EVALUATING THE VIABILITY OF INTRUSION DETECTION SYSTEM BENCHMARKING

A Thesis in TCC 402

Presented to:

The Faculty of the School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science in Computer Engineering

By

Kenneth J. Pickering

Computer Engineering

On my honor as a University student, on this assignment I have neither given nor received unauthorized aid as defined by the Honor Guidelines for Papers in TCC Courses

____

APPROVED: Dr. Patricia Click, TCC Advisor__________________________________

APPROVED: Dr. David Evans, Tech Advisor__________________________________

PREFACE

I would like to thank Dr. David Evans, my Technical Advisor, for his support and advisory work during the course of this project, and Dr. Patricia Click for helping me keep this task managed properly, as well as the aid she provided as my TCC advisor with editing. I would also like to thank MIT's Lincoln Labs for providing the data sets and documentation necessary to produce this project. The CS Systems Department provided me with resources and lab space, for which I am very grateful. Lastly, I would like to thank the Snort development staff for producing a great, free product and the users on the mailing list for providing help when it was needed.

The main reason I undertook this endeavor was to improve the quality of intrusion detection systems, which can be a valuable tool in detecting malicious attacks that strike corporate networks. If one of these systems is utilized, it is imperative to maintain it and make sure it is able to detect the newest forms of exploits, as well as the older methods of attack. An efficient way to do this would be to devise an evaluation of intrusion detection systems, which is something that Lincoln Labs attempted to do. I believe this test and the methodology behind it are not up to the task, and it should not be as frequently used as it is. It was the goal of this project to invalidate the Lincoln Labs test and point out the flaws. In the conclusion, I put forth ideas that could be used to develop a better and more rigorous intrusion detection evaluation.

TABLE OF CONTENTS

TABLE OF FIGURES

ABSTRACT

This report evaluates the DARPA-LL intrusion detection system evaluation. Intrusion detection systems are not easily constructed or maintained due to the almost daily evolution of network traffic and known exploits. The most popular and rigorous system devised to date is the DARPA-LL 1998 and 1999 Intrusion Detection System Evaluation. I will evaluate it through analysis of the documentation published for the lab as well as experimentation using different rule customizations.

Snort was selected because of its price and easy customization. Through manipulation of its rules files, it was to be customized to perform better in certain situations using the DARPA-LL evaluation criteria. This shows that this benchmarking system can be easily manipulated. Developers looking to enhance performance can alter their rules files to better detect attacks. This system could be manipulated to produce better results, and thus becomes less a test of developers testing their true systems and more a test of how well developers can interpret the testing data.

This project shows that benchmarking intrusion detections systems cannot be done effectively at this time. Until we develop more advanced artificial intelligence and datamining techniques, it will be very hard to evaluated intrusion detection systems. The amount of customization that goes into effectively using one, as well as the ever-changing number of viable network exploits makes it impossible at this time.

INTRODUCTION

For my undergraduate thesis, I evaluated the Lincoln Labs' (LL) DARPA (Defense Advanced Research Projects Agency) intrusion detection system benchmarking data set and evaluation. This project found discrepancies by analyzing the data set between different years the evaluation was performed. Running the data through an intrusion detection system with a variety of configuration settings discovered these inconsistencies. The thesis also analyzes the actual way in which the test was taken and evaluated originally. This research will try to prove that the DARPA evaluation is not an acceptable way to measure the performance of an intrusion detection system, and may actually impede development of better systems due to evaluation based on a bad standard.

Scope and Method

Network security is a thriving industry in this country as more and more of the corporate workspace is converted to digital media. Because companies and home users keep sensitive information on their computers, there is a great need to protect that information from those who would exploit it. One way to help keep attackers at bay is by using an intrusion detection system (IDS), which are designed to locate and notify systems administrators to the presence of malicious traffic. The current systems are not effective right now because detecting intrusions and other forms of malicious traffic in a large, modern corporate network is difficult. Something must be done in order to improve performance and make these systems ready for reliable operation in a dynamic environment.

We can classify IDS's as host-based and network-based. Host-based intrusion detection systems monitor the computer the software is running on and often integrates closely with the operating system (Durst et al., 54). Network IDS "monitor network traffic between hosts. Unlike host-based systems, which detect malicious behavior outright, these systems deduce behavior based on the content and format of data packets on the network" (Durst et al., 55). This project looked exclusively at network-based intrusion detection systems, as opposed to host-based intrusion detection. My thesis used MIT's Lincoln Labs data (also known as DARPA-LL data), available at http://www.ll.mit.edu/. This data consists of two weeks of traffic captured using tcpdump, a well-known open-source packetsniffer, which can be replayed in a network environment. The documentation and procedures produced by Lincoln Labs are analyzed in Chapter 3.

Snort, the IDS that this project utilizes, can take tcpdump files as an input and scan the traffic for abnormalities. All of the malicious traffic introduced into the DARPA-LL test data set is known, so administrators know how well their system picks up the given exploits and also know when false positives are generated. Most of the academic IDS's researched, as well as a lot of commercial systems, use the DARPA data as a common benchmark. Two different years of the provided tcpdump traffic were used to determine whether Lincoln Labs' information and procedures can be used as a viable benchmark for something as complex as an intrusion detection system that is thrown into a much larger and more dynamic environment.

My main purpose for undertaking this study was to improve the overall quality intrusion detection system benchmarking through analyzing the current ways to test these systems. Based on my previous experience as a network systems administrator, I can attest to the shortcomings of the current commercial and open-source solutions. They generate many “false positives,” which is standard traffic being diagnosed as malicious data, and sometimes a few “false negatives,” which are attacks gone unnoticed by the IDS. Both of these lead to a viable system’s resources being wasted. The amount of false positives clogs up log files with erroneous reports, thus masking a legitimate attack in a sea of false alarms. Most systems administrators will ignore the IDS’s data due to this fact. The other problem stems from attacks appearing to be friendly, normal traffic, which is even more alarming, since an attacker would be able to creep into the network without an alert from the IDS.

By evaluating the current academic standard in benchmarking using Snort, it can be determined whether the benchmark is, in fact, a valid test to run. A positive performance on these tests can give IDS programmers a false sense of security, which could lead to a degeneration of future development. If the data MIT provides is not up to par, perhaps a newer, more rigorous form of testing can be used.

Overview

This report will consist of a few major sections. Chapter 2 reviews literature relevant to my project. This section delves into previous work in intrusion detection systems and suggestions made to improve the current systems. Past IDS evaluations and the problems found within DARPA-LL after analysis of their documentation will be discussed in Chapter 3. The set up of my project and decisions made to change my initial proposal is discussed in Chapter 4.

The next chapters delve into the actual experimentation of my project and the four runs of the DARPA-LL evaluation using the Snort IDS. Chapter 5 gives examples of output of Snort and SnortSnarf and discusses how the rules sets were developed. Chapter 6 discusses my analysis of the project, where I delve into the information collected and try to determine the validity of DARPA-LL's benchmark. Chapter 7 gives the results of the paper and presents a conclusion.

INTRUSION DETECTION

A reliable and efficient intrusion detection system (IDS) is a necessary component in any network. It can alert administrators of possible attackers and give a good view of the network's status. This section of the proposal looks at current systems, proposals for new types of IDSs, and higher level ideas that could be carried over into IDS development. Many of the academic and commercial systems available were tested using the DARPA-LL IDS test, so all of the systems presented could benefit from a viable benchmark. It is the main goal of this project to look at how the DARPA-LL tested systems perform in a real-world environment and, if the Lincoln Labs' test is determined to be a bad benchmark, propose new ways to test all forms of IDS.

Snort

The IDS looked at most closely in this project, Snort, is a rules-based network intrusion detection system (NIDS). Martin Roesch, in his paper entitled "Snort – Lightweight Intrusion Detection for Networks," says "Snort fills an important 'ecological niche' in the realm of network security: a cross-platform, lightweight network intrusion detection tool that can be deployed to monitor small TCP/IP networks and detect a wide variety of suspicious network traffic as well as outright attacks" (1). The SANS Institute also reported Snort as becoming the standard among intrusion detection experts due to the fact that it is open-source, frequently updated, and free of charge (2). Snort generates a number of false positives, which can number in the thousands per day on a network attached to the

Internet running a default installation of Snort (Hoagland, 376). Thankfully, many programs, like SnortSnarf, are available to help parse through large amounts of false alerts to access relevant data.

Improving Current Systems

Many IDS experts have proposed different ideas for improving the current systems in use. Sekar et al. propose that a new universal intrusion detection rules language be developed to make creating rules for different IDSs easier in their paper "A High-Performance Network Intrusion Detection System" (8). Also, Lee and Stolfo point out that building an IDS is a huge engineering task and imply that, in order to make production of rules easier, a debugger for rules languages should be developed to reduce the amount of effort involved in implementation (228-9). Barruffi, Milano, and Montanari think that automated responses should be added into current IDSs to block attacks without relying on the administrator and allow the system to manage intrusion recovery (74). Another possible improvement would be making systems fault-tolerant, so that a hacker cannot subvert the IDS itself. Shen et al. proposed "a hybrid of distributed, redundant, and cross-corroborating techniques" (425).

Others believe that a new system of communication protocols or a redesign of routing protocols should be developed to help combat many problems stemming from an inability to effectively trace attackers. Schnackenberg et al. proposed a new Cooperative Intrusion Traceback and Response Architecture (CITRA) across IDSs, routers, firewalls, and other network appliances that would

"(1) trace intrusions across network boundaries, (2) prevent or mitigate subsequent damage from intrusions, (3) consolidate and report intrusion activities and (4) coordinate responses on a system-wide basis" (56). A denial of service attack could be performed on routers, either by a malformed router or malicious attacker. Cheung and Levitt say smarter routers must be developed to detect and ignore bad or compromised routers (94).

Making IDS More Intelligent

Many academic programmers are using new techniques to make IDSs more intelligent. Fawcett and Provost at Bell Atlantic Science and Technology theorized a high-level approach to intrusion detection in their article about activity monitoring. They believe that the same theories used in detecting cellular telephone fraud, which rely on user profiling, can be used in a computer network environment (59-60). Statistical Process Control, developed by Arizona State University and used in their system ISA-IDS, uses Chi-square techniques to detect anomalies in a network environment as well as a rules-based system, which they call "Clustering" (Ye, Emran, Li, and Chen , 3, 10). In another paper by Ye and a different group of professors entitled "Probabilistic Techniques for Intrusion Detection Based on Computer Audit Data," Ye et al. propose other probabilistic techniques "including Hotelling's T^2 Test, chi-square multivariate test, and Markov chain," and use these methods with the same data set to gauge efficiency (Ye, Li, Emran, and Xu 266). Johns Hopkins University attempted to set up an IDS composed of Neural Networks that can function as an anomaly detector (Lee and Heinbuch, 1171). The system they proposed was host-based, protecting a network server.

Many experts in the network security and intrusion detection field have proposed viable solutions to the problems with network security. All of the above solutions, especially the ones that used the DARPA-LL IDS test data, could benefit from a better testing schema for use in their development cycle.

EVALUATING IDS

Lincoln Labs' intrusion detection evaluation was not the first effort at testing IDS systems, but was the first attempt at an all-conclusive test of whole categories of standard network exploits and other forms of malicious traffic. In Lippmann et al.'s paper on the 1999 DARPA evaluation, they discuss the previous endeavors. Before the DARPA-LL test, the most advanced benchmarking system previously tried involved simple telnet and FTP traffic with automated attacks (Lippmann et al, 2000, 2). Along the same lines, there was also a product comparison of 10 commercial IDS products in 1999 done by G. Shipley (Lippmann et al, 2000, 2).

1998 Evaluation: Background Information

The 1998 evaluation consisted of seven weeks of test learning data. The purpose behind producing this was to give IDS evaluators a chance to tweak their rules based and anomaly detection systems by familiarizing them with the typical traffic running through the network. There were also attacks thrown into each of the learning data files, to show typical attacks. It gave the systems using datamining and learning algorithms a chance to have sample data, which helped them "learn" how the network operated (Lippmann et al, 1999, 2).

The MIT lab used a test bed to generate all its background data for the 1998 evaluation. Since it was difficult to take Air Force network data and manage to remove sensitive information from it for evaluation purposes, the lab used custom software to generate traffic. It allowed Lincoln Labs to simulate the activities of hundreds of programmers, managers and secretaries, as well as make a few hosts appear to be thousands of terminals. A packetsniffer was located on the internal network to capture the generated traffic. All simulated attacks were launched from "outside" the base, so a traffic sniffer located at the gateway would be able to catch it all (Lippmann et al, 1999, page 3).

Intrusion detection systems were supposed to detect the following categories: denial of service (DoS), port scanning and probes, user to root attacks (U2R), and remote to local (R2L). In the DoS categories, which should be fairly easy to detect, the best system could only pick up 65% of attacks. The probes category had two of the systems detecting 90% of probing activities. In the U2R category, the best systems could only find 60% to 70% of attacks. In what is probably the most serious of attacks, the R2L, which allows remote users to gain local access (in some cases root access), the best system could only find 35% of attacks. The systems that could interpret BSM audit data on Sun workstations could improve performance slightly (Lippmann et al, 1999, 9-12).

Problems With 1998 Evaluation

There were several problems with the 1998 evaluatio n, some of which Lincoln Labs' acknowledged in its write-up of the 1999 evaluation. Lincoln Labs cited that the 1998 evaluation was only to provide exploits against UNIX hosts and was only supposed to initially be used for IDS that had been developed using DARPA grants (Lippmann et al, 2000, 4). The oversight of Windows NTwhen it was arguably the most popular business operating system at the time of the evaluation, was probably used by government and military personal as well. Leaving this particular operating system out really harms the 1998 evaluation’s credibility.

One of the major drawbacks in running the evaluation is that a listing of attacks in the actual test data is not available. I instead had to use two of the normal “learning data” weeks. In week 6 of the testing data, a bad router added too many ICMP packets into the data set, as computers constantly “pinged” each other. This generated massive log files and a major slowdown in Snort performance, as it logged over 1.5 million alerts upon completion, in some instances.

The two biggest problems, though, were the methodology used to come up with the exploits, and the way they were executed. There were 38 different kinds of malicious traffic used, but they were executed in no real logical order. Some sort of attacker intelligence should have been placed into the attack routines, even in the preliminary data. Merely adding malicious traffic is not sufficient. Attacks and exploits are sent for a purpose and usually come in a set order. Similar to real-life crime, there is always some amount of reconnaissance before an attack takes place. Even relatively unsophisticated attacks like DoS are usually performed for a purpose and are probably somewhat calculated.

Lincoln Lab used a scoring method that weighed amount of attacks detected versus the amount of false positives found, using receiver operating characteristic (ROC), a technique originally used in signal detection (Lippmann et al, 1999, 2). It should also have taken the amount of resources used by the system, ease-of-use, and what type of system it is.

System resources and ease-of-use are huge factors in using any sort of system. Snort, for instance, can run on a variety of computers, depending on the amount of traffic it has to handle. For home and small networks, it is possible to get away using a low-grade Intel Pentium or even Intel i486 processor. If a system can only run on a super-computer cluster, it will severely affect how feasible the system is. Also, if a system is extremely complex or not well documented, it could affect how easily users can manipulate it.

1999 Evaluation: General Information and Problems

The 1999 evaluation set out to improve upon the evaluation performed a year earlier, with extensions added on and more attack types. This included the addition of Windows NT exploits. The test bed to generate this particular data was similar to the 1998 tool, but also included updated statistics and the addition of Windows NT hosts. Also, the same attack sub-categories were used and are listed above (Lippmann et al, 2000, 7-10).

The full results of this evaluation can be seen in the summary of the results that Lincoln Labs published on pages 14 - 18 of its summary. The scoring method Lincoln Labs showed in their charts was the percentage of attacks found with below 10 false alarms of that attack instance per day, and also had a detection rate above 40%.

This evaluation had many improvements over its predecessor, although it was still far from perfect. The scoring mechanism is similar to the one used in the 1998 evaluation and is, once again, a problem. I found the listings of the individual IDS performance to be confusing and the evaluation still did not use the criteria discussed in the section on the 1998 evaluation. A less confusing and more honest representation of the data would be to just list the amount of attacks found versus the amount of false positives. Capping it at a certain number and detection percentage leaves out the systems that did not “make the grade.” The full listing is easier to represent and makes more sense (Lippmann et al, 2000, 14-18).

General Problems with DARPA-LL

DARPA meets its goal of being the most advanced IDS benchmarking system to date, but is still lacking in some areas. The major flaws in the system itself deal with the test bed, and how the researchers decided to evaluate the systems.

The test bed generates a series of generic packets based on statistical properties and uses a minimum of hosts. Even though the tool is not publicly available and the statistical properties used to develop it are unknown, it can not be as accurate as capturing real-time traffic in a corporate or government network. This helps avoid security implications of publicizing government network traffic, but comes with an inherent cost. It is extremely difficult to replicate people’s actions within a computer program, although from analyzing the alerts, MIT did a reasonably good job. From an accuracy point of view, it would have been much better to use real traffic.

System type factors heavily into how well certain attacks are detected. For instance, local-only attacks, where a user abuses software on his own computer, could not be detected by a network-based IDS, unless it somehow uses an Ethernet adapter (like a “land” attack, which the user issues packets to himself). On the other hand, host-based systems will miss attacks against routers and other computers in most cases. Since the two systems vary heavily in operation and purpose, it would make sense to evaluate them separately.

Although this evaluation was quite advanced, it could have been more inclusive, or at least more open in its methods, after the evaluation was complete. Not knowing how all of the background traffic was generated is a serious drawback to the system. Also, the scoring methodology and separate evaluations for the different types of IDSs could also have helped to evaluate systems after the data was collected.

EXPERIMENTATION SET-UP

Initial Set-Up

The initial set-up was, with three computers connected with a hub (Figure 1). This process was as not easy as anticipated. The main problems that occurred stemmed from lack of time and resources. In order to do what was originally specified, I would have had to edit NetPoke so that I could funnel all the data to one host-based system, unlike the original Lincoln Labs closed configuration, which consisted of several interconnected computers. Also, a separated routing infrastructure would have to be set up in order to get the Linux and Windows computers to be able to recognize each other. This was too much to ask of the CS Systems staff, who were kind enough to provide me with a lab space, computers, and assistance.

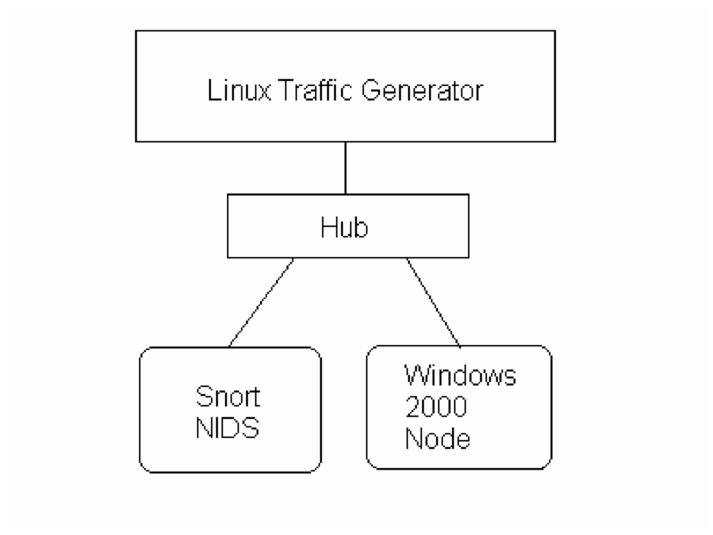


Figure 1 – Initial Set-Up

Resulting Decisions

The host-based system was dropped and Snort was exclusively used, with two different versions of this IDS ran against the data set. Snort 1.7, which was released over a year ago, is timed much closer to the actual date the data was captured, since the evaluations were created in 1998 and 1999. Snort 1.8.3 is also used, as it was the most recent stable release of the software package at the time this thesis was performed. Different rules sets are to be plugged into each IDS, giving each different functionality, even though they use similar engines.

Secondary Set-Up

One computer, with the DARPA data sets stored locally and Snort running in a Red Hat Linux 7.1 environment, was used. Snort has the ability to parse the tcpdump logs itself, so no outside utility (NetPoke) was utilized. This computer needed a lot of disk space in order to adequately hold the results of several data sets. I ran both of the applications twice: once with full settings, where any sort of abnormality is noticed, and once with a lightweight, customized rules set that only looks for the specific attacks that I want Snort to recognize.

The sole computer used in my experiment had a reasonably fast AMD Athalon processor (1.2 GHz) and 768 MB of RAM. An additional 70 GB hard disk was added to the machine to hold the massive alert files Snort was generating, and the data parsed with SnortSnarf. At one point, the video card in this computer ceased to function, causing a delay of about 3 days while the computer was fixed.

Initially, the CS systems staff installed the 70 GB disk formatted in FAT32 format. This caused errors with Snort 1.8.3 (but not Snort 1.7), as some of the automatic filenames it generates are Linux Ext2 specific. Because of the massive amount of data already run through using Snort 1.7, it was impossible to salvage it before formatting the disk, resulting in a time loss of about a week's worth of work.

Configuring Snort

Snort is an open-source project started by Martin Roesch and is available on a variety of platforms for download, including Windows, Linux, and Solaris. It has been around for a few years and acquired a decent number of developers and users. From personal use, it seems to be quite easy to customize and very flexible in terms of possible uses.

Snort has a main configuration file that allows you to add and remove pre-processor requirements as well as the rules files included. This is where you typically specify the limit of fragmentation you want to take notice of and if you want the packets reconstructed or not. Below is a selection of the configuration file that allows you to select how the tcp stream is reassembled. The comments in the file are very descriptive (comments are preceded with the '#' symbol) and there are a number of options defined, such as what network ports to watch and which side (client, server or both) of the connection.

```
# tcp stream reassembly directive
# no arguments loads the default configuration
#   Only reassemble the client,
#   Only reassemble the default list of ports (See below),
#   Give alerts for "bad" streams
#
# Available options (comma delimited):
#   clientonly - reassemble traffic for the client side of a
      connection only
#   serveronly - reassemble traffic for the server side of a
      connection only
#   both - reassemble both sides of a session
#   noalerts - turn off alerts from the stream reassembly stage
      of stream4
# ports [list] - use the space separated list of ports in
      [list], "all"
#      will turn on reassembly for all ports, "default" will turn
#      on reassembly for ports 21, 23, 25, 53, 80, 143, 110, 111
#      and 513

preprocessor stream4_reassemble
```

Another important selection of the configuration file is the rules specification, where you can add and remove rules libraries, in a very similar fashion to a typical programming language. The custom rules file includes I used for Snort version 1.8.3 is seen here:

```
#==========================================
# Include all relevant rulesets here
#
# shellcode, policy, info, backdoor, and virus rulesets are
# disabled by default.  These require tuning and maintance.
# Please read the included specific file for more information.
#==========================================

include bad-traffic.rules
include exploit.rules
include scan.rules
include finger.rules
include ftp.rules
include telnet.rules
include smtp.rules
include rpc.rules
include rservices.rules
include dos.rules
include ddos.rules
include dns.rules
include tftp.rules
include web-cgi.rules
include web-coldfusion.rules
include web-frontpage.rules
include web-iis.rules
include web-misc.rules
include web-attacks.rules
include sql.rules
include x11.rules
include icmp.rules
include netbios.rules
include misc.rules
include attack-responses.rules
# include backdoor.rules
# include shellcode.rules
# include policy.rules
# include porn.rules
# include info.rules
# include icmp-info.rules
# include virus.rules
# include experimental.rules
include local.rules
```

There is also a specific language to specify the rules themselves. This is where Snort's true flexibility lies. You can basically choose to filter any kind of network traffic you want. Full documentation of rules specification language is available in the Snort documentation (http://www.snort.org). A few things to notice in a typical "rule" are the type of protocol, the destinations to be watched, the sources to be watched, the label of the attack, the contents of the packet to be searched for, and a reference describing the attack, if applicable. A small selection of the "backdoor" rules file is shown below:

```
# (C) Copyright 2001, Martin Roesch, Brian Caswell, et al.  All
rights reserved.
# $Id: backdoor.rules,v 1.16 2001/12/19 18:40:04 cazz Exp $
#---------------
# BACKDOOR RULES
#---------------
#

alert tcp $EXTERNAL_NET 27374 -> $HOME_NET any (msg:"BACKDOOR
      subseven 22"; flags: A+; content:
      "|0d0a5b52504c5d3030320d0a|"; reference:arachnids,485;
      sid:103;  classtype:misc-activity; rev:3;)
alert tcp $EXTERNAL_NET 1024: -> $HOME_NET 2589 (msg:"BACKDOOR -
      Dagger_1.4.0_client_connect"; flags: A+; content: "|0b 00
      00 00 07 00 00 00|Connect"; depth: 16;
      reference:arachnids,483; sid:104;  classtype:misc-activity;
      rev:3;)
```

```
alert tcp $HOME_NET 2589 -> $EXTERNAL_NET 1024: (msg:"BACKDOOR -
      Dagger_1.4.0"; flags: A+; content:
      "|3200000006000000|Drives|2400|"; depth: 16;
      reference:arachnids,484; sid:105;  classtype:misc-activity;
      rev:3;)
alert tcp $EXTERNAL_NET 80 -> $HOME_NET 1054 (msg:"BACKDOOR
      ACKcmdC trojan scan"; seq: 101058054; ack: 101058054;
      flags: A;reference:arachnids,445; sid:106;  classtype:misc-
      activity; rev:3;)
alert tcp $EXTERNAL_NET 16959 -> $HOME_NET any (msg:"BACKDOOR
      subseven DEFCON8 2.1 access"; content: "PWD";
      content:"acidphreak"; nocase; flags: A+; sid:107;
      classtype:misc-activity; rev:4;)
alert tcp $EXTERNAL_NET any -> $HOME_NET 7597 (msg:"BACKDOOR QAZ
      Worm Client Login access"; flags: A+; content:"|71 61 7a 77
      73 78 2e 68 73 71|"; reference:MCAFEE,98775; sid:108;
      classtype:misc-activity; rev:3;)
alert tcp $HOME_NET 12345 -> $EXTERNAL_NET any (msg:"BACKDOOR
      netbus active"; flags: A+; content: "NetBus";
      reference:arachnids,401; sid:109;  classtype:misc-activity;
      rev:3;)
```

The particular type of Snort installation used in this particular experiment just places the alerts of positive attacks in an “alerts” text file. Other versions of this IDS allow you to place the data in a MySQL or ACID data base, which were not used in this project. There are several open-source solutions to parsing the data from the text file into a useable output. SnortSnarf by James Hoagland is a perl script that takes the text file and sorts it by alert type and IP address into a slick HTML interface. Snort and SnortSnarf will be discussed in further detail in the next chapter.

EXPERIMENTATION AND CREATION OF RULE SETS

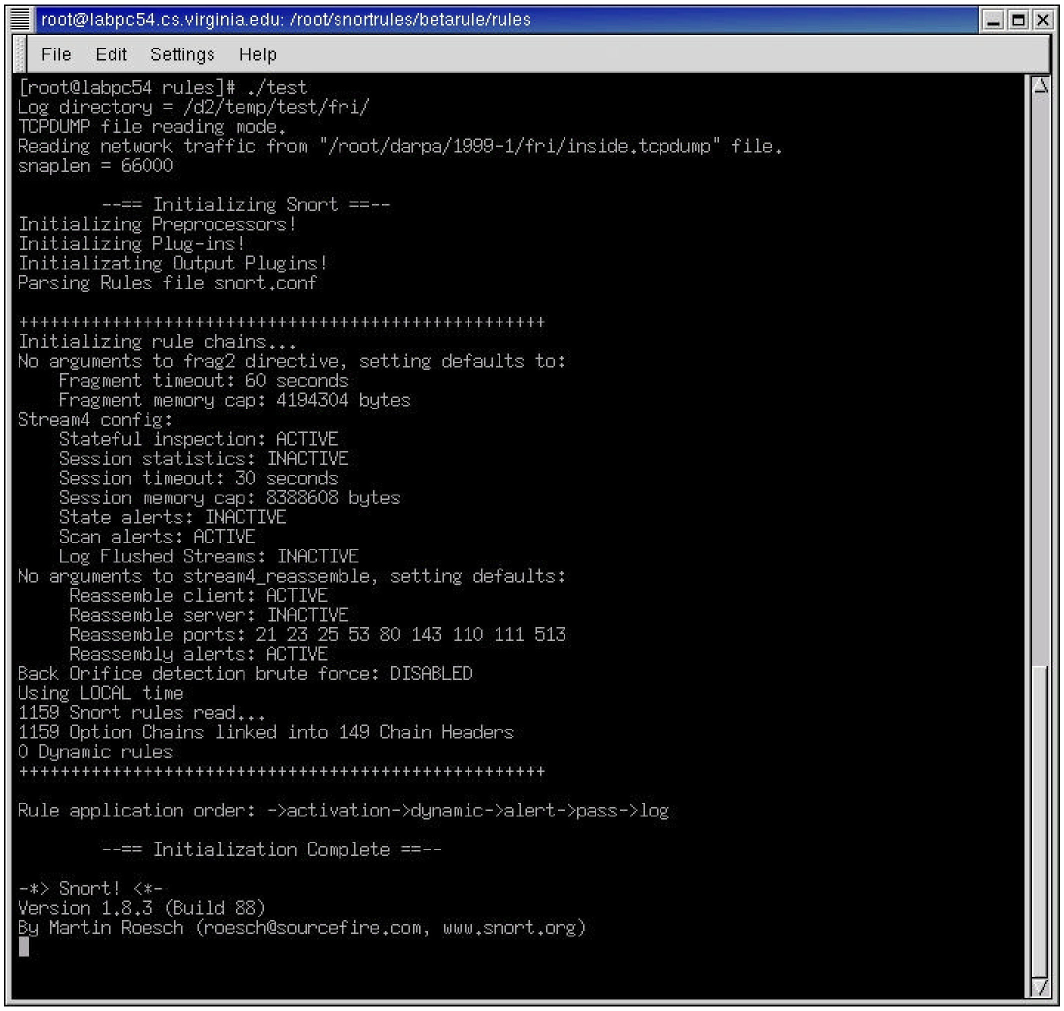
The data set used in this experiment was selected from the listings at Lincoln Labs' web site, and has just recently become publicly available (where before, they were available only through correspondence with the lab). I chose to use the Week 6 and Week 7 preliminary test data from 1998 and the actual two weeks of evaluation data from 1999. The incident logs, which are listings of the attacks and abnormalities, were not available for the actual 1998 test data.

MIT provides an open-source tool, called NetPoke, that can replay tcpdump files on a network at a user specified rate. I noticed that on a 10 Mbit network, a large number of packets were being dropped by the traffic generation machine, requiring me to run the logs in "real-time," which would have taken me approximately four months to complete the all the log files. Since I made the changes listed in the previous chapter, it was much quicker and more efficient to run the data through the Snort IDS, itself. Snort will not drop any packets and can run most of the days in a couple hours.

Snort Configuration

The preliminary configuration of Snort was reasonably straightforward, although there were a few problems with the rpm packages, which are essentially Red Hat's solution to easy installation. It instead had to be compiled manually, using the source code on the web site. A few additional libraries had to be installed in order to get the IDS to function properly, as Snort requires packet capture drivers to operate.

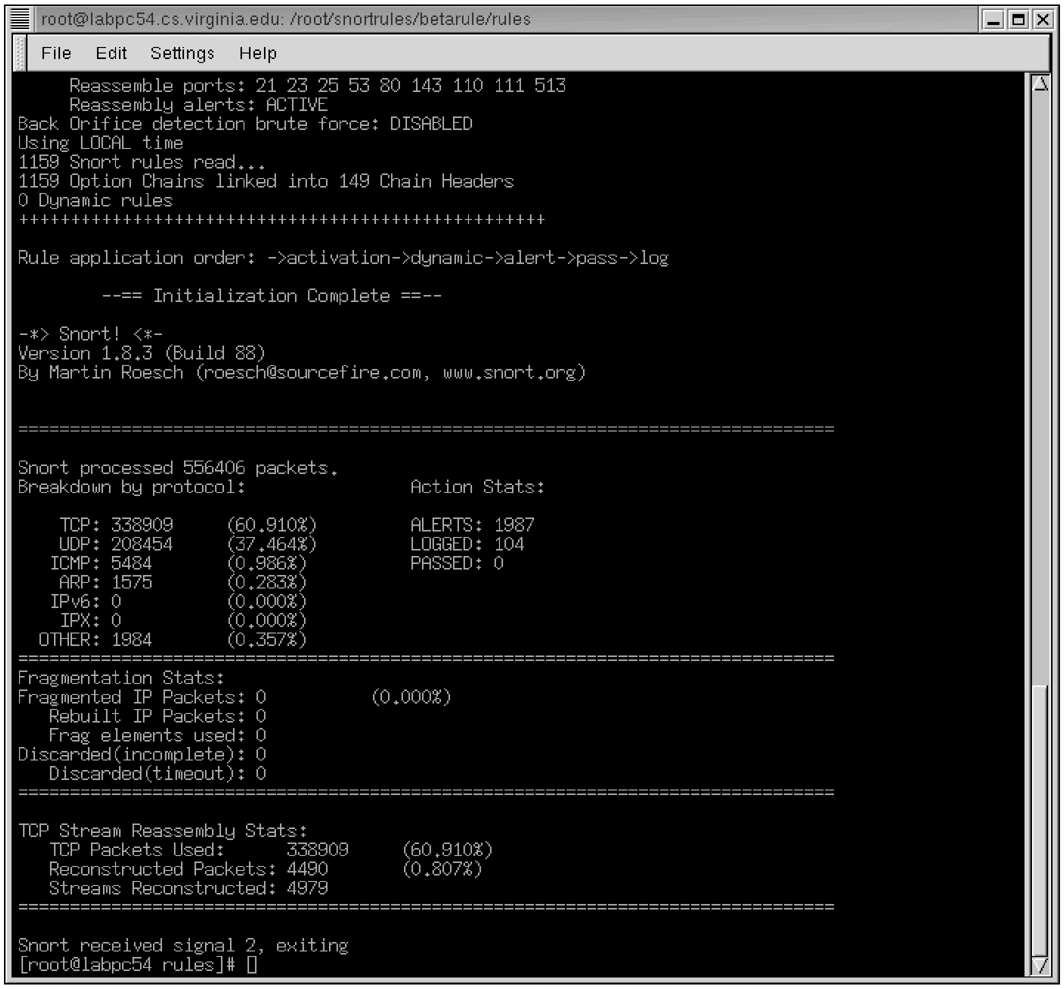
After the installation, Snort was activated in real-time detection mode to make sure it was configured properly. After a few minutes of intercepting live traffic and examining the logs, it was then time to use offline DARPA-LL data. I selected a few of the data files and proceeded to run them through Snort. The typical output from the start-up of the IDS can be seen below (Fig 2).


root@labpc54.cs.virginia.edu: /root/snortrules/betarule/rules
File Edit Settings Help

```
[root@labpc54 rules]# ./test
Log directory = /d2/temp/test/fri/
TCPDUMP file reading mode.
Reading network traffic from "/root/darpa/1999-1/fri/inside.tcpdump" file.
snaplen = 66000

        --== Initializing Snort ==--
Initializing Preprocessors!
Initializing Plug-ins!
Initializating Output Plugins!
Parsing Rules file snort.conf

+++++++++++++++++++++++++++++++++++++++++++++++++++
Initializing rule chains...
No arguments to frag2 directive, setting defaults to:
    Fragment timeout: 60 seconds
    Fragment memory cap: 4194304 bytes
Stream4 config:
    Stateful inspection: ACTIVE
    Session statistics: INACTIVE
    Session timeout: 30 seconds
    Session memory cap: 8388608 bytes
    State alerts: INACTIVE
    Scan alerts: ACTIVE
    Log Flushed Streams: INACTIVE
No arguments to stream4_reassemble, setting defaults:
     Reassemble client: ACTIVE
     Reassemble server: INACTIVE
     Reassemble ports: 21 23 25 53 80 143 110 111 513
     Reassembly alerts: ACTIVE
Back Orifice detection brute force: DISABLED
Using LOCAL time
1159 Snort rules read...
1159 Option Chains linked into 149 Chain Headers
0 Dynamic rules
+++++++++++++++++++++++++++++++++++++++++++++++++++

Rule application order: ->activation->dynamic->alert->pass->log

        --== Initialization Complete ==--

-*> Snort! <*-
Version 1.8.3 (Build 88)
By Martin Roesch (roesch@sourcefire.com, www.snort.org)
```

Figure 2 – Snort Pre-Run Screen

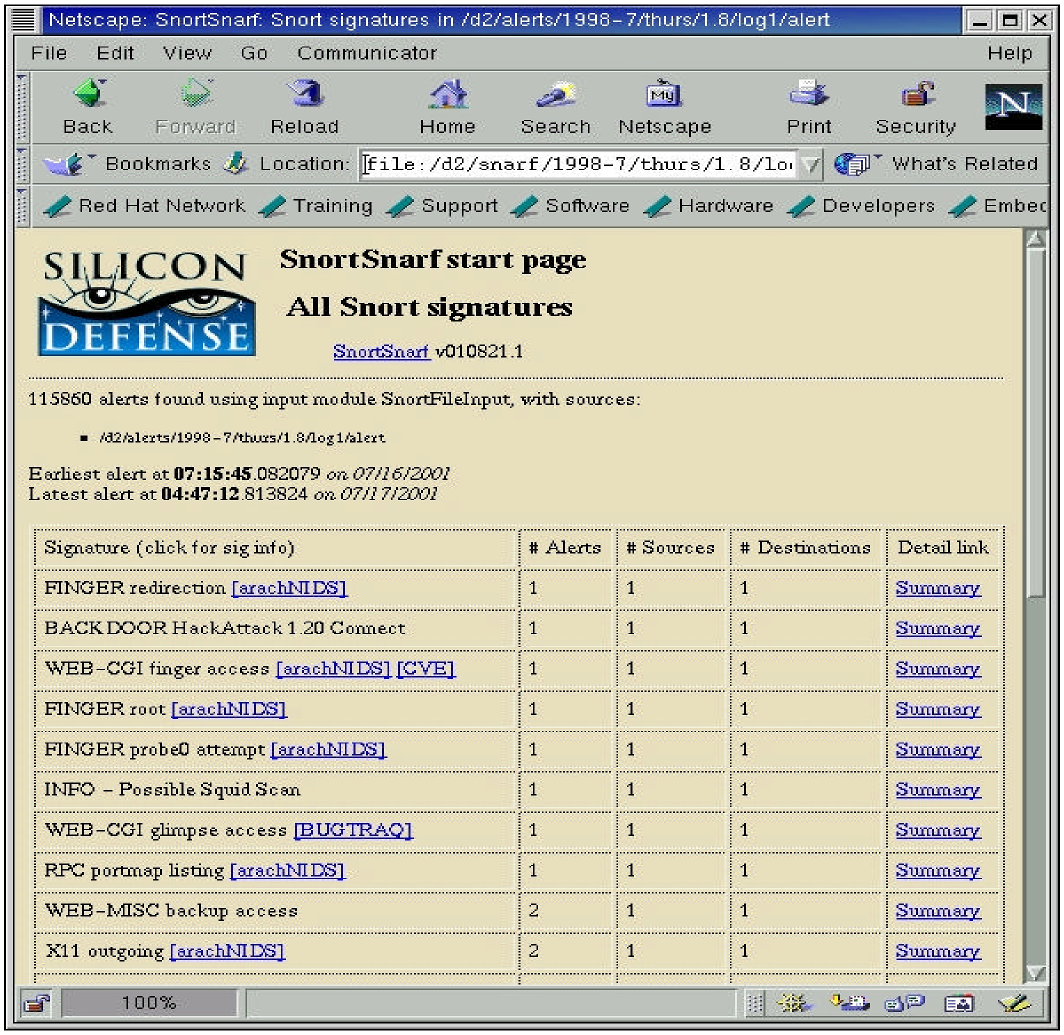
I looked at the alerts that had been generated and noticed their large size. Anywhere from zero to a million alerts had been generated, depending on the log file inputted. The task of going through these text files manually was going to be too time and labor intensive. Snort lists the number of packets scanned and alerts found when it has completed running, as can be seen in Fig. 3. Thankfully, SnortSnarf, can take these massive files and put them into a easy-to-read HTML format.


root@labpc54.cs.virginia.edu: /root/snortrules/betarule/rules
File Edit Settings Help

```
     Reassemble ports: 21 23 25 53 80 143 110 111 513
     Reassembly alerts: ACTIVE
Back Orifice detection brute force: DISABLED
Using LOCAL time
1159 Snort rules read...
1159 Option Chains linked into 149 Chain Headers
0 Dynamic rules
+++++++++++++++++++++++++++++++++++++++++++++++++++

Rule application order: ->activation->dynamic->alert->pass->log

        --== Initialization Complete ==--

-*> Snort! <*-
Version 1.8.3 (Build 88)
By Martin Roesch (roesch@sourcefire.com, www.snort.org)


===============================================================================

Snort processed 556406 packets.
Breakdown by protocol:                Action Stats:

    TCP: 338909     (60.910%)         ALERTS: 1987
    UDP: 208454     (37.464%)         LOGGED: 104
   ICMP: 5484       (0.986%)          PASSED: 0
    ARP: 1575       (0.283%)
   IPv6: 0          (0.000%)
    IPX: 0          (0.000%)
  OTHER: 1984       (0.357%)
===============================================================================
Fragmentation Stats:
Fragmented IP Packets: 0          (0.000%)
   Rebuilt IP Packets: 0
   Frag elements used: 0
Discarded(incomplete): 0
   Discarded(timeout): 0
===============================================================================

TCP Stream Reassembly Stats:
   TCP Packets Used:      338909     (60.910%)
   Reconstructed Packets: 4490       (0.807%)
   Streams Reconstructed: 4979
===============================================================================

Snort received signal 2, exiting
[root@labpc54 rules]# []
```

Figure 3 – Snort Post-Run Screen

I took the practice data I had been working with, and parsed the resultant text files into HTML using SnortSnarf. After some rudimentary configuration, the data formed easy to read HTML. This was a much better method to view the data, because it links all the data and sorts it by alert type and IP, plugging in fairly seamlessly with Snort's classifications and logging methodology. A screen shot of the typical HTML output can be seen below, in Fig 4.


Netscape: SnortSnarf: Snort signatures in /d2/alerts/1998-7/thurs/1.8/log1/alert
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security
Bookmarks Location: file:/d2/snarf/1998-7/thurs/1.8/lo What's Related
Red Hat Network Training Support Software Hardware Developers Embec
SILICON DEFENSE
100%

SnortSnarf start page

All Snort signatures

SnortSnarf v010821.1

115860 alerts found using input module SnortFileInput, with sources:

- /d2/alerts/1998-7/thurs/1.8/log1/alert

Earliest alert at **07:15:45**.082079 *on 07/16/2001*
Latest alert at **04:47:12**.813824 *on 07/17/2001*

Signature (click for sig info)	# Alerts	# Sources	# Destinations	Detail link
FINGER redirection [arachNIDS]	1	1	1	Summary
BACKDOOR HackAttack 1.20 Connect	1	1	1	Summary
WEB-CGI finger access [arachNIDS] [CVE]	1	1	1	Summary
FINGER root [arachNIDS]	1	1	1	Summary
FINGER probe0 attempt [arachNIDS]	1	1	1	Summary
INFO - Possible Squid Scan	1	1	1	Summary
WEB-CGI glimpse access [BUGTRAQ]	1	1	1	Summary
RPC portmap listing [arachNIDS]	1	1	1	Summary
WEB-MISC backup access	2	1	1	Summary
X11 outgoing [arachNIDS]	2	1	1	Summary

Figure 4 – SnortSnarf Output

Snort Configuration Methodology

The configuration files for each particular run of the IDS were selected to show the variance that rules manipulation can cause to the IDS system. Based on listings of attacks in the 1998 test data and 1999 learning data, it was possible to predict the exploits used in the actual offline test given in 1999. Each particular run of the separate rules files is discussed in the next chapter, in my analysis section.

The exploits in 1998 Week 6 and Week 7 learning data, which the systems were run against, as well at the listings of the 1999 Week 2 learning data (used for reference) are listed in Appendix D. This data shows a very close correlation to actual test data used in 1999, which can be seen in Appendix G. The Snort 1.8.3 engine was run with two different configurations, one with a complete set of rules minus icmp-info rules. This particular rule set was left out because it generated too many false positives in the Snort 1.7 run. The Snort 1.8.3 custom configuration had a few rule sets removed, which, with the exception of a few attacks that it missed, had far fewer false positives. This would give it a much better performance rating on the Lincoln Labs' test.

The configuration files for each of the Snort builds are included in the Appendix section of this paper. A full listings of the libraries used is available on http://www.snort.org and are freely available for download.

IDS PERFORMANCE

Four types of system configurations were used in this project. The first IDS used was Snort 1.7 with the default rules set fully enabled. The second run opted to ignore the ICMP informational rules, while including everything else. The configuration files are included in the Appendix C. The second version I utilized was the latest stable build of Snort, 1.8.3. I used the a full rules set for this particular version as well as a customized version, created after viewing the 1998 and 1999 test data. The full rules set that Snort used listed all the rules except for icmp-info, which generated too many false positives under Snort 1.7. The second configuration file included rules that were purposely chosen to run better on the evaluation data because they had a decrease in the amount of false positives over the fuller rules set.

All of the results for the four different runs of the evaluation are included in Appendix E of this thesis paper, as well as the total number of alerts for each instance, which are found in Appendix F. The glossary of the attack abbreviations used for both of the evaluation years can be found in Appendix G.

Snort 1.7 Custom

The rules used in this particular instance of the build were derived from the rules bundle that came with this initially. The icmp-info rules file was left out, but all the others were left in.

This particular build had almost no false positives, but also picked up a minority of the attacks. Basically, the only malicious traffic this IDS picked up was Denial of Service (DoS) attacks that were comprised of malformed packets in some way, like the Ping of Death (PoD) or teardrop attacks. It did not detect portscans or remote-to-local attacks, so performed abysmal in these particular categories.

However, the minumum of false positives on the DoS attacks it picked up would give it a pretty favorable rating in that category.

Snort 1.7 Full

The rules file for this particular build was the full set of rules from the Snort 1.7 bundle that was used. It logged a large portion of ICMP traffic as a result.

This particular configuration was very good at detecting some Denial of Service (DoS) attacks as well as scanning attempts (port and IP scans). It performed much better than customized rules set (Snort 1.7 Custom, see above). It generated an enormous amount of false positives, however, since it was tracking ICMP traffic. In order to pick up portscans, ipsweeps, notice FTP probes, and pick up vulnerability scanners, you need to capture some of this traffic.

This particular build did much worse at detecting Remote-to-Local attacks than the later versions and rulesets.

Snort 1.8.3 Full

The rules configuration for this particular run was the majority of rules listed in the current CVS, except for rules which would generate a large number of false positives, with little to no gain in performance.

This particular build performed the best, as far as strictly detecting attacks. It performed admirably at noticing a variety of remote-to-local, DoS and reconnaissance attacks, although had a reasonably high false positive rate (second to the Snort 1.7 Full configuration). The improvement over the 1.7 runs is readily visible.

Snort 1.8.3 Custom

This build was the one designed to prove my thesis, as it was customized to perform well on the benchmark. I added the rules sets that would notice the most attacks with a minimum of false positives.

The performance of this system was the best, according to DARPA-LL standards. It detected almost the same amount of attacks as the Snort 1.8.3 Full configuration, with a reduction in false positive rates. It was possible, looking at past traffic and types of attacks, to predict which rule sets would be the best at performing on this particular benchmark.

Results

The results of this particular section led to invalidating the DARPA-LL benchmark. Since it was possible to use a variety of rules to attain a varying level of performance, this particular test was easily manipulated. A true benchmark should be resistant to manipulation in this fashion. Based on 1998 and 1999 test data, a superior performing rules configuration of Snort was easily found. The developers of the systems to be tested should not be given any inclination of the vulnerabilities to be present or the exact weighting of the scores. A blind test is really the only good option in testing these systems, otherwise you run the risk of testing an organization's ability at forecasting what the test will be, rather than testing the system itself.

CHAPTER 7: CONCLUSION

Summary

Looking at the documentation of the Lincoln Labs' Intrusion Detection System Evaluation, it was easy to find several flaws in its system. The major problems that need to be addressed are the test bed software and the actual evaluation criteria. Since the test bed traffic generation software is not publicly available, it is not possible to determine how accurate the background traffic inserted into the evaluation is. Also, the evaluation criteria does not account for system resources used, ease of use, or even what type of system it is (Host, Network, or a combination of the two).

The experimentation of my project revealed how easy it was to edit Snort's rule sets based on attack listings in the 1998 evaluation and 1999 test data. It was possible to drastically reduce the number of false positives in the 1999 test data while still being able to detect most, if not all, attacks that the full rule set could detect.

Interpretation

This paper points out the flaws of the DARPA-LL Intrusion Detection System Evaluation in order to be able to improve upon them and develop a better method of testing, if, indeed, one could be found. It is my opinion, based on my investigation of this benchmark, that it falls short of its intended goals, although it is the best system devised to date. The improvements listed in the previous chapters are things that can be taken into consideration when improving upon this system, or building a new one.

To build an all-inclusive intrusion detection benchmarking system would be a monumental task, but not necessarily an unreasonable one. Several factors would need to be considered in order to do this properly. The Bugtraq mailing list (http://www.securityfocus.com), reports a number of new system vulnerabilities every day. The system would have to evolve as network applications and systems evolve. Also, since these systems are used to protect a variety of networks, all with different needs and configurations, a system based on each networks average traffic should be used, with the attacks listings inserted.

Basically, the "ideal" system, or even a really useful system, is a somewhat unreasonable goal, until we get more advanced datamining and artificial intelligence capabilities at our disposal. The system Lincoln Labs designed was a passable, although somewhat easy to manipulate, evaluation when it first came out. However, a large amount of users have downloaded the data since then to evaluate their own systems, for whatever reason (Lippman et al, 2000, 6). This should be cautioned against for the reasons listed in this paper. Evaluating your system with an invalid benchmark can be worse than not having an evaluation at all, especially if it leads to placing a weak system on the market. If evaluators are making corrections to their systems based on information found in the preliminary data, as was very easy to do with Snort, then it can give them a much better score on the test. This test can give the developers a false sense of security, thinking a potentially invalid system is in proper working order. This is not to suggest that testing pre- and post-production systems is useless during development and verification, but that coming up with a common system of evaluating different IDS serves no real use unless the "ideal" testing system is created.

There is adequate evidence presented in this paper to suggest that the Lincoln Labs' DARPA-funded evaluation is not up to speed. Unless certain changes are made, benchmarking, testing and evaluating these extremely complex systems is not useful unless serious breakthroughs in machine intelligence are made.

Recommendations for Further Research

Designing an easily updated, intelligent evaluation system for IDS would be quite useful, but data sets would vary from company to company, as the usual traffic of most corporate networks can vary greatly. Developing a test similar to the Lincoln Labs evaluation would not be very useful. Further advances in datamining and artificial intelligence could help devise better evaluations.

Also, this paper does not mean to suggest that testing individual systems is too complicated. Anyone who uses an intrusion detection system should constantly test and configure it for the network they are trying to protect. Another topic of further research would be to establish a set of guidelines that systems administrators could use to devise their own tests. The main problem with DARPA-LL is that it tries to make one concrete test for a variety of different systems, and systems to be run on a variety of networks. This, as has been shown, is not a very easy, or even really feasible, thing to do.

APPENDIX A: WORKS CITED

Barruffi, Rosy, Michela Milano, and Rebecca Montanari. “Planning for Security Management”. IEEE Intelligent Systems & Their Applications. Los Alamitos, CA: IEEE Computer Society, Publications Office.

Cheung, Steven and Karl Levitt. Protecting Routing Infrastructures from Denial of Service Using Cooperative Intrusion Detection. 1997 New Security Paradigms Workshop. Langdale, Cumbria, UK, 23-26 September 1997. New York: ACM Press, 1999.

Durst, Robert et al. “Testing and Evaluating Computer Intrusion Detection Systems”. Communications of the ACM Volume 42 Issue 7 (July 1999): 53-61

Hoagland, James A., and Stuart Staniford. Viewing IDS alerts: Lessons from SnortSnarf. Proceedings of 2001 DARPA Information Survivability Conference and Exposition (DISCEX 2001). Anaheim, CA, 12-14 June 2001. New York: Institute of Electrical and Electronics Engineers, 2001.

Fawcett, Tom and Foster Provost. Activity Monitoring: Noticing interesting changes in behavior. Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD-99). San Diego, CA, 1999. New York: ACM Press, 1999.

Lee, Susan C., and David V. Heinbuch. Building a True Anomaly Detector for Intrusion Detection. Proceedings of MILCOM 2000, 21st Century Military Communications Conference. Los Angeles, CA, 22-25 Oct. 2000. New York: Institute of Electrical and Electronics Engineers, 2001.

Lee, Wenke and Salvatore J. Stolfo. “A Framework for Constructing Features and Models for Intrusion Detection Systems”. ACM Transactions on Information and System Security (TISSEC) Volume 3 Issue 4 (November 2000): 227-261.

Lippmann, Richard et al. Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation. Proceedings of DARPA Information Survivability Conference & Exposition (DISCEX), Hilton Head, South Carolina, 25-27 January 2000. Los Alamitos, CA: IEEE Computer Society, 1999: Vol. 2, 12-26.

Lippmann, Richard et al. “The 1999 DARPA Off-line Intrusion Detection Evaluation”. http://www.ll.mit.edu/IST/ideval/pubs/2000/1999Eval-ComputerNetworks2000.pdf, 2000.

Nash, David A., and Ragsdale, Daniel J. Simulation of self-similarity in network utilization patterns as a precursor to automated testing of intrusion detection systems. Proceedings of the 1st Annual IEEE Systems, Man, and Cybernetics Information Assurance Workshop, West Point, NY, June 6-7, 2000, pp. 53-57

Roesch, Martin. “Snort – Lightweight Intrusion Detection for Networks”. http://www.snort.org/docs/lisapaper.txt. 1999.

SANS Institute. “101 Security Solutions”. http://www.101com.com/solutions/security/article.asp?articleid=569. 2001.

Schnakenberg, Dan et al. Cooperative Intrusion Traceback and Response Architecture. Proceedings of 2001 DARPA Information Survivability Conference and Exposition (DISCEX 2001). Anaheim, CA, 12-14 June 2001. New York: Institute of Electrical and Electronics Engineers, 2001.

Sekar, R. et al. A High-Performance Network Intrusion Detection System. Proceedings of the 6th ACM conference on Computer and Communications Security. Singapore, November 1999. New York: ACM Press, 1999.

Shen, Y. Peggy et al. Attack Tolerant Enhancement of Intrusion Detection Systems. Proceedings of MILCOM 2000, 21st Century Military Communications Conference. Los Angeles, CA, 22-25 Oct. 2000. New York: Institute of Electrical and Electronics Engineers, 2001.

Ye, Nong, Syed Masum Emran, Xiangyang Li, and Qiang Chen. Statistical Control for Computer Intrusion Detection. Proceedings of 2001 DARPA Information Survivability Conference and Exposition (DISCEX 2001). Anaheim, CA, 12-14 June 2001. New York: Institute of Electrical and Electronics Engineers, 2001.

Ye, Nong, Xiangyang Li, Syed Masum Emran, and Mingming Xu. “Probabilistic Techniques for Intrusion Detection Based on Computer Audit Data”. IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans Volume 31 Issue 4 (July 2001): 266-274.

APPENDIX B: BIBLIOGRAPHY

Barruffi, Rosy, Michela Milano, and Rebecca Montanari. "Planning for Security Management". IEEE Intelligent Systems & Their Applications. Los Alamitos, CA: IEEE Computer Society, Publications Office.

Cheung, Steven and Karl Levitt. Protecting Routing Infrastructures from Denial of Service Using Cooperative Intrusion Detection. 1997 New Security Paradigms Workshop. Langdale, Cumbria, UK, 23-26 September 1997. New York: ACM Press, 1999.

Cole, Eric. Hackers Beware: Defending Your Network From the Wiley Hacker. Indianapolis, Indiana: New Riders Publishing, 2002.

Durst, Robert et al. "Testing and Evaluating Computer Intrusion Detection Systems". Communications of the ACM Volume 42 Issue 7 (July 1999): 53-61

Hoagland, James A., and Stuart Staniford. Viewing IDS alerts: Lessons from SnortSnarf. Proceedings of 2001 DARPA Information Survivability Conference and Exposition (DISCEX 2001). Anaheim, CA, 12-14 June 2001. New York: Institute of Electrical and Electronics Engineers, 2001.

Fawcett, Tom and Foster Provost. Activity Monitoring: Noticing interesting changes in behavior. Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD-99). San Diego, CA, 1999. New York: ACM Press, 1999.

Lee, Susan C., and David V. Heinbuch. Building a True Anomaly Detector for Intrusion Detection. Proceedings of MILCOM 2000, 21st Century Military Communications Conference. Los Angeles, CA, 22-25 Oct. 2000. New York: Institute of Electrical and Electronics Engineers, 2001.

Lee, Wenke and Salvatore J. Stolfo. "A Framework for Constructing Features and Models for Intrusion Detection Systems". ACM Transactions on Information and System Security (TISSEC) Volume 3 Issue 4 (November 2000): 227-261.

Lippmann, Richard et al. Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation. Proceedings of DARPA Information Survivability Conference & Exposition (DISCEX), Hilton Head, South Carolina, 25-27 January 2000. Los Alamitos, CA: IEEE Computer Society, 1999: Vol. 2, 12-26.

Lippmann, Richard et al. "The 1999 DARPA Off-line Intrusion Detection Evaluation". http://www.ll.mit.edu/IST/ideval/pubs/2000/1999Eval-ComputerNetworks2000.pdf, 2000.

Nash, David A., and Ragsdale, Daniel J. Simulation of self-similarity in network utilization patterns as a precursor to automated testing of intrusion detection systems. Proceedings of the 1st Annual IEEE Systems, Man, and Cybernetics Information Assurance Workshop, West Point, NY, June 6-7, 2000, pp. 53-57

Northcutt, Steven, and Judy Novak. Network Intrusion Detection: An Analysts Handbook. Indianapolis, IN: New Riders Publishing, 2001.

Roesch, Martin. "Snort – Lightweight Intrusion Detection for Networks". http://www.snort.org/docs/lisapaper.txt. 1999.

SANS Institute. "101 Security Solutions". http://www.101com.com/solutions/security/article.asp?articleid=569. 2001.

Scambray, Joel, Stuart McClure, and George Kurtz. Hacking Exposed: Network Security Secrets and Solutions Second Edition. Berkeley, CA: Osborne/McGraw-Hill, U.S. National Security Agency. NSA Guide Windows 2000 Security Recommendation Guide, 2001. Available at http://nsa1.www.conxion.com/win2k/index.html 2001.

Schnakenberg, Dan et al. Cooperative Intrusion Traceback and Response Architecture. Proceedings of 2001 DARPA Information Survivability Conference and Exposition (DISCEX 2001). Anaheim, CA, 12-14 June 2001. New York: Institute of Electrical and Electronics Engineers, 2001.

Sekar, R. et al. A High-Performance Network Intrusion Detection System. Proceedings of the 6th ACM conference on Computer and Communications Security. Singapore, November 1999. New York: ACM Press, 1999.

Shen, Y. Peggy et al. Attack Tolerant Enhancement of Intrusion Detection Systems. Proceedings of MILCOM 2000, 21st Century Military Communications Conference. Los Angeles, CA, 22-25 Oct. 2000. New York: Institute of Electrical and Electronics Engineers, 2001.

Ye, Nong, Syed Masum Emran, Xiangyang Li, and Qiang Chen. Statistical Control for Computer Intrusion Detection. Proceedings of 2001 DARPA Information Survivability Conference and Exposition (DISCEX 2001). Anaheim, CA, 12-14 June 2001. New York: Institute of Electrical and Electronics Engineers, 2001.

Ye, Nong, Xiangyang Li, Syed Masum Emran, and Mingming Xu. “Probabilistic Techniques for Intrusion Detection Based on Computer Audit Data”. IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans Volume 31 Issue 4 (July 2001): 266-274.

APPENDIX C: SNORT CONFIGURATION FILES

This section contains the configuration files that were used for each build of Snort that I used. To learn more about the role of configuration files, and the rules that Snort uses, please visit http://www.snort.org.

<u>Snort 1.7 Full Configuration</u>

```
#--------------------------------------------------
#   http://www.snort.org     Snort 1.7.0 Ruleset
#         Ruleset Release -- 05/21/2001
#  Contact:  Jim Forster - jforster@rapidnet.com
#--------------------------------------------------
# NOTE:This ruleset only works for 1.7.0 and later
#--------------------------------------------------
include local.rules
include exploit.rules
include scan.rules
include finger.rules
include ftp.rules
include telnet.rules
include smtp.rules
include rpc.rules
include rservices.rules
include backdoor.rules
include dos.rules
include ddos.rules
include dns.rules
include netbios.rules
include sql.rules
include web-cgi.rules
include web-coldfusion.rules
include web-frontpage.rules
include web-misc.rules
include web-iis.rules
include icmp.rules
include misc.rules
include policy.rules
include info.rules
include virus.rules
```

Snort 1.7 Custom

```
#--------------------------------------------------
#   http://www.snort.org     Snort 1.7.0 Ruleset
#         Ruleset Release -- 05/21/2001
#  Contact:  Jim Forster - jforster@rapidnet.com
#--------------------------------------------------
# NOTE:This ruleset only works for 1.7.0 and later
#--------------------------------------------------
include local.rules
include exploit.rules
include scan.rules
include finger.rules
include ftp.rules
include telnet.rules
include smtp.rules
include rpc.rules
include rservices.rules
include backdoor.rules
include dos.rules
include ddos.rules
include dns.rules
include netbios.rules
include sql.rules
include web-cgi.rules
include web-coldfusion.rules
include web-frontpage.rules
include web-misc.rules
include web-iis.rules
include icmp.rules
include misc.rules
include policy.rules
# include info.rules
include virus.rules
```

Snort 1.8.3 Full Rule Set

```
#--------------------------------------------------
#   http://www.snort.org     Snort 1.8.1 Ruleset
#     Contact: snort-sigs@lists.sourceforge.net
#--------------------------------------------------
# NOTE:This ruleset only works for 1.8.0 and later
#--------------------------------------------------
# $Id: snort.conf,v 1.79 2002/01/02 16:12:54 cazz Exp $
####################################################
include bad-traffic.rules
include exploit.rules
include scan.rules
include finger.rules
include ftp.rules
include telnet.rules
include smtp.rules
include rpc.rules
include rservices.rules
include dos.rules
include ddos.rules
include dns.rules
include tftp.rules
include web-cgi.rules
include web-coldfusion.rules
include web-frontpage.rules
include web-iis.rules
include web-misc.rules
include web-attacks.rules
include sql.rules
include x11.rules
include icmp.rules
include netbios.rules
include misc.rules
include attack-responses.rules
include backdoor.rules
include shellcode.rules
include policy.rules
include porn.rules
include info.rules
# include icmp-info.rules
include virus.rules
include experimental.rules
include local.rules
```

Snort 1.8.3 Custom Rules:

```
#--------------------------------------------------
#   http://www.snort.org     Snort 1.8.1 Ruleset
#     Contact: snort-sigs@lists.sourceforge.net
#--------------------------------------------------
# NOTE:This ruleset only works for 1.8.0 and later
#--------------------------------------------------
# $Id: snort.conf,v 1.79 2002/01/02 16:12:54 cazz Exp $
####################################################
include bad-traffic.rules
include exploit.rules
include scan.rules
include finger.rules
include ftp.rules
include telnet.rules
include smtp.rules
include rpc.rules
include rservices.rules
include dos.rules
include ddos.rules
include dns.rules
include tftp.rules
include web-cgi.rules
include web-coldfusion.rules
include web-frontpage.rules
include web-iis.rules
include web-misc.rules
include web-attacks.rules
include sql.rules
include x11.rules
include icmp.rules
include netbios.rules
include misc.rules
include attack-responses.rules
# include backdoor.rules
# include shellcode.rules
# include policy.rules
# include porn.rules
# include info.rules
# include icmp-info.rules
# include virus.rules
# include experimental.rules
include local.rules
```

APPENDIX D: ATTACK DESCRIPTIONS

This section contains explanations for all the different abbreviations and definitions of the attacks used in the DARPA-LL trials, as well as the actual results from my evaluation of the data sets.

1998 Attack Listings

Data from 1998 evaluation website at http://www.ll.mit.edu/IST/ideval/

back	Denial of service attack against apache webserver where a client requests a URL containing many backslashes.
dict	Guess passwords for a valid user using simple variants of the account name over a telnet connection.
eject	Buffer overflow using eject program on Solaris. Leads to a user to root transition if successful.
ffb	Buffer overflow using the ffbconfig UNIX system command leads to root shell
format	Buffer overflow using the fdformat UNIX system command leads to root shell
ftp-write	Remote FTP user creates .rhost file in world writable anonymous FTP directory and obtains local login.
guest	Try to guess password via telnet for guest account.
imap	Remote buffer overflow using imap port leads to root shell
ipsweep	Surveillance sweep performing either a port sweep or ping on multiple host addresses.
land	Denial of service where a remote host is sent a UDP packet with the same source and destination
loadmodule	Non-stealthy loadmodule attack which resets IFS for a normal user and creates a root shell
multihop	Multi-day scenario in which a user first breaks into one machine
neptune	Syn flood denial of service on one or more ports.
nmap	Network mapping using the nmap tool. Mode of exploring network will vary--options include SYN

perlmagic	Perl attack which sets the user id to root in a perl script and creates a root shell
phf	Exploitable CGI script which allows a client to execute arbitrary commands on a machine with a misconfigured web server.
pod	Denial of service: ping of death
portsweep	Surveillance sweep through many ports to determine which services are supported on a single host.
rootkit	Multi-day scenario where a user installs one or more components of a rootkit
satan	Network probing tool which looks for well-known weaknesses. Operates at three different levels. Level 0 is light
smurf	Denial of service icmp echo reply flood.
spy	Multi-day scenario in which a user breaks into a machine with the purpose of finding important information where the user tries to avoid detection. Uses several different exploit methods to gain access.
syslog	Denial of service for the syslog service connects to port 514 with unresolvable source ip.
teardrop	Denial of service where mis-fragmented UDP packets cause some systems to reboot.
warez	User logs into anonymous FTP site and creates a hidden directory.
warezclient	Users downloading illegal software which was previously posted via anonymous FTP by the warezmaster.
warezmaster	Anonymous FTP upload of Warez (usually illegal copies of copyrighted software) onto FTP server.

1999 Learning Data Exploits: Week 2

ID	Date	Start_Time	Score	Name
1	03/08/1999	08:01:01	1	NTinfoscan
2	03/08/1999	08:50:15	1	pod
3	03/08/1999	09:39:16	1	back
4	03/08/1999	12:09:18	1	httptunnel
5	03/08/1999	15:57:15	1	land
6	03/08/1999	17:27:13	1	secret
7	03/08/1999	19:09:17	1	ps attack
8	03/09/1999	08:44:17	1	portsweep
9	03/09/1999	09:43:51	1	eject
10	03/09/1999	10:06:43	1	back
11	03/09/1999	10:54:19	1	loadmodule
12	03/09/1999	11:49:13	1	secret
13	03/09/1999	14:25:16	1	mailbomb
14	03/09/1999	13:05:10	1	ipsweep
15	03/09/1999	16:11:15	1	phf
16	03/09/1999	18:06:17	1	httptunnel
17	03/10/1999	12:02:13	1	satan
18	03/10/1999	13:44:18	1	mailbomb
19	03/10/1999	15:25:18	1	perl (Failed)
20	03/10/1999	20:17:10	1	ipsweep

21	03/10/1999	23:23:00	1	eject (console)
22	03/10/1999	23:56:14	1	crashiis
23	03/11/1999	08:04:17	1	crashiis
24	03/11/1999	09:33:17	1	satan
25	03/11/1999	10:50:11	1	portsweep
26	03/11/1999	11:04:16	1	neptune
27	03/11/1999	12:57:13	1	secret
28	03/11/1999	14:25:17	1	perl
29	03/11/1999	15:47:15	1	land
30	03/11/1999	16:36:10	1	ipsweep
31	03/11/1999	19:16:18	1	ftp-write
32	03/12/1999	08:07:17	1	phf
33	03/12/1999	08:10:40	1	perl (console)
34	03/12/1999	08:16:46	1	ps (console)
35	03/12/1999	09:18:15	1	pod
36	03/12/1999	11:20:15	1	neptune
37	03/12/1999	12:40:12	1	crashiis
38	03/12/1999	13:12:17	1	loadmodule
39	03/12/1999	14:06:17	1	perl (Failed)
40	03/12/1999	14:24:18	1	ps
41	03/12/1999	15:24:16	1	eject
42	03/12/1999	17:13:10	1	portsweep

43	03/12/1999	17:43:18	1	ftp-write

Attack Descriptions - 1999

back	Denial of service attack against apache webserver where a client requests a URL containing many backslashes.
crashiis	A single, malformed http request causes the webserver to crash.
dict	Guess passwords for a valid user using simple variants of the account name over a telnet connection.
eject	Buffer overflow using eject program on Solaris. Leads to a user->root transition if successful.
ffb	Buffer overflow using the ffbconfig UNIX system command leads to root shell
format	Buffer overflow using the fdformat UNIX system command leads to root shell
ftp-write	Remote FTP user creates .rhost file in world writable anonymous FTP directory and obtains local login.
guest	Try to guess password via telnet for guest account.
httptunnel	There are two phases to this attack: Setup - a web "client" is setup on the machine being attacked, which is configured, perhaps via crontab, to periodically make requests of a "ser ver" running on a non-privilaeged port on the attacking machine. Action - When the periodic requests are recieved, the server encapsulates commands to be run by the "client" in a cookie.. things like "cat /etc/passwd".. etc..
imap	Remote buffer overflow using imap port leads to root shell
ipsweep	Surveillance sweep performing either a port sweep or ping on multiple host addresses.
land	Denial of service where a remote host is sent a UDP packet with the same source and destination
loadmodule	Non-stealthy loadmodule attack which resets IFS for a normal user and creates a root shell
mailbomb	A Denial of Service attack where we send the mailserver many large messages for delivery in order to slow it down, perhaps effectively halting normal operation.
multihop	Multi-day scenario in which a user first breaks into one machine
neptune	Syn flood denial of service on one or more ports.
nmap	Network mapping using the nmap tool. Mode of exploring network will vary--options include SYN
ntinfoscan	A process by which the attacker scans an NT machine for information concerning its configuration, including ftp services, telnet services, web services, system account information, file systems and permissions.
perlmagic	Perl attack which sets the user id to root in a perl script and creates a root shell

phf	Exploitable CGI script which allows a client to execute arbitrary commands on a machine with a misconfigured web server.
pod	Denial of service ping of death
portsweep	Surveillance sweep through many ports to determine which services are supported on a single host.
ps	Ps takes advantage of a racecondition in the ps command in Sol. 2.5, allowing a user to gain root access.
rootkit	Multi-day scenario where a user installs one or more components of a rootkit
satan	Network probing tool which looks for well-known weaknesses. Operates at three different levels. Level 0 is light
smurf	Denial of service icmp echo reply flood.
spy	Multi-day scenario in which a user breaks into a machine with the purpose of finding important information where the user tries to avoid detection. Uses several different exploit methods to gain access.
syslog	Denial of service for the syslog service connects to port 514 with unresolvable source ip.
teardrop	Denial of service where mis-fragmented UDP packets cause some systems to reboot.
warez	User logs into anonymous FTP site and creates a hidden directory.
warezclient	Users downloading illegal software which was previously posted via anonymous FTP by the warezmaster.
warezmaster	Anonymous FTP upload of Warez (usually illegal copies of copywrited software) onto FTP server.

APPENDIX E: IDS EVALUATION RESULTS

1998 Learning Data Week 6

Week	Day	Attack Name	Snort 1.7 Full	Snort 1.7 Custom	Snort 1.8 Full	Snort 1.8 Custom
6	Mon	phf	N	N	Y	Y
6	Mon	satan	N	N	Y	Y
6	Mon	neptune	N	N	N	N
6	Tues	portsweep	N	N	N	N
6	Tues	pod	M	M	Y	Y
6	Tues	land	N	N	Y	Y
6	Wed	ipsweep	Y	N	Y	Y
6	Wed	neptune	N	N	N	N
6	Wed	back	N	N	Y	Y
6	Thurs	ipsweep	**Y/M**	N	NA*	NA*
6	Thurs	ipsweep	**Y/M**	N	NA*	NA*
6	Thurs	eject	N	N	NA*	NA*
6	Thurs	ffb	N	N	NA*	NA*
6	Thurs	eject	N	N	NA*	NA*
6	Thurs	eject	N	N	NA*	NA*
6	Thurs	eject	N	N	NA*	NA*
6	Thurs	pod	Y	Y	NA*	NA*

6	Thurs	pod	Y	Y	NA*	NA*
6	Thurs	pod	Y	Y	NA*	NA*
6	Thurs	dict	N	N	NA*	NA*
6	Thurs	ipsweep	**Y/M**	N	NA*	NA*
6	Thurs	phf	N	N	NA*	NA*
6	Thurs	neptune	N	N	NA*	NA*
6	Thurs	portsweep	**Y/M**	N	NA*	NA*
6	Thurs	eject	N	N	NA*	NA*
6	Thurs	portsweep	**Y/M**	N	NA*	NA*
6	Thurs	smurf	Y	N	NA*	NA*
6	Thurs	land	N	N	NA*	NA*
6	Thurs	neptune	N	N	NA*	NA*
6	Thurs	teardrop	Y	Y	NA*	NA*
6	Thurs	satan	**Y/M**	N	NA*	NA*
6	Thurs	ipsweep	**Y/M**	N	NA*	NA*
6	Thurs	eject	N	N	NA*	NA*
6	Thurs	portsweep	**Y/M**	N	NA*	NA*
6	Thurs	ffb	N	N	NA*	NA*
6	Thurs	ipsweep	**Y/M**	N	NA*	NA*
6	Thurs	land	N	N	NA*	NA*
6	Thurs	teardrop	Y	Y	NA*	NA*

6	Thurs	pod	Y	Y	NA*	NA*
6	Thurs	pod	Y	Y	NA*	NA*
6	Thurs	perlmagic	N	N	NA*	NA*
6	Thurs	satan	N	N	NA*	NA*
6	Thurs	perlmagic	N	N	NA*	NA*
6	Thurs	eject	N	N	NA*	NA*
6	Thurs	smurf	Y	Y	NA*	NA*
6	Thurs	eject	N	N	NA*	NA*
6	Thurs	ffb	N	N	NA*	NA*
6	Thurs	eject	N	N	NA*	NA*
6	Thurs	eject	N	N	NA*	NA*
6	Thurs	eject	N	N	NA*	NA*
6	Fri	teardrop	N	N	NA*	NA*
6	Fri	neptune	N	N	NA*	NA*
6	Fri	smurf	Y	Y	NA*	NA*

Y/M : Attack may have been noticed based on Snort output, but was not intuitively obvious

NA* : There were too many alerts generated by Snort 1.8.3 for SnortSnarf to be able to parse them with the computer I had on these two days

1998 Learning Data Week 7

Week	Day	Attack Name	Snort 1.7 Full	Snort 1.7 Custom	Snort 1.8 Full	Snort 1.8 Custom
7	Mon	satan	Y/M	N	N	N
7	Mon	syslog	N	N	N	N
7	Mon	phf	N	N	Y	Y
7	Mon	land	Partial	N	Y	Y
7	Tues	portsweep	N	N	Y/M	Y/M
7	Tues	pod	Y	Y	Y	Y
7	Tues	ffb	N	N	N	N
7	Tues	eject	N	N	N	N
7	Wed	phf	N	N	Y	Y
7	Wed	loadmodule	N	N	N	N
7	Wed	teardrop	Y	Y	Y	Y
7	Wed	ipsweep	Y	N	Y/M	Y
7	Wed	portsweep	N	N	N	N
7	Thurs	smurf	Y	N	Y	Y
7	Thurs	satan	Y/M	N	Y	Y
7	Thurs	perlmagic	N	N	N	N
7	Thurs	ipsweep	Y	N	Y/M	Y
7	Fri	neptune	N	N	Y/M	Y/M
7	Fri	smurf	N	N	N	N
7	Fri	neptune	N	N	Y/M	Y/M
7	Fri	back	N	N	Y	Y

1999 Test Data Week 1

Week	Day	Attack Name	Snort 1.7 F	Snot 1.7 C	Snort 1.8 F	Snort 1.8 C
1	Mon	ps	N	N	N	N
1	Mon	sendmail	N	N	N	N
1	Mon	ntfsdos	N	N	N	N
1	Mon	portsweep	Y	N	Y	Y
1	Mon	sshtrojan	N	N	N	N
1	Mon	portsweep	Y	N	Y	Y
1	Mon	xsnoop	N	N	N	N
1	Mon	snmpget	N	N	N	N
1	Mon	guesstelnet	N	N	N	N
1	Mon	portsweep	Y	N	Y	Y
1	Mon	guessftp	N	N	Y	N
1	Mon	ftpwrite	N	N	Y	Y
1	Mon	yaga	N	N	Y	Y
1	Mon	crashii	N	N	N	N
1	Mon	portsweep	Y	N	Y	N
1	Mon	secret	N	N	N	N
1	Mon	smurf	N	N	N	N
1	Tues	httptunnel	N	N	N	N
1	Tues	phf	N	N	Y	Y
1	Tues	loadmod	N	N	N	N
1	Tues	ps	N	N	N	N
1	Tues	ntfsdos	N	N	N	N
1	Tues	secret	N	N	N	N

1	Tues	sqlattack	N	N	N	N
1	Tues	sechole	N	N	N	N
1	Tues	land	N	N	Y	Y
1	Tues	mailbomb	N	N	N	N
1	Tues	processtable	N	N	N	N
1	Tues	crashii	N	N	Y	Y
1	Weds	satan	Y/M	N	Y	Y
1	Weds	nc-setup	N	N	Y	Y
1	Weds	imap	N	N	Y	Y
1	Weds	ppmacro	N	N	N	N
1	Weds	processtable	N	N	N	N
1	Weds	fdformat	N	N	N	N
1	Weds	nc-breakin	N	N	Y	Y
1	Weds	warez	N	N	N	N
1	Weds	arppoison	N	N	N	N
1	Weds	ncftp	N	N	Y	Y
1	Weds	secret	N	N	N	N
1	Weds	named	N	N	N	N
1	Weds	guessftp	N	N	Y	Y
1	Weds	smurf	N	N	N	N
1	Weds	guest	N	N	N	N
1	Weds	portsweep	Y	N	N	N
1	Weds	mailbomb	Y	N	N	N
1	Weds	guesstelnet	Y/M	N	Y	Y
1	Weds	snmpget	N	N	N	N
1	Thurs	teardrop	Y	Y	Y	Y

1	Thurs	netbus	N	N	Y	N
1	Thurs	sshtrojan	N	N	N	N
1	Thurs	dosnuke	N	N	Y	Y
1	Thurs	ncftp	N	N	N	N
1	Thurs	ppmarco	N	N	N	N
1	Thurs	guest	N	N	N	N
1	Thurs	xlock	N	N	N	N
1	Thurs	guesspop	N	N	N	N
1	Thurs	phf	N	N	Y	Y
1	Thurs	processtable	N	N	N	N
1	Thurs	mailbomb	N	N	N	N
1	Thurs	sqlattack	N	N	N	N
1	Fri	smurf	N	N	N	N
1	Fri	arppoison	N	N	N	N
1	Fri	sshtrojan	N	N	N	N
1	Fri	ipsweep	Y	N	Y/M	Y/M
1	Fri	xlock	N	N	N	N
1	Fri	named	N	N	N	N
1	Fri	portsweep	Y	N	Y/M	Y/M
1	Fri	ncftp	N	N	N	N
1	Fri	netbus	N	N	Y	N
1	Fri	mailbomb	N	N	N	N
1	Fri	ipsweep	Y	N	Y/M	Y/M
1	Fri	loadmod	N	N	N	N
1	Fri	sechole	N	N	N	N
1	Fri	portsweep	Y	N	Y/M	Y/M

1	Fri	ipsweep	Y	N	Y/M	Y/M
1	Fri	secret	N	N	N	N

Y/M: Attack may have been noticed based on Snort output, but was not intuitively obvious

<u>1999 Test Data Week 2</u>

Week	Day	Attack Name	Snort 1.7 F	Snot 1.7 C	Snort 1.8 F	Snort 1.8 C
2	Mon	pod	Y	Y	Y	Y
2	Mon	portsweep	Y	N	Y	Y
2	Mon	pod	Y	N	Y	Y
2	Mon	pod	Y	N	Y	Y
2	Mon	warezclient	N	N	N	N
2	Mon	smurf	N	N	Y	Y
2	Mon	portsweep	Y	N	Y	Y
2	Mon	apache2	N	N	N	N
2	Mon	guesstelnet	N	N	N	N
2	Mon	dosnuke	N	N	Y	Y
2	Mon	loadmodule	N	N	N	N
2	Mon	ffbconfig	N	N	N	N
2	Mon	smurf	N	N	Y	Y
2	Mon	arppoison	N	N	N	N
2	Mon	apache2	N	N	N	N
2	Mon	pod	Y	N	Y	Y
2	Mon	imap	N	N	Y	Y
2	Mon	ipsweep	N	N	Y	Y
2	Mon	dict	N	N	N	N
2	Mon	syslogd	N	N	N	N
2	Mon	neptune	N	N	Y	Y
2	Mon	crashiis	N	N	Y	Y
2	Mon	ls_domain	N	N	N	N

2	Mon	dosnuke	N	N	Y	Y
2	Mon	udpstorm	N	N	N	N
2	Mon	selfping	N	N	Y	Y
2	Mon	ncftp	N	N	Y	Y
2	Tues	tcpreset	N	N	N	N
2	Tues	teardrop	Y	Y	Y	Y
2	Tues	casesen	N	N	N	N
2	Tues	xsnoop	N	N	N	N
2	Tues	selfping	N	N	N	N
2	Tues	xterm	N	N	N	N
2	Tues	ftpwrite	N	N	Y	Y
2	Tues	back	N	N	Y	Y
2	Tues	ps	N	N	N	N
2	Tues	neptune	N	N	Y	Y
2	Tues	httptunnel	N	N	N	N
2	Tues	eject	N	N	N	N
2	Tues	pod	Y	Y	Y	Y
2	Tues	yaga	N	N	N	N
2	Tues	crashiis	N	N	N	N
2	Tues	ppmacro	N	N	N	N
2	Tues	syslog	N	N	N	N
2	Tues	perl	N	N	N	N
2	Tues	fdformat	N	N	N	N
2	Tues	secret	N	N	N	N
2	Tues	queso	N	N	N	N
2	Tues	neptune	N	N	Y	Y

2	Tues	dosnuke	N	N	Y	Y
2	Tues	portsweep	Y	N	Y	Y
2	Tues	ncftp	N	N	N	N
2	Weds	udpstorm	N	N	N	N
2	Weds	selfping	N	N	N	N
2	Weds	xlock	N	N	N	N
2	Weds	phf	N	N	Y	Y
2	Weds	tcpreset	N	N	N	N
2	Weds	netbus	N	N	N	N
2	Weds	back	N	N	N	N
2	Weds	netcat	N	N	N	N
2	Weds	queso	N	N	Y	Y
2	Weds	portsweep	Y	N	Y	Y
2	Weds	perl	N	N	N	N
2	Weds	queso	N	N	N	N
2	Weds	snmpget	N	N	N	N
2	Weds	processtable	N	N	N	N
2	Weds	back	N	N	N	N
2	Weds	ffbconfig	N	N	N	N
2	Weds	apache2	N	N	N	N
2	Weds	portsweep	Y	N	N	N
2	Thurs	ps	N	N	N	N
2	Thurs	phf	N	N	Y	Y
2	Thurs	casesen	N	N	N	N
2	Thurs	ntfsdos	N	N	N	N
2	Thurs	portsweep	Y	N	Y	Y

2	Thurs	ntinfoscan	N	N	N	N
2	Thurs	yaga	N	N	N	N
2	Thurs	crashiis	N	N	Y	Y
2	Thurs	httptunnel	N	N	N	N
2	Thurs	fdformat	N	N	N	N
2	Thurs	satan	N	N	Y	Y
2	Thurs	teardrop	Y	Y	Y	Y
2	Thurs	sechole	N	N	N	N
2	Thurs	resetscan	N	N	N	N
2	Thurs	ipsweep	Y	N	Y	Y
2	Thurs	snmpget	Y	N	N	N
2	Thurs	ntinfoscan	N	N	N	N
2	Thurs	ls_domain	N	N	N	N
2	Thurs	warez	N	N	N	N
2	Thurs	mscan	Y	N	Y	Y
2	Thurs	arppoison	N	N	N	N
2	Fri	portsweep	N	N	Y	Y
2	Fri	xsnoop	N	N	N	N
2	Fri	crashiis	N	N	Y	Y
2	Fri	insidesniffer	N	N	Y	Y
2	Fri	back	N	N	N	N
2	Fri	insidesniffer	N	N	Y	Y
2	Fri	netcat	N	N	Y	Y
2	Fri	xterm	N	N	N	N
2	Fri	portsweep	Y	N	N	N
2	Fri	anypw	N	N	N	N

2	Fri	guest	N	N	N	N
2	Fri	tcpreset	N	N	N	N
2	Fri	perl	N	N	N	N
2	Fri	framespoofer	N	N	N	N
2	Fri	portsweep	N	N	Y	Y
2	Fri	sqlattack	N	N	N	N
2	Fri	yaga	N	N	N	N
2	Fri	crashiis	N	N	Y	Y
2	Fri	telnet	N	N	N	N
2	Fri	crashiis	N	N	Y	Y
2	Fri	syslogd	N	N	N	N
2	Fri	eject	N	N	N	N
2	Fri	land	N	N	Y	Y
2	Fri	syslogd	N	N	N	N
2	Fri	sendmail	N	N	N	N
2	Fri	xterm	N	N	N	N
2	Fri	neptune	N	N	Y	Y
2	Fri	perl	N	N	N	N
2	Fri	warez	N	N	N	N
2	Fri	queso	Y	N	N	N
2	Fri	cassen	N	N	N	N
2	Fri	secret	N	N	N	N

APPENDIX F: NUMBER OF ALERTS

This appendix give the total amount of alerts generated each day by each of the Snort rule sets.

Year	Week	Day	Snort 1.7 F	Snort 1.7 C	Snort 1.8 F	Snort 1.8 C
1998	6	Mon	6627	0	154	8
1998	6	Tue	5431	10	182	41
1998	6	Wed	6337	0	4560	4439
1998	6	Thu	> 1 x 10^6	289	> 1 x10^6	> 1 x10^6
1998	6	Fri	> 500000	100	> 500000	> 500000
1998	7	Mon	303	0	375	216
1998	7	Tue	307	10	474	302
1998	7	Wed	921	100	1051	897
1998	7	Thu	118191	0	115860	8756
1998	7	Fri	77237	0	80587	50734
1999	1	Mon (inside)	55205	0	723	61
1999	1	Mon (outside)	21656	0	504	45
1999	1	Tues (inside)	0	0	0	0
1999	1	Tues (outside)	850	0	831	145
1999	1	Weds (inside)	6015	0	1503	202
1999	1	Weds	1516	0	1344	195

		(outside)				
1999	1	Thurs (inside)	1335	13	1389	271
1999	1	Thurs (outside)	1362	0	1218	248
1999	1	Fri (inside)	6512	0	592	139
1999	1	Fri (outside)	2229	0	765	129
1999	2	Mon (inside)	9833	1	1120	265
1999	2	Mon (outside)	10691	1	1266	620
1999	2	Tues (inside)	1284	55	8540	3834
1999	2	Tues (outside)	1270	45	8306	4101
1999	2	Weds (inside)	1119	0	392	358
1999	2	Weds (outside)	1083	0	542	513
1999	2	Thurs (inside)	3636	11	5340	5160
1999	2	Thurs (outside)	3600	11	5598	5459
1999	2	Fri (inside)	1249	0	1976	1787
1999	2	Fri	1223	0	1916	1781

		(outside)				

APPENDIX G: FULL ATTACK DATABASE

The full listing of all the attacks used in the 1998 and 1999 evaluation is available at http://www.ll.mit.edu/IST/ideval/docs/1999/attackDB.html. It was included in this report to explain the abbreviations for attacks listed in the results section. They are divided into five major sections, defined by attack type, which are Denial of Service, User to Root, Remote to Local, Probes, and Data.

<u>Denial of Service Attacks</u>

Apache2	The Apache2 attack is a denial of service attack against an apache web server where a client sends a request with many http headers. If the server receives many of these requests it will slow down, and may eventually crash.
arppoison	ARP Poison is a Denial of Service attack that was developed specifically for the 1999 MIT-LL Evaluation. In this attack the goal is to trick hosts on the same ethernet into "learning" the wrong "Mac" address for known IP addresses. The attacker must have access to the Local Area Network.
Back	In this denial of service attack against the Apache web server, an attacker submits requests with URL's containing many frontslashes. As the server tries to process these requests it will slow down and becomes unable to process other requests.
Crashiis	CrashIIS is a Denial of Service attack against the NT IIS webserver. The attacker sends a malformed GET request via telnet to port 80 on the NT victim. The command "GET ../.." crashes the web server and sometimes crashes the ftp and gopher daemons as well, because they are part of IIS.
dosnuke	DoSNuke is a Denial of Service attack that sends Out Of Band data (MSG_OOB) to port 139 (NetBIOS), crashing the NT victim (bluescreens the machine.
Land	The Land attack is a denial of service attack that is effective against some older TCP/IP implementations. The only vulnerable platform used in the 1998 DARPA evaluation was SunOS 4.1. The Land attack occurs when an attacker sends a spoofed SYN packet in which the source address is the same as the destination address.

Mailbomb	A Mailbomb is an attack in which the attacker sends many messages to a server, overflowing that server's mail queue and possible causing system failure.
SYN Flood (Neptune)	A SYN Flood is a denial of service attack to which every TCP/IP implementation is vulnerable (to some degree). Each half-open TCP connection made to a machine causes the 'tcpd' server to add a record to the data structure that stores information describing all pending connections. This data structure is of finite size, and it can be made to overflow by intentionally creating too many partially-open connections.
Ping Of Death	The Ping of Death is a denial of service attack that affects many older operating systems. Although the adverse effects of a Ping of Death could not be duplicated on any victim systems used in the 1998 DARPA evaluation, it has been widely reported that some systems will react in an unpredictable fashion when receiving oversized IP packets. Possible reactions include crashing, freezing, and rebooting.
Process Table	The Process Table attack is a novel denial-of-service attack that was specifically created for this evaluation. The Process Table attack can be waged against numerous network services on a variety of different UNIX systems. The attack is launched against network services which fork() or otherwise allocate a new process for each incoming TCP/IP connection.
selfping	The selfping attack is a denial of service attack in which a normal user can remotely reboot a machine with a single ping command. This attack can be performed on Solaris 2.5 and 2.5.1.
Smurf	In the "smurf" attack, attackers use ICMP echo request packets directed to IP broadcast addresses from remote locations to create a denial-of-service attack address of many subnets, resulting in a large, continuous stream of 'ECHO' replies that flood the victim.
sshprocesstable	SSH Processtable is similar to the processtable attack in that the goal of the attacker is to cause sshd daemon on the victim to fork so many children that the victim can spawn no more processes. This is due to a kernel limit on the number of processes that the OS will allow.
Syslogd	The Syslogd exploit is a denial of service attack that allows an attacker to remotely kill the syslogd service on a Solaris server. When Solaris syslogd receives an external message it attempts to do a DNS lookup on the source IP address. If this IP address doesn't match a valid DNS record, then syslogd will crash with a Segmentation Fault.

tcpreset	TCP Reset is a denial of service attack that disrupts TCP connections made to the victim machine. That is, the attacker listens (on a local or wide-area network) for tcp connections to the victim, and sends a spoofed tcp RESET packet to the victim, thus causing the victim to inadvertently terminate the TCP connection.
Teardrop	The teardrop exploit is a denial of service attack that exploits a flaw in the implementation of older TCP/IP stacks. Some implementations of the IP fragmentation re-assembly code on these platforms does not properly handle overlapping IP fragments.
Udpstorm	A Udpstorm attack is a denial of service attack that causes network congestion and slowdown. When a connection is established between two UDP services, each of which produces output, these two services can produce a very high number of packets that can lead to a denial of service on the machine(s) where the services are offered. Anyone with network connectivity can launch an attack; no account access is needed.

<u>User to Root Attacks</u>

anypw	NukePW is a Console User to Root attack that allows the attacker to logon to the system without a password. A boot disk is used to modify the NT authentication package so that a valid username can login with any password string. Logins via telnet also work with any password.
casesen	CaseSen is a User to Root attack that exploits the case sensitivity of the NT object directory. The attacker ftps three attack files to the victim: soundedt.exe, editwavs.exe, psxss.exe (the names of the files were chosen to make the attack more stealthy). The attacker then telnets to the victim and runs soundedt.exe. A new object is created in the NT object directory called \??\c: which links to the directory containing the attack files. A posix application is started activating the trojan attack file, psxss.exe, which results in the logged in user being added to the Administrators user group.
Eject	The Eject attack exploits a buffer overflow is the 'eject' binary distributed with Solaris 2.5.
Ffbconfig	The Ffbconfig attack exploits a buffer overflow is the 'ffbconfig' program distributed with Solaris 2.5.

Fdformat	The Fdformat attack exploits a buffer overflow is the 'fdformat' program distributed with Solaris 2.5. The fdformat program formats diskettes and PCMCIA memory cards. The program also uses the same volume management library, libvolmgt.so.1, and is exposed to the same vulnerability as the eject program.
Loadmodule	The Loadmodule attack is a User to Root attack against SunOS 4.1 systems that use the xnews window system. The loadmodule program within SunOS 4.1.x is used by the xnews window system server to load two dynamically loadable kernel drivers into the currently running system and to create special devices in the /dev directory to use those modules. Because of a bug in the way the loadmodule program sanitizes its environment, unauthorized users can gain root access on the local machine.
ntfsdos	This console-based attack reboots the system from a floppy disk containing NTFSDOS.EXE. This executable is used to mount the hard drives, giving the attacker the ability to read and copy files that would otherwise be protected by Windows NTFS security. The attack may be consider a User to Root attack because the attacker can access files that only the Administrator has permission to use.
Perl	The Perl attack is a User to Root attack that exploits a bug in some Perl implementations.
Ps	The Ps attack takes advantage of a race condition in the version of 'ps' distributed with Solaris 2.5 and allows an attacker to execute arbitrary code with root privilege.
sechole	The attacker (a regular user) ftps to the victim and uploads test.exe and testfile.dll (filenames were chosen to be stealthy). The attacker then telnets to the victim and runs test.exe. The result is the attacker is added to the Administrators group.
Xterm	The Xterm attack exploits a buffer overflow in the Xaw library distributed with Redhat Linux 5.0 (as well as other operating systems not used in the simulation) and allows an attacker to execute arbitrary instructions with root privilege.
yaga	Yaga is a User-to-Root attack. It adds the attacker to the Domain Admins group by hacking the registry. The attacker edits the victim's registry so that the next time a system service crashes on the victim, the attacker is added to the Domain Admins group.

Remote to Local

Dictionary	The Dictionary attack is a Remote to Local User attack in which an attacker tries to gain access to some machine by making

	repeated guesses at possible usernames and passwords. Users typically do not choose good passwords, so an attacker who knows the username of a particular user (or the names of all users) will attempt to gain access to this user's account by making guesses at possible passwords.
FrameSpoofer	This attacks tricks the victim into believing he is viewing a trusted web site, but in actuality the page's main body is spoofed with a frame created by the attacker.
Ftp-write	The Ftp-write attack is a Remote to Local User attack that takes advantage of a common anonymous ftp misconfiguration. The anonymous ftp root directory and its subdirectories should not be owned by the ftp account or be in the same group as the ftp account. If any of these directories are owned by ftp or are in the same group as the ftp account and are not write protected, an intruder will be able to add files (such as an rhosts file) and eventually gain local access to the system.
Guest	The Guest attack is a variant of the Dictionary attack described in Section 8.1. On badly configured systems, guest accounts are often left with no password or with an easy to guess password. Because most operating systems ship with the guest account activated by default, this is one of the first and simplest vulnerabilities an attacker will attempt to exploit.
HttpTunnel	In an Http Tunnel attack, the attacker gains local access to the machine to be attacked and then sets up and configures an http client to periodically query a web server that the attacker has setup at some remote host. When the client connects, the server is able to send cookies that could request information be sent by the client, such as the password file on the victim machine. In effect, the attacker is able to "tunnel" requests for information through the http protocol.
Imap	The Imap attack exploits a buffer overflow in the Imap server of Redhat Linux 4.2 that allows remote attackers to execute arbitrary instructions with root privileges. The Imap server must be run with root privileges so it can access mail folders and undertake some file manipulation on behalf of the user logging in.
Named	The Named attack exploits a buffer overflow in BIND version 4.9 releases prior to BIND 4.9.7 and BIND 8 releases prior to 8.1.2. An improperly or maliciously formatted inverse query on a TCP stream destined for the named service can crash the named server or allow an attacker to gain root privileges.
ncftp	Ncftp is an ascii UI ftp program for linux. This attack exploits one of the popular features of the program: the ability to get

	subdirectories recursively. New (sub)directories are created on the local machine using the system() command (e.g. if any directories on the remote host contain an expression in backticks, that expression will be evaluated on the local machine when the directory is created.
netbus	NetBus is a Remote to Local attack. The attacker uses a trojan program to install and run the Netbus server on the victim machine. Once Netbus is running, it acts as a backdoor. The attacker can then remotely access the machine using the Netbus client.
netcat	NetCat is a Remote to Local attack. The attacker uses a trojan to install and run the netcat program on the victim machine on a specific port (53). Once netcat is running, it acts as a backdoor. The attacker can remotely access the machine through the netcat port without a username or password.
Phf	The Phf attack abuses a badly written CGI script to execute commands with the privilege level of the http server. Any CGI program which relies on the CGI function escape_shell_cmd() to prevent exploitation of shell-based library calls may be vulnerable to attack. In particular, this vulnerability is manifested by the "phf" program that is distributed with the example code for the Apache web server.
ppmacro	This Remote to Local attack uses a trojan PowerPoint macro to read secret files. This attack is based on a particular scenario. The victim user usually receives PowerPoint templates from an outside source via email attachment. He runs a built-in macro which inserts a graph displaying web statistics, saves the presentation as a ppt file, and posts it on the web.
Sendmail	The Sendmail attack exploits a buffer overflow in version 8.8.3 of sendmail and allows a remote attacker to execute commands with superuser privileges. By sending a carefully crafted email message to a system running a vulnerable version of sendmail, intruders can force sendmail to execute arbitrary commands with root privilege.
sshtrojan	In SSH Trojan attack, the attacker tricks the system administrator into installing (as a "Y2K Upgrade") a trojan version of the SSH program. This trojan version allows the attacker (or anyone!) to login to the victim, via ssh, with the login "monkey" and no password. Upon login, a root priviledge shell is spawned for the attacker.
Xlock	In the Xlock attack, a remote attacker gains local access by fooling a legitimate user who has left their X console unprotected,

	into revealing their password. An attacker can display a modified version of the xlock program on the display of a user who has left their X display open (as would happen after typing 'xhost +'), hoping to convince the user sitting at that console to type in their password.
Xsnoop	In the Xsnoop attack, an attacker watches the keystrokes processed by an unprotected X server to try to gain information that can be used gain local access the victim system. An attacker can monitor keystrokes on the X server of a user who has left their X display open. A log of keystrokes is useful to an attacker because it might contain confidential information, or information that can be used to gain access to the system such as the username and password of the user being monitored.

Probes

insidesniffer	Here the attacker merely attaches a new machine to an inside ethernet hub, configured with an ip, and begins sniffing traffic.
Ipsweep	An Ipsweep attack is a surveillance sweep to determine which hosts are listening on a network. This information is useful to an attacker in staging attacks and searching for vulnerable machines.
ls_domain	Here the attacker uses the "nslookup" command in interactive mode to "list" all machines in a given DNS domain from a mis-configured primary or secondary DNS server. Thus the attacker can learn what machines (IP addresses) belong to (and perhaps exist in) the domain.
Mscan	Mscan is a probing tool that uses both DNS zone transfers and/or brute force scanning of IP addresses to locate machines, and test them for vulnerabilities.
Nmap	Nmap is a general-purpose tool for performing network scans. Nmap supports many different types of portscansùoptions include SYN, FIN and ACK scanning with both TCP and UDP, as well as ICMP (Ping) scanning [45]. The Nmap program also allows a user to specify which ports to scan, how much time to wait between each port, and whether the ports should be scanned sequentially or in a random order.
NTinfoscan	NTInfoScan is a NetBIOS based security scanner. It scans the NT victim to obtain share information, the names of all the users, services running, and other information. The results are saved in an html file named .html where victim is the victim's hostname.

queso	QueSO is a utility used to determine a what type of machine/operating system exists at a certain IP adress. QueSO sends a series of 7 tcp packets to any one port of a machine and uses the return packets it receives to lookup the machine in a database of responses.
resetscan	ResetScan sends reset packets to a list of IP addresses in a subnet to determine which machines are active. If there is no response to the reset packet, the machine is alive. If a router or gateway responds with "host unreachable," the machine does not exist.
Saint	SAINT is the Security Administrator's Integrated Network Tool. In its simplest mode, it gathers as much information about remote hosts and networks as possible by examining such network services as finger, NFS, NIS, ftp and tftp, rexd, statd, and other services.
Satan	SATAN is an early predecessor of the SAINT scanning program described in the last section. While SAINT and SATAN are quite similar in purpose and design, the particular vulnerabilities that each tools checks for are slightly different.

Data

Secret	A "secret" attack is an attack where the attacker maliciously or mistakenly transfers data which they have access to to a place where it doesn't belong. For example, transferring data from a classified computer/network to a non-classified computer/network would constitute a "secret" attack.