulars arrives when detailing very large networks or networks that change rapidly. One can assume that editing each file is time consuming in itself, but re-creating each configuration file every time the address, hostname, or IP changes is not within the realm of sanity to the average system administrator. Each host definition file must contain information about the machine so that it can be instantiated properly within Nagios. Each host must have a pre-defined *check_command* that allows the service scanner to know what check to perform. In the above example, it is *check_ssh*. *Check_ssh* is a preconfigured C program to open a connection to the target address. Nagios will attempt to open a connection to the target host based on the address given. This address must be pre-defined by the creator of the host file. If the address is invalid or non-responsive, the test will fail. The problem with many of these passive preconfigured tests is their inability to log in after making a connection or test commands on the target host. This is where SnoScan takes precedence.

3.3 Configuration Creator and Database

One of the first steps in creating a wrapper around Nagios was to make the configuration process simpler. In computer terms, a wrapper is something that sits between one process and another. In this case, the wrapper is a system to create the configuration files for Nagios without the overhead of manually entering in each system one at a time. By creating a PHP web application, it is easier to input and modify different values of each host without changing the every *hostfile*. This PHP application is similar to those found in many

online application forms, requesting various information as well as authentication so that other users do not overwrite pre-existing information. One of the major issues with Nagios is that it must be restarted before changes to any configuration get initiated. Fortunately through PHP we are able to execute system level commands that check the validity of the config files and restart the application automatically. Below is a screenshot of the interface created for the 2008 ISU CDC:

ISU CDC Service Scanner Setup Page!

Team Code: [masked] Team Name: Beta

SSH
 FTP
 WWW
 IMAP & SMTP
 DNS
 RDP

Check All Uncheck All

SSH:
 Host Name: shell.cdc.net Host IP: 199.100.16.101 Username: scanner Password: [masked] Port: 22

FTP:
 Host Name: ftp.cdc.net Host IP: 199.100.16.102 Username: scanner Password: [masked] Port: 21

WWW:
 Host Name: www.cdc.net Host IP: 199.100.16.100 Port: 80

SMTP:
 Host Name: mail.cdc.net Host IP: 199.100.16.104 Username: testacct Password: [masked] Port: 25

IMAP:
 Host Name: mail.cdc.net Host IP: 199.100.16.104 Username: testacct Password: [masked] Port: 143

DNS:
 Host Name: ns-1.cdc.net Host IP: 199.100.16.254 Port: 53

RDP:
 Host Name: remote.cdc.net Host IP: 199.100.16.103 Username: scanner Password: [masked] Port: 3389

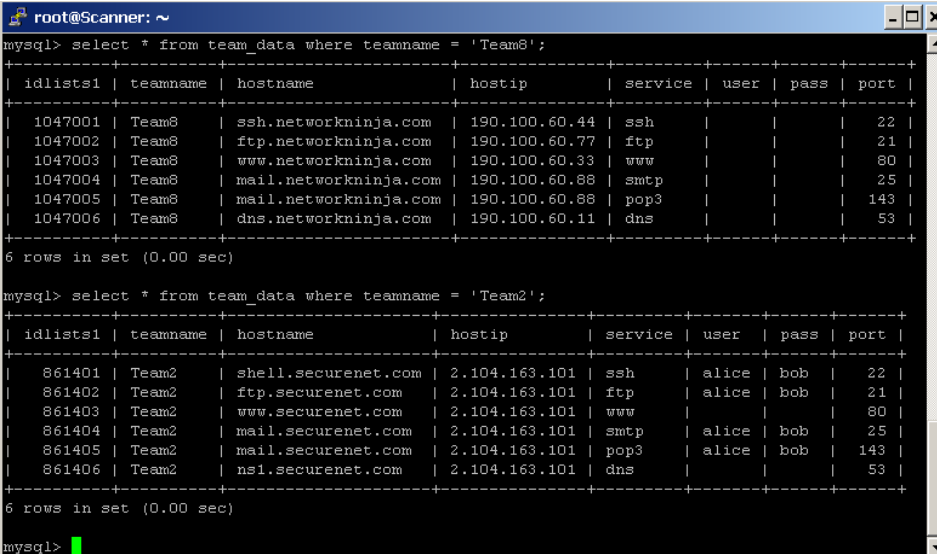
Additional Comments
 SMTP and IMAP hosted off the same server.

Submit Please only click once.

Done

Figure 7. ISU CDC Service Scanner Setup Page

While simplistic in its nature, it provides the user clear definition of what they want to control. Each field serves a purpose for the given system. Let us take FTP for example. It asks for a hostname, IP address, port, username, and password. It takes these fields and builds a host config file from it so that Nagios can interpret it. The configuration creator then takes the information provided and drops it into a database. This database holds a unique ID, teamname, host, host IP, the service that is running, a username and password, and what port it is running on. This set of data is the minimum needed to perform a connection to the target host without making the hosts less anonymous. The rationale for requesting the *host IP* in addition to the *hostname* is so that if the *hostname* fails to resolve, it is still able to attempt a connection. If the *hostname* fails but the *IP address* works, the system will send a warning instead of an error flag. The return codes are to notify the system administrator of an issue without causing alarm. If the hostname fails but the IP address works, this is an indicator that the Domain Name Server is unable to resolve the *hostname* requested. Below is a screenshot of the database from the 2008 ISU Cyber Defense Competition.



```

root@Scanner: ~
mysql> select * from team_data where teamname = 'Team8';
+-----+-----+-----+-----+-----+-----+-----+-----+
| idlists1 | teamname | hostname | hostip | service | user | pass | port |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1047001 | Team8 | ssh.networkkinja.com | 190.100.60.44 | ssh | | | 22 |
| 1047002 | Team8 | ftp.networkkinja.com | 190.100.60.77 | ftp | | | 21 |
| 1047003 | Team8 | www.networkkinja.com | 190.100.60.33 | www | | | 80 |
| 1047004 | Team8 | mail.networkkinja.com | 190.100.60.88 | smtp | | | 25 |
| 1047005 | Team8 | mail.networkkinja.com | 190.100.60.88 | pop3 | | | 143 |
| 1047006 | Team8 | dns.networkkinja.com | 190.100.60.11 | dns | | | 53 |
+-----+-----+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)

mysql> select * from team_data where teamname = 'Team2';
+-----+-----+-----+-----+-----+-----+-----+-----+
| idlists1 | teamname | hostname | hostip | service | user | pass | port |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 861401 | Team2 | shell.securenet.com | 2.104.163.101 | ssh | alice | bob | 22 |
| 861402 | Team2 | ftp.securenet.com | 2.104.163.101 | ftp | alice | bob | 21 |
| 861403 | Team2 | www.securenet.com | 2.104.163.101 | www | | | 80 |
| 861404 | Team2 | mail.securenet.com | 2.104.163.101 | smtp | alice | bob | 25 |
| 861405 | Team2 | mail.securenet.com | 2.104.163.101 | pop3 | alice | bob | 143 |
| 861406 | Team2 | ns1.securenet.com | 2.104.163.101 | dns | | | 53 |
+-----+-----+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)

mysql>

```

Figure 8. Backend Database for SnoScan

As you can see Team 8 is much different than Team 2 in its configuration, as well as did not provide a username and password. If the user chooses to ignore these fields, the tests simply fall back to the baseline Nagios tests. This will allow the systems to be passively monitored for status, but not functionality. The team data is stored with all of the necessary information to create a valid connection. Each user and password must be a valid combination as if any normal user would be using the service. This is to ensure that any outside source is able to access the service without hindrance or manipulation of a specific user to allow excess functionality. Information can vary quite differently from *hostgroup* to *hostgroup*. This is very common in the real world as well. Each network is unique, and thus each configuration is unique. The main problem with manually manipulating each configuration file is the scale. When you have four networks with five services each, that is a relatively easy number to manage, but when the count gets into the hundreds with six to seven services, manually changing each file becomes insurmountable.

3.4 Nagios Plug-in and Result Table

Nagios has a simple interface in which to create custom test scripts for use. The only thing Nagios expects is a return code. This code ranges from 0 to 3, where 0 is working as intended, and 3 is failure with no explanation. Accompanying the return code can be a string that is pulled from the test script. Each test script can decide what it deems relevant and return it to Nagios. For the default SSH, it will return the first line that is printed to the screen after successfully connecting to the server.

Nagios by default does not keep track of the past. While it is able to notify an administrator of a system failure via e-mail, it is an inaccurate history as to what the service was doing over a course of time. The system in place is useful if you wish to have a snapshot at any given time, instead we wish to keep data over time so there can be an accurate guaranteed server uptime reporting. If this were in a live environment with corporations relying on up-to the second information, most corporations tend to try to use the Six Sigma process model for detailing the reliability of its services. To do this, each of the tests must be recorded in a database for comparisons. Below is a screenshot of the database test case from the 2008 ISU CDC.

```
mysql> select * from result_data where teamname = 'team12' and idlists1 > 20;
```

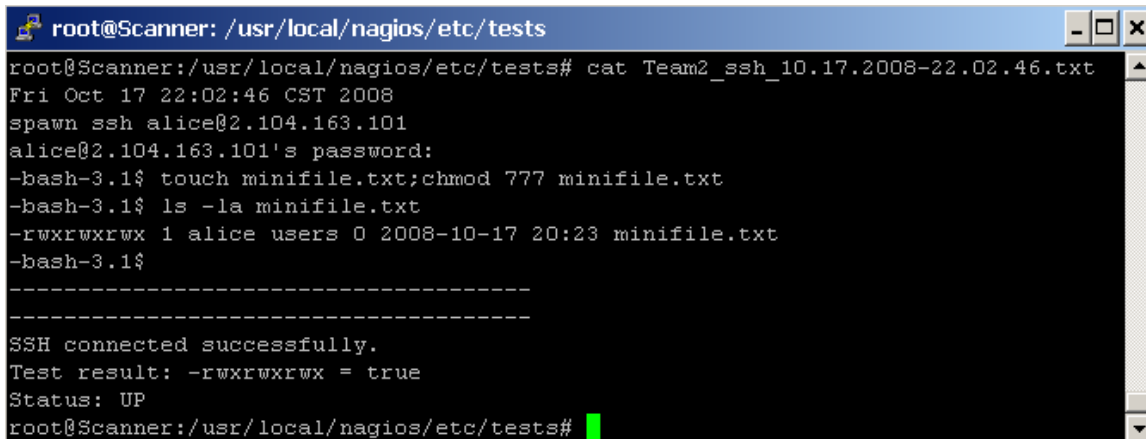
idlists1	teamname	hostname	hostip	service	port	test	status	time
23	Team12	shell.bsod.com	64.5.53.1	SSH	22	chmod 777 minifile.txt	DOWN	2008-10-17 23:13:40
29	Team12	shell.bsod.com	64.5.53.1	SSH	22	chmod 777 minifile.txt	DOWN	2008-10-17 23:44:41
30	Team12	shell.bsod.com	64.5.53.1	SSH	22	chmod 777 minifile.txt	UP	2008-10-17 23:48:49
35	Team12	shell.bsod.com	64.5.53.1	SSH	22	touch minifile.txt;chmod 777 minifile.txt	DOWN	2008-10-17 23:58:48
38	Team12	shell.bsod.com	64.5.53.1	SSH	22	touch minifile.txt;chmod 777 minifile.txt	DOWN	2008-10-18 00:03:28
39	Team12	shell.bsod.com	64.5.53.1	SSH	22	touch minifile.txt;chmod 777 minifile.txt	UP	2008-10-18 00:06:23
40	Team12	shell.bsod.com	64.5.53.1	SSH	22	ls -laR /;&sleep 10&ps -aux	DOWN	2008-10-18 00:59:23
41	Team12	shell.bsod.com	64.5.53.1	SSH	22	ls -laR /;ps -aux	DOWN	2008-10-18 01:01:54
42	Team12	shell.bsod.com	64.5.53.1	SSH	22	ls -laR / >> lol_ls.txt;ps -aux	DOWN	2008-10-18 01:03:32
43	Team12	shell.bsod.com	64.5.53.1	SSH	22	ls -laR / >> lol_ls.txt;ps -aux	UP	2008-10-18 01:06:46

```
10 rows in set (0.00 sec)

mysql>
mysql>
```

Figure 9. Result Database for SnoScan

The previous tests were run against the same host in the same format. The only field that changed was what is tested and when. It is important to know the exact time that the test was performed for heuristics. Each test also has a given output file that references the test indicated. The output has the exact commands run on the machine as well as the test validity. Each test is unique in its code for verification to give accurate return values to the Nagios implementation. Below is an actual output file from SnoScan. The data from the terminal is displayed as well as the verification that the test command succeeded.



```

root@Scanner: /usr/local/nagios/etc/tests
root@Scanner:/usr/local/nagios/etc/tests# cat Team2_ssh_10.17.2008-22.02.46.txt
Fri Oct 17 22:02:46 CST 2008
spawn ssh alice@2.104.163.101
alice@2.104.163.101's password:
-bash-3.1$ touch minifile.txt;chmod 777 minifile.txt
-bash-3.1$ ls -la minifile.txt
-rwxrwxrwx 1 alice users 0 2008-10-17 20:23 minifile.txt
-bash-3.1$
-----
SSH connected successfully.
Test result: -rwxrwxrwx = true
Status: UP
root@Scanner:/usr/local/nagios/etc/tests# █

```

Figure 10. Result File from Testing

3.5 Result Interface

The purpose of this program is ultimately providing feedback to system administrators. While the Nagios interface will give you a snapshot in time of the overall status of a network, we care more about the service availability over time. As displayed in figure 7, all of the resultant information is stored in a MySQL database, but this is generally not a friendly way to represent data to a user or administrator. By creating a web interface with various selectable fields, one can effectively view exactly what they want. Below is a screenshot of the web interface in which users and system administrators alike can view the status of a network.

SnoScan Result Files

Select a team and service to view the last 20 test results.

Team Name:

SSH FTP MAIL RDP WWW Show All

teamname	hostname	hostip	service	port	test	status	Ascend time	linkfile
Beta	www.snoscan.net	192.168.1.204	WWW	80	Check for string within page.	UP	2008-11-25 08:12:11	LINK
Beta	www.snoscan.net	192.168.1.204	WWW	80	Check for string within page.	UP	2008-11-25 08:07:11	LINK
Beta	www.snoscan.net	192.168.1.204	WWW	80	Check for string within page.	UP	2008-11-25 08:02:11	LINK
Beta	www.snoscan.net	192.168.1.204	WWW	80	Check for string within page.	UP	2008-11-25 07:57:11	LINK
Beta	www.snoscan.net	192.168.1.204	WWW	80	Check for string within page.	UP	2008-11-25 07:52:11	LINK
Beta	www.snoscan.net	192.168.1.204	WWW	80	Check for string within page.	UP	2008-11-25 07:47:11	LINK
Beta	www.snoscan.net	192.168.1.204	WWW	80	Check for string within page.	UP	2008-11-25 07:42:11	LINK
Beta	www.snoscan.net	192.168.1.204	WWW	80	Check for string within page.	UP	2008-11-25 07:37:11	LINK
Beta	www.snoscan.net	192.168.1.204	WWW	80	Check for string within page.	UP	2008-11-25 07:32:11	LINK
Beta	www.snoscan.net	192.168.1.204	WWW	80	Check for string within page.	UP	2008-11-25 07:27:11	LINK
Beta	www.snoscan.net	192.168.1.204	WWW	80	Check for string within page.	UP	2008-11-25 07:22:11	LINK
Beta	www.snoscan.net	192.168.1.204	WWW	80	Check for string within page.	UP	2008-11-25 07:17:11	LINK
Beta	www.snoscan.net	192.168.1.204	WWW	80	Check for string within page.	UP	2008-11-25 07:12:11	LINK
Beta	www.snoscan.net	192.168.1.204	WWW	80	Check for string within page.	UP	2008-11-25 07:07:11	LINK

Figure 11. Result interface for SnoScan

The table represented above is a select statement of the *result_data* MySQL table where all of the system information is stored. This select statement finds all elements where *teamname* is equal to the selected item from the drop down menu above it. The above table is a representation of all of the SSH tests done to Beta throughout the last two weeks. There are multiple different tests that were initiated against the system, testing various aspects of the functionality of the shell programming environment. The system is setup to progress through a variety of tests sequentially, moving to the next test only if the previous test has

been successful represented by a status event of UP. As represented by the table above, it shows the *teamname*, *hostname* of the system, IP that is tied to the *hostname*, port, service, command run, status, time of the test, and a link file. This link file is probably the most important only second to the status field. The link field is a direct link to the output of the test, so that if a test fails or succeeds, the administrator can see exactly what occurred. The inherent problem with a system of functionality is while you know there is only one exact result that is expected to make the test true, there can be multiple points of failure along the way. By providing the administrators access to the raw files generated by SnoScan, it opens a window into why the functionality is not working as intended. Below is a screenshot showing the link file pane below the table.

The screenshot shows a Mozilla Firefox browser window displaying a table of test results. The table has columns for host name, IP address, protocol, port, test description, status, and time. The second row is highlighted with a red border, indicating it was the one clicked. Below the table, a terminal window shows the output of the test, including the command used and the resulting file permissions.

Beta	shell.snoscan.net	192.168.1.203	SSH	22	List all directories and write results to a file.	UP	2008-11-25 07:31:52	LINK
Beta	shell.snoscan.net	192.168.1.203	SSH	22	Change permissions on file with chmod.	UP	2008-11-25 07:27:32	LINK
Beta	shell.snoscan.net	192.168.1.203	SSH	22	List all directories and write results to a file.	UP	2008-11-25 07:26:52	LINK

```

/usr/local/nagios/etc/results/Beta/Beta-ssh-11.25.08-07.27.11.txt
spawn ssh alice@192.168.1.203
alice@192.168.1.203's password:
Linux shell 2.6.24-19-server #1 SMP Wed Aug 20 23:54:28 UTC 2008 i686

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/
Last login: Tue Nov 25 07:26:52 2008 from 192.168.1.104

$ touch minifile.txt;chmod 777 minifile.txt
$ ls -la minifile.txt
-rwxrwxrwx 1 alice alice 2949239 2008-11-25 07:27 minifile.txt
$
0
=====
ssh : Connected Successfully.
Test result: touch minifile.txt;chmod 777 minifile.txt = true
Status: UP
Done

```

Figure 12. In-depth view of Result File from SnoScan Interface

When the administrator clicks on the LINK, it displays in a pane below the table showing exactly what is happening during the test. In the example above, the highlighted area was the link that was clicked, referencing the file Beta-ssh-11.25.08-07.27.11.txt. As you can see, the test to tell if the test ran successfully was the comparison between – *rwxrwxrwx* and what is gathered from the output. *-rwxrwxrwx* is the file permissions that are set when a user *chmod*'s a file. While this filename seems overly complex, it is necessary to keep track of team, service, date, and time for consistency between the systems being monitored. This does limit the naming of systems so that each host must have a unique name

with regards to its *hostgroup* Nagios. This is true when dealing with an individual network, but not so when crossing multiple domains. As an example, two separate corporations, corpA and corpB have external networks that are mutually exclusive, but each internal network names their internal mail servers mail.corp.com. To combat overlapping system names, we regress to the overall network name, in this case corpA.mail.corp.com and corpB.mail.corp.com.

3.6 SnoScan Internals and Functionality

At the heart of SnoScan is a program that does the actual functionality tests. This program is executed through the Nagios configuration scripts when the *check_command* is set to SnoScan. There are three flags that are required when running, -h for host, -s for service, and -t for test type. The host flag signifies which network group to poll the database for information. Each host inside of a hostgroup is uniquely identified by its *hostgroup.hostname*. This is to differentiate between each groups' individual networks. Since a server can play host to multiple services, the service flag tells the program which set of tests to run. Each set of tests has a variety of testing sequences inside of it. There are three options for testing functionality: singular, random, or cyclical. Once SnoScan has the parameters, it can start pulling data from the database. It pulls out all the essential information that it will need to instantiate a connection and run commands on the appropriate host.

It should be noted that the program is actually logging into the target host as the user provided when the configuration files were created. This is an important thing to remember

when monitoring for security. It is possible that SnoScan will trigger an intrusion detection or prevention system in its process of checking the functionality of a host. This is most primarily due to its “brute force” technique of testing the connection and subsequent functionality. Unfortunately if the connection is interrupted, SnoScan will interpret it as a failure and report the incident as such.

Once the parameters are read from the database, it builds a set of commands to run. This is where the test type comes into play. If the administrator specifies the singular test for a client running SSH, it will run the very basic command of “touch minifile.txt” every time the functionality check is initiated. This is a very rudimentary way to test functionality but none the less it provides some vital information. It can show that a user can login remotely via SSH and create a file, which in turn tells us the hosts’ file system is intact enough to write files to the users’ home directory and that the authentication system is working. The next test is random, which is clear by its name, the program chooses one of the multiple tests. The last test and most functional is cyclic which will run through each test, incrementing in functionality testing. While testing a gamut of functions can present problems such as file size limitations or process caps, it is not SnoScan’s job to tell whether a user should be able to have twenty simultaneous processes or not but simply to tell if there is in fact a limitation on functionality.

3.6.1 Secure Shell

SSH is a network protocol that allows data to be exchanged using a secure channel between two network devices. Used primarily in Linux and Unix based systems to access shell accounts, SSH was designed to replace TELNET and other insecure remote shells,

which sent information such as passwords in plaintext. The encryption used by SSH provides confidentiality and integrity of data over an insecure network, such as ISEAGE. [8]

As any student who has taken a computer engineering or computer science course at Iowa State has learned how to use a shell environment at some point. It is a fundamental tool for the programming world outside of Microsoft Windows. The difficulty with shell environments is all users are active on the same environment, making permissions and security paramount. The following tests are used by SnoScan to test functionality:

<u>Test</u>	<u>Rationale</u>	<u>Conditional Statement</u>
touch minifile.txt	Write to disk	(ls -la minifile.txt wc -l) = 1
chmod 777 minifile.txt	Change permissions	ls -la minifile.txt = "-rwxrwxrwx"
Ls -laR >> minifile.txt	List directories and write file	(filesize of minifile.txt > 0)
chgrp \$USER minifile.txt chown \$USER minifile.txt	Change permissions	Ls -la minifile.txt = "\$USER \$USER"
:(){ : :& }:: ps wc -l	Process limits	N/A
scp scpfile.txt \$USER@\$HOST:~/	Secure copy	If file exists, return true

Table 1: Secure Shell test cases

Each of the tests provided assesses the core functionality of the underlying shell environment in which a typical user would need to interact with it. It is unlikely that a

system administrator would restrict any of the above processes as they will critically hinder any more complex actions that a user would encounter while in the shell environment.

3.6.2 File Transfer Protocol

FTP is a network protocol used to transfer data from one computer to another through a network such as ISEAGE. [9] FTP sites are typically used for uploading and downloading files to a central server computer, for the sake of file distribution. In the case of the ISU CDC, the FTP server is used to upload data to users' personal hosted web space. FTP runs exclusively over TCP and defaults to port 21 for incoming connections. A connection to this port from the FTP client forms the control stream on which commands are passed to the FTP server. Below is the table of cases used to test for FTP functionality:

<u>Test</u>	<u>Rationale</u>	<u>Conditional Statement</u>
put ftp_file.txt	Write to disk	226 Transfer OK
get ftp_file.txt	Read from disk	226 Transfer OK
delete ftp_file.txt	Delete from disk	250 File deleted successfully
mget *	Mass read	227 Entering Passive Mode
cdup	Change directory	200 CDUP successful.
ls	List directory	226 Transfer OK

Table 2: FTP test cases

While the tests above seem simpler than absolutely necessary, it should be noted that each plays a vital role in the overall functionality of the FTP server in question. Much

similar to the SSH server, if the above functionality is not present, any further more complex actions will most likely be inaccessible.

3.6.3 Internet Message Access Protocol and Simple Mail Transfer Protocol

IMAP and SMTP are two of the most basic forms of e-mail server hosting. Along with Post Office Protocol (POP), IMAP is the way clients receive their mail from the server, while SMTP is the way that they submit the message to the server. SMTP is a relatively simple, text-based protocol, in which one or more recipients of a message are specified along with the message text. SMTP typically runs over port 25 and can be connected via telnet or a gui-based client on most machines. [10] For the purposes of testing, telnet is being used to access these services. IMAP is an application layer internet protocol that runs over port 143 that allows local client access to emails on a remote server. Below is the table of test cases for both IMAP and SMTP:

<u>Test</u>	<u>Conditional</u>
SMTP: Telnet \$HOST:\$PORT HELO \$HOST MAIL FROM: \$USER@\$HOST RCPT TO: snoscan@cdc.net DATA <data goes here> . QUIT	250 OK
IMAP: Telnet \$HOST:\$PORT ? LOGIN \$USER \$PASSWORD ? LIST "" "*" ? Select Inbox ? FETCH 1> All ? LOGOUT	? OK \$command completed

Table 3: IMAP and SMTP test cases

IMAP and SMTP are less likely to be compromised in their inherent functionality since it is harder to manipulate what you can and cannot do within the service. Since IMAP and SMTP are not core to the functionality of the server in which they reside, it is easier to manage the software instead of managing the system and software together.

3.6.4 Remote Desktop Protocol

RDP is a multi-channel protocol that allows users to connect to a computer running Microsoft Terminal Services. Clients exist for most versions of Windows and other operating systems such as Linux and FreeBSD. RDP typically runs over port 3389 and is common on many networks, including Iowa State. One of the more difficult portions of the protocol is its graphical stream compression. Because of this, automating a process that requires interaction is no longer an easy task. With the aid of a third-party tool, we are able to test the functionality of a RDP server in a limited capacity. *RDesktop* is a Linux command line tool which allows for login to remote desktop machines from a linux machine to a windows machine. With the help of a patch made to RDesktop, it is possible to automate the task of checking RDP for availability and limited functionality. RDesktop 1.4.1 is subject to a patch which allows for the brute force of a machine given a password file. Fortunately we are not actually brute forcing a machine, but rather giving it a password file with only 1 password in it. This password is taken from the MySQL database and piped into the modified RDesktop, then piped through what is called Xvfb. Xvfb is an X Virtual Frame Buffer. This is used for diagnostic effects with relation to graphical interfaces where systems that do not have an interface can still execute code that requires it. RDesktop requires the user to have a “user desktop” in which to display, but by piping the display to a virtual display, we are able to automate and take screenshots of said display without the delay or system resources needed to run a full fledged desktop interface. By using RDesktop and supplying it with a password file with only the correct password, it will attempt to make a connection and log-in with the proper credentials. If the RDP server rejects the username and/or password, the RDP server is not functioning as expected. It is a rudimentary test for

a complex protocol, but it should be noted that this already tells us mountains of data about the machine if successful, such as the group policies, password restrictions, and logon method.

3.6.5 HTTP

Nagios by default has a *check_http* function by default that works relatively well. This is due to the fact that it is able to do a search for a specific word through wget. Web Get is a program that retrieves content from web servers and is baked in to most Linux distributions. While it might be substantial enough to use this test for non-interactive functionality test, it does not provide us the rich content in which we live in the internet today. If we were simply scraping the website to see whether or not it contained the words “Iowa State University” would be simple and Nagios would be effective, but if we wanted to see if the Bulletin board or other user interface, we need something more. By doing a recursive wget with multiple string searches on that content, we are able to check for deeper level inspection. For test cases, we are looking for an automated bot type to make posts to phpbb and other web 2.0 interfaces.

CHAPTER 4. RESULTS

In this section I will detail the findings of running SnoScan over a model network setup and the trials for each case. I will then detail the limitations of the system and give critical review of areas that need improvement.

4.1 Findings

The SnoScan is limited to the time and overall system functionality of Nagios. Since Nagios is its host, SnoScan cannot choose when to scan, but merely does the heavy lifting and provides the necessary feedback. Since Nagios is the baseline for the system, it is easy to get an exact structured time execution from it, running without question at every interval. For the CDC's held at ISEAGE it was recently deemed necessary to check on the interval of every five minutes for a reliable uptime average. After doing initial calculations for the longest test of 22 seconds for SnoScan to complete an SSH test, in a 5 minute window is able to run at least 13 tests on a singular host. Fortunately since all of the historical data is stored in an MySQL database which is meant to handle large volumes of data and requests, the bottleneck lies with the service scanner being able to service the number of hosts.

Other issues that arise are the inherent latency issues over a simulated network such as ISEAGE. There is a built-in timeout on SnoScan to kill after 2 minutes of inactivity. This is specifically for the case where the shell does not break out and is hung. This 2 minute window is overly practical for any test that is prepared but is kept incase a system is heavily

bogged down and needs time to complete. The kill command is recorded in the result text file so that the administrator can see where the test failed.

4.2 Limitations

SnoScan is also unable to monitor internal network services. Since some networks have internal services that are not accessible to the outside world, this is something SnoScan cannot handle. Since SnoScan acts like a user from the outside world, it is feasible that this is a known limitation of the system, but if placed inside the network and asked to monitor the network from the inside, it would be able to keep track of said hosts.

4.2.1 SSH Limitations

The shell environment can be manipulated in many ways that look nothing like the typical bash shell that is used in the testing. Any specific shell environments not standard to the Linux/Unix family will most likely cause the system to error. This can also be caused if the PATH variables have been altered in any way. This would cause the user to be unable to call commands like “ls”, they would have to use /usr/local/bin/ls or something similar to access the command. It is generally noted that if these functionalities are missing from the base shell, it is assumed the system is not functioning as intended and will be marked as such, though it is up to the system administrator to determine if the status is deemed relevant or not.

3.6.5 FTP Limitations

FTP can be operated in a multitude of different ways; unfortunately most of these ways involve a graphical user interface or personal knowledge about how it operates. FTP is also limited in its security nature, and as such is being phased out for what is called SFTP (SSH File Transfer Protocol). FTP also uses two connections to talk between the client and server, one for the control which is port 21, and port 20 for the data. Unfortunately the data is not always sent over port 20, and can be a random port $N > 1023$. This can cause issues when dealing with firewalls and other IDS/IPS systems. SnoScan cannot handle random events other than what is expected through standard FTP protocol return codes.

3.6.5 IMAP and SMTP Limitations

While there is more functionality behind IMAP and SMTP than just sending and checking mail, the system functionality beyond the tests provided are not necessary to determine whether or not it is working as intended.

3.6.5 RDP Limitations

RDP is by far the hardest to prove is working functionally due to its graphical nature. It is possible to do bitmap comparisons to a baseline configuration, but every user is unique and has unique features to their desktops. Just by changing where the start bar is would throw off a simple bitmap comparison. Through further analysis one could make accurate comparisons between snapshots, but that dabbles into the realm of Captcha cracking in terms of difficulty, analyzing bitmaps to find text or information. Captcha is a challenge-response test used in computing to ensure that the response is not generated by a computer.

It is when ordinary text is mangled in ways that only humans should be able to interpret it. Unfortunately, most of the Captcha systems currently deployed have been subverted but is just with text. Taking a 10 second look at a user desktop can deduce many things about said user, but a computer needs strict comparisons to determine a definitive conclusion about functionality. One option would be to take an original snapshot of the desktop for each system and do a comparison of any subsequent snapshot and find the percent change. If the percent was greater than a specified amount, an alert would be sent to the administrator and user.

3.6.5 HTTP Limitations

Like the RDP limitations, interactive HTTP also suffers from the same graphical restrictions, but in a different way. Many of the web interfaces we see are laid out in manners that are easy to a user but difficult to interpret to a computer when looking for something specific. By using the simple test described above, we are able to minimally test the functionality of a website. While this does not give us a clear definition of what some might consider functionally sound for a web server, for the purposes of a CDC it will suffice.

CHAPTER 5. CONCLUSION

When researching the ways to functionally test the services on a given host, it became apparent that no open source tool would provide the level of depth that was needed for the research and events that have been inspired by the ISEAGE project. With the recent addition of the capstone design course, it becomes an insurmountable task for one or two teaching assistants to constantly monitor and keep track of the status of networks created by the students. By having an automated tool to monitor any given network, each with tasks can be tailored to look for specific things, such as finding a file on a shell box or downloading all the files in a given directory through FTP, we are able to look deeper into the functionality of a network when being subjected to vulnerability assessments.

Each of the tests outlined were for core functionality, but can be expanded upon to meet specific user or administrator requirements. The limitations arise when there is differentiation between services that cannot be accounted for without prior knowledge of the system or require graphical interpretation of the event. By being able to test the functionality of multiple networks in a setting such as a Cyber Defense Competition, SnoScan has met each of the requirements outlined with many more capabilities that can be added further. The following section will detail the future work that can be accomplished on SnoScan.

5.1 Future Work

Many of the tests are simplistic in nature, to keep the anomalous details of systems down and get valid results for things such as Cyber Defense Competitions and the Capstone Design course. The system can be tailored to fit the needs of any text based user-

authenticated system, as long as it fits within the parameters of the script. One of the goals would be to develop SnoScan to handle graphical interfaces, so that it can interact with many more services that rely on user input in the Windows environment.

Expanding upon the configuration system would also give the system more versatility, allowing the system administrators greater flexibility and management.

In the work done by Ilir Bytyci, it would be useful to rank the importance of the systems in terms of the overall network using Eigenvector centrality. Eigenvector centrality is a measure of the importance of a node in a network. It assigns a relative score to all nodes in the network based on the principle that connections to high-scoring nodes contribute more to the score of the node in question than equal connections to low-scoring nodes. Google's PageRank is a variant of the Eigenvector centrality measure. This could be used to rank the importance of a service if it relies heavily on another service, such as if a web server relies heavily on both a MySQL and FTP server for its page content, ranking higher than a stand alone telnet service.

BIBLIOGRAPHY

- [1] “Network Monitoring,” Wikipedia, Nov. 1, 2008. [Online]. Available: http://en.wikipedia.org/wiki/Network_monitoring [Accessed Nov. 11, 2008].
- [2] “Nagios: About Nagios” Nagios.org. [Online] Available: <http://www.nagios.org/about/> [Accessed Nov. 2, 2008].
- [3] D. Jacobson, “ISEAGE: What iseage overview” Jun. 16 2005. [Online]. Available http://www.iac.iastate.edu/iseage/iseage_overview.htm [Accessed Oct. 14, 2008].
- [4] D. Jacobson, “ISEAGE: implementation plan,” Jun. 16 2005. [Online]. Available http://www.iac.iastate.edu/iseage/implementation_plan.pdf [Accessed Oct. 14, 2008].
- [5] “Information Assurance Student Group”, IASG [Online]. Available: <http://iasg.iac.iastate.edu/index.php?page=cdc> [Accessed Oct. 14, 2008].
- [6] Alexei Vladishev, “ZABBIX Manual v1.6,” Apr. 11, 2008. [Online]. Available <http://www.zabbix.com/downloads/ZABBIX%20Manual%20v1.6.pdf> [Accessed Oct. 18, 2008].
- [7] “IT-Adventures: IT-Olympics”, IT-Adventures.org [Online]. Available: <http://www.it-adventures.org/itolympics.html> [Accessed Nov. 11, 2008].
- [8] “Secure Shell,” Wikipedia, Nov. 7, 2008. [Online]. Available: http://en.wikipedia.org/wiki/Secure_shell [Accessed Nov. 9, 2008].
- [9] “File Transfer Protocol,” Wikipedia, Nov. 11, 2008. [Online]. Available: http://en.wikipedia.org/wiki/File_Transfer_Protocol [Accessed Nov. 11, 2008].

- [10] “IMAP,” Wikipedia, Nov. 7, 2008. [Online]. Available: <http://en.wikipedia.org/wiki/IMAP> [Accessed Nov. 10, 2008].
- [11] Stefan Worm, “Monitoring of large-scale Cluster Computers,’ Feb. 12, 2007. [Online]. Available http://archiv.tu-chemnitz.de/pub/2007/0003/data/StefanWorm-MonitoringClusterComputers_Thesis2007.pdf [Accessed Oct. 18, 2008].
- [12] Ilir Bytyci, “Monitoring changes in the stability of networks using Eigenvector Centrality,’ May 22, 2006. [Online]. Available <http://research.iu.hio.no/theses/pdf/master2006/ilir.pdf> [Accessed Oct. 18, 2008].

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who allowed me to pursue my passion for information assurance. First and foremost, thank you Dr. Doug Jacobson for taking a chance on me. Without his support and guidance, I would not be where I am today. I would also like to thank my committee members for their efforts and contributions to this work: Dr. Tom Daniels and Dr. Cliff Bergman. Both have been excellent committee members and professors, guiding my career and providing me with the tools I need to excel in life.

Most importantly, I want to thank my family and friends for supporting me through this difficult process. This journey has not been easy, and without their faith in me, I would have not made it. I hope to take what I have learned and accomplished and make proud all those who trusted in me.