FLOWHUNT— Specification-Based Intrusion Detection using Neural Networks

Gregory V. Bard

October 3, 2003

Abstract

Intrusion Detection is vital to protecting military, governmental and commercial infrastructures. Intrusion Detection was once normally divided into two classes: Anomaly-Based and Signature-Based. Signature-Based requires attacks to be known before they can be detected. Anomaly-Based can find new attacks, but is computationally intensive and is prone to false alarms. Is there another way? Specification-Based Intrusion Detection compares each packet with its protocol specification to ensure its legitimacy, first defined by Ko, Fink and Levitt [10]. It can detect new attacks before a signature is developed, but is more computationally intensive than a signature. FLOWHUNT is an IDS that uses Specification-Based Intrusion Detection to detect application-layer attacks.

This paper will show how FLOWHUNT can protect FTP, NNTP, SMTP, POP, and HTTP transactions. We will demonstrate that Specification-Based testing can discover vulnerabilities, and is not so complex as to create a bottleneck for traffic. There is a low false alarm rate, and an experimentally verified 100% detection rate for attacks within its domain in the Lincoln Labs data set.

Keywords: Intrusion Detection, Neural Networks, Specification-Based, Network Protocol Security.

1 Introduction

The field of Intrusion Detection Systems (IDS) has become increasingly vital over the last decade as network information systems have grown into the daily life of most businesses, government agencies and private citizens. While commercial products and theoretical designs are quite diverse, they have classically been divided by two parameters into four camps.[1]

The first parameter is host-based versus network-based. However, in a heterogeneous network consisting of many diverse operating systems, it is a challenge to design a system that can run on each machine. Moreover, large Internet Service Providers (ISPs) or universities may be unable by law, regulation or good business practices, to compel their users to run a particular IDS program. Network-based systems naturally have a higher work load, but have the advantage of operating without interfering with (or the knowledge and interference of) the end user. The second parameter has been signature-based versus anomaly-based. This classical trade-off has characterized much IDS debate in the Department of Defense, since while signature-based is by definition incapable of detecting novel attacks, it is multiple decimal orders of magnitude faster and has a low false alarm rate [1].

1.1 Specification Based Intrusion Detection

While the traditional distinction of anomaly-based versus signature-based detection was once considered a descriptive partition of the set of Intrusion Detection systems, a third category, specification-based, has evolved.

[1] As designs evolve these distinctions become misleading, as most IDSs will incorporate mechanisms from multiple camps, but individual IDS functions can still be well described in this nomenclature.

What is Specification-Based Intrusion Detection? The purpose of an intrusion detection system is to define a boundary between acceptable and non-acceptable activity, and detect events crossing that boundary. Specification based systems are categorized by the structure of this boundary.

While signature-based detection operates from a set of descriptions of known attacks (signatures), and anomaly-based detection operates from statistics of the host or network shown either experimentally or through experience to represent intrusions, specification-based detection operates by specifying the bounds of legal activity within a protocol. The bounds are defined by a policymaker with heavy reliance on the original specification.

Thus signature-based requires expert foreknowledge of attacks, anomaly-based requires expert foreknowledge of user and host statistics, while specification-based requires expert foreknowledge of communication protocols.

Advantages Like anomaly-based detection, specification-based detection has the capacity to detect novel intrusions, while signature-based detection requires foreknowledge of the description and features of an attack (the signature) and thus cannot defend against novel attacks. More on novel attacks will be discussed in Section 4.7.

However, like signature-based detection, specification-based systems will only alarm on activity that is defined to be intrusive, not merely uncommon activity—which will trigger anomaly-based systems. This reduces the false alarm rate, provided that the specification used is actually accurate. If there exist valid activities which fall outside the specification, then false alarms will occur when these activities take place. Thus the core of a specification-based system are the boundaries defined for "non-malicious" activity.

Through RFCs and other standards documents, these boundaries are pre-defined. However, they must converted into a precise logical definition, which is currently done manually by a human. Note, however, that this need be done only once for each protocol, and while attacks and user statistics change very frequently, basic Internet protocols evolve slowly. This bounds the effort required from system designers.

Also note that while signature-based and anomaly-based systems are "innocent until proven guilty" approaches, specification-based is "guilty until proven innocent." This means that "arbitrary" (non-carefully crafted) traffic will alarm the IDS under specification-based systems, aiding in the detection of poorly crafted traffic such as an amateur denial-of-service.

Disadvantages Beyond the human-effort in defining these specification, there are other disadvantages. Certainly, specification-based detection is more computationally intensive than signature-based. The precise computational requirements of our system, FLOWHUNT, are measured.

Also, as new protocols are introduced, specifications must be created. However, in a high-security network, such as milnet or other government entities, there exists no motivation to permit Internet applications beyond those required for work (e.g. HTTP, IMAP or POP/SMTP, FTP or SCP, Telnet or SSH, et cetera...) Most new protocols that have achieved widespread popularity, creating labor for the signature-based community, have been in the leisure arena. (e.g. file sharing, streaming radio or instant messaging.) Specification-based detection will be found to be more useful in networks where the users can be restricted in the protocols used.

1.2 Origins of Specification-Based Intrusion Detection

The National Security Agency and DARPA initiated research into Specification-Based Intrusion Detection through a contract at the University of California at Davis, with Calvin Ko, George Fink and Karl Levitt in 1993 [10]. The result was a program policy specification language, for specific privileged BSD UNIX programs, and an execution monitor which would analyze audit trails compared to the specification. The specification was written in logical and regular expressions, and delineated the behavior of those particular programs.

Ko & Levitt continued this research at UC-Davis, and in 1997 with Manfred Ruschitzka, published [11], which generalized their results. The new system included security violations caused by improper

synchronization in distributed programs. While the above in [10] was clearly Host-Based (and not Network-Based like FLOWHUNT,) in [11] Specification-Based IDS's begin to examine network traffic.

DARPA continued research through Bellcore and Iowa State University. Sekar, Segal and Cai published [14], which focused on an application program's legitimate interactions with its operating system. Moreover, Sekar, Segal and Cai added fault-detection to the specification-based environment. Further, an isolation mechanism was added, allowing administrators to isolate specific programs from the operating system, in order to deceive and monitor attackers—an outstanding damage control tool. Also of interest is Cai's Master's Thesis [3] which describes the design of the Auditing Specification Language (ASL).

A specific implementation of the Specification-Based method, is the Cheung and Levitt publication "A Formal Specification-Based Approach for Protecting the Domain Name System," which as the name suggests is intended to protect DNS [16]. This work is most similar to FLOWHUNT in that the system is attempting to protect an application layer protocol by doing specification-based IDS from the network. It differs from FLOWHUNT in that the analysis is done from a special wrapper, which essentially and materially changes the way packets are handled on receipt and transmission, whereas FLOWHUNT is passive. Another example is a paper by Tseng, Balasubramanyam, Ko, Limprasittiporn, Rowe and Levitt, which applies specification-based intrusion detection to AODV (The Ad Hoc On Demand Distance Vector) Routing Protocol [4]. That paper uses a state-machine for analyzing specification compliance, as does FLOWHUNT.

Levitt & Ko continued their research through DARPA, with Paul Brutch, Jeff Rowe, and Guy Tsafnat, published [9], describing System Health and Intrusion Monitoring (SHIM.) The system (while executing) will be monitored for violation of a hierarchy of constraints which describe correct behavior. SHIM does not detect intrusions directly, but the violations of these constraints which intrusions cause. SHIM is applicable to both network protocols and host programs. Balepin, Maltsev, Rowe and Levitt also have created the "automated response broker" and described how it can be used to enable automated Intrusion Response Systems, coupled with a specification-based host-based Intrusion Detection System [2].

Two papers have been published recently which attempt to apply Specification-Based Intrusion Detection to the Lincoln Labs data set. These papers [17, 15], both by Sekar et al., were released after the experiments performed were completed; however, we can compare our results. In [15], a layer of the IDS exists between packet reception and processing, and another layer between system call reception and processing. The later layer makes the IDS host-based, as it resides inside the operating system. The system described in [17] is described in section 5.

2 The FLOWHUNT Proof-of-Concept

In order to keep the project limited in scale, it was decided to focus initially on the Application Layer, and five protocols were chosen: HTTP, FTP, POP, NNTP and SMTP. The goal was to demonstrate that specification-based testing at the application layer can detect intrusions, that it is not computationally infeasible, that false alarms will be bounded, that the system would be scalable, and to determine efficient methods for the execution of these goals. Research continued intermittently through 1999–2001, and then the project was canceled.

2.1 Design Methodology

In selecting a set of protocols for FLOWHUNT, five principle applications were selected. While HTTP, NNTP, POP3, SMTP, and FTP are very common, they have other advantages [20]. First, they have relatively large command sets. Second, they contain overlapping commands— for example, USER and PASS are used by several of them. This gives rise to some difficulty for the neural net due to this ambiguity. (For example, USER is found, but is it USER from FTP, or USER from POP?) This "extra" difficulty would ensure that our experiment would capture the advantages and disadvantages of specification-based Intrusion Detection. After all, if this type of system can differentiate between protocols with this overlap, then surely it would be able to differentiate in the absence of any overlap.

For the data set, the 1998 DARPA MIT Lincoln Labs Intrusion Detection data set was used, which is a nine-week sample of network traffic, five days a week for 18 hours a day, (700 MB/day, or 31 GB total) from a network assembled to provide examples of intrusions in a background of natural network traffic among several operating systems [18, 7, 12, 5, 13]. This set has been used by several other IDS tests, including the specification-based IDS in [17, 15], to which FLOWHUNT will be compared and contrasted. Each attack is described in detail in Kendall's Master's Thesis, which was an absolutely invaluable resource [8]. For neural network training, the data set was conveniently divided into a seven-week training set, and a two-week testing set. The training set contained both malicious and normal network traffic, and was fully annotated. The training of the neural net required less than a day of processor time on an Ultra SPARC 60.

A Note about the Age of the Data Sets The author is aware of the age of the dataset used in the testing of the system. However, the Lincoln Labs set is valuable for several reasons. First, any errors which may exist in it are long since identified, since the data has been available for a time, and has been analyzed by many. Second, a wide variety of attacks and operating systems are represented. Third, most of this work was done in the years 1999–2000, when fewer options were available, and the research cannot be repeated as the project is canceled. Still the lessons learned here may be valuable to other researchers who may attempt to engage in parallel research, or attempt to apply this structure to other protocols.

3 FLOWHUNT Structure

FLOWHUNT is a proof-of-concept of the Specification-Based IDS idea, and uses custom state-machines designed for 5 major TCP applications to detect attacks against common network applications, as interpreted by a Neural Network. A similar state-machine is used by Tseng et al, [4] in protecting AODV, an ad hoc routing protocol.

The FLOWHUNT system consists of three pieces:

- The Preprocessor (tcpclaunch)
- A Set of state machines that describes each protocol's state transitions (ftp.h, http.h, ...)
- The neural network, which generates alarms and flags suspicious sessions for retention.

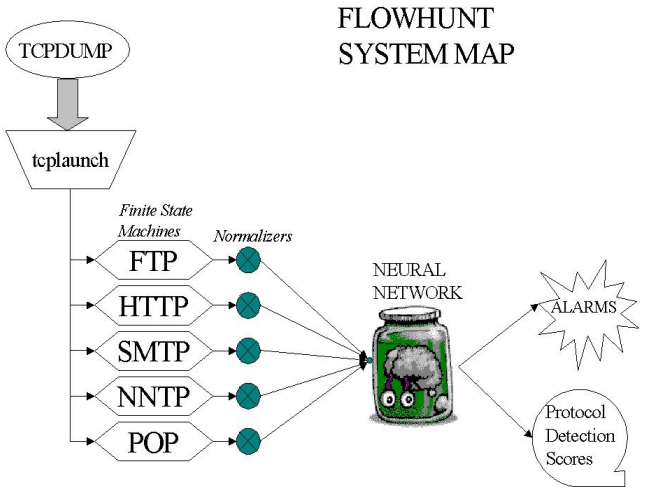

Figure 1: A System Diagram of FLOWHUNT

The system map is given in Figure 1. The preprocessor takes the network traffic (as provided by `tcpdump`) and parses it into TCP sessions. When each session terminates (two FINs or one RESET, or a user-set time-out), it gets passed to the next stage, where a copy is given to each state machine for analysis. Each state

machine extracts what would be commands if that session were actually from its protocol, parsed according to that protocol's rules. The state machines then categorize each command and response into one of four buckets for each direction (4 each for commands, and 4 each for responses) and an "unknown" bucket. The buckets are explained below.

The number of packets in each bucket, or bucket counts, (9 buckets $\times$ 5 protocols = 45 numbers) are fed through a consolidator/normalizer and then into the neural network. The neural network calculates, and outputs five real values on the interval [0,1]. This represents a prediction that the session is a valid use of one each of the five protocols. Finally, alarms are generated or sessions are flagged, if necessary.

3.1 Input

The input to FLOWHUNT is a stream of packets in libpcap format, as supplied by `tcpdump`. While FLOWHUNT is intended to run "live," it can also be run on stored data to check efficiency or detect attacks in a forensic circumstance. FLOWHUNT currently analyzes TCP traffic only, not UDP and ICMP. However, one could expand FLOWHUNT to other protocols.

As the various TCP packets enter the system, they are aggregated by socket session (in/out IP address and in/out port). When a session is closed, either by a RESET or two FINs, the session is considered complete and is forwarded to each state machine, as explained below. Since the entire session is present, it can be processed statefully, yet very little processing is done on receipt. At first it may seem like a disadvantage to process only on session completion. This does prevent attacks from being stopped midway during a session. However, there are four primary reasons for this design decision:

Stateful Analysis with Stateless In-Processing To execute a stateful analysis of packets as they arrive could cause bottlenecks, or require the storage of large amounts of state data. Alternatively, stateless analysis of packets can only detect attacks entirely contained in one packet. Therefore, storing the packets into session groups allows the sessions to be analyzed statefully, but keeps the actions upon receipt of a packet simple.

Prevents Bottlenecks Analyzing at the session level, rather than the packet level, makes handling a packet arrival easy, and reduces the number of times that a connection will be scrutinized (once, as compared to once per each packet.)

Halting an Attack is Unlikely Intercepting an attack while it is midway through completion is unlikely, because most countermeasures will need to be confirmed by a human. If not, then automated countermeasures could lead to denial of service attacks.

Facilitates Record Keeping Often times, for forensic purposes or to understand how an attack works, it is useful to have the entire session stored intact. If the packets were analyzed individually and not session-by-session then only those packets *after* the offending packet would be stored. (Unless all traffic is stored which is rather difficult.)

Data Sources used during Development The training data used for the Neural Network was the seven weeks of training data from the DARPA 1998 Intrusion Detection Evaluation test suite, described above. The testing data for attack detections also came from this suite [18]. In addition two weeks of test data was also made available, identically marked. This was used to produce our accuracy results.

For testing the speed of the system, a US military command provided us with packet captures of their own networks, under a special agreement. This was live traffic of real users and real hosts. This provides a good simulation of how FLOWHUNT would behave in a "real-world" deployment. More importantly, after looking at the Lincoln Labs data alone, it is not obvious that the results achieved by FLOWHUNT would translate into equally good results in another environment, with differing attacks and network topologies. On the other hand, since over 100 GB of real packet traffic from a complex and changing network was inputted to FLOWHUNT, with no changes to the neural network or retraining, one can be confident that the behavior of FLOWHUNT on any network should be reasonably effective. Since the military traffic was

not annotated with known attacks, detection ratios cannot be calculated. However, of the small number of alarms generated, many were investigated and the false alarm rate was comparable to the results in this paper (See Section 4.)

Our system can currently handle only FTP, HTTP, SMTP, POP and NNTP sessions, and so all other protocols are blocked. Therefore, for both training and testing, all sessions without at least one port number being 21, 25, 110, 119, or 80, 8000 and 8080 were dropped. This was to minimize the number of alarms of "unknown protocol."

3.2 State Machine Organization

The purpose of the state-machine of a given protocol is to see if the commands issued are valid responses to what precedes them, assuming the protocol of that state-machine is in fact in use by that session. More precisely, each state-machine parses the session transcript, into commands and responses, as if its own protocol were the protocol being used. Therefore, gibberish should likely result if this were not the case.

Each session transcript gets passed to each state machine for evaluation. This is in contrast to traditional IDS systems, where a transaction on port 80 (http) for example, would be checked for web intrusions, web anomalies or web normality, and not for other protocols. That traditional mindset vests some degree of trust in the port number of the TCP packet. However, FLOWHUNT does not trust, allowing the detection of, as an example, FTP communications taking place on port 80, perhaps to bypass a firewall or logging mechanisms. This mistrust is useful, as many instant messaging and exfiltration (e.g. Kazaa) programs will masquerade on other ports, especially the unblockable port 80.

Packet Buckets Our system as currently implemented has such a state-machine for HTTP, FTP, NNTP, SMTP, and POP. As stated above, each command of the session is extracted. The sessions are interpreted as a series of commands and responses, each of which is interpreted in the context of the state of the session. Then each command/response is entered into one of the following buckets:

C_{exp} The string is a valid and expected command for the current state of the protocol.
R_{exp} The same as C_{exp}, but for responses.
C_{val} The string is a valid command, but unexpected. This can be rarely used features, or error conditions, or redundant commands.
R_{val} As above, but for responses.
C_{unx} The string is a command in this protocol, but not one valid in the current state. Therefore, something has gone awry.
R_{unx} As above, but for responses.
CR_{unk} The string is not a command or response in any part of the protocol. Since it is a totally unknown string, it cannot even be classified as a command or a response, which is why there is not a separate C_{unk} and R_{unk}.

Every command or response is placed in exactly one bucket, and the buckets are tabulated. Also, commands and responses often trigger a state-transition into a new state. These nine counts are fed into a normalizer, and then become the inputs to the neural network, described shortly.

Extracting Application Layer Commands Each session stream is then parsed by the state machine according to the rules of that protocol. Therefore, if the session is not communicating in the protocol of that state machine, it would be expected that the results would not fit the necessary structures at all, and would create a series of entirely unintelligible results.

Detecting Masquerading Since the "wrong" protocol appears unintelligible under a protocol state-machine, one can use that fact to detect one protocol masquerading as another. When an FTP server runs on port 80, masquerading as web traffic, it scores low (i.e. almost always 0) on HTTP, and high (i.e. almost always 1) on FTP. This is the reason why the system does not stop after the state-machines, which could be

used as valid/intrusive session detectors. By analyzing each session against each protocol, FLOWHUNT is more likely to detect exfiltration attempts, or masquerading services, rather than detect only attacks. One could even make the argument that all Computer Network Attacks that result in the release of classified information must involve an exfiltration at some point. Therefore, preventing successful exfiltration (or catching it after the fact) is as much of a success as preventing or detecting the attack that made it possible.

Generating the State Machines Creating the state-machine for a given protocol is done by analyzing the RFC and common implementations. The states tend to suggest themselves (for FTP, they are authentication, log in, et cetera...) In fact, the RFC for FTP itself suggested the format to be used for our state-machines [20]. While the protocols are often complex, there is a finite and very defined list of commands in every case. Therefore, the process of making a state-machine is merely one of organizing the activity into reasonable clusters of similar behavior, and defining the transitions between these behavior sets. And though while the protocols are complex, the state-machines are not and one can be confident that they are complete by simply checking that all typical uses of each command are present in the appropriate states, and omitted in others.

An Example: A Web Script Suppose an attacker is uncertain of the security of a target website. For a web script, where perhaps a filename is called for or some other variable length input, he/she generates a long string which if not truncated, will include commands for execution by the shell after the filename, thus allowing him/her the ability to execute arbitrary commands on the target machine. Or perhaps the target website will check the string, reject it, and create a 404 response. The former, with its long string and unusual characters would alarm under the HTTP state-machine, since it would be unlike normal HTTP packets. Yet, the latter or 404 response, while a "rare condition" or "error condition" is only one non-good response out of a series of commands and responses, and so is unlikely to alarm. In fact, the 404 response would be under "neutral" during normalization, see below.

3.3 Neural Network Utilization

The neural network has fifteen inputs. For each session, three inputs are provided from the data from each of the five state machines. However, each state machine outputs nine numbers. An aggregation of these nine quantities into three quantities (for five protocols this is fifteen inputs) was done to reduce the amount of input data to train the IDS. This aggregation does destroy some of the sensitivity gained by the granularity of nine buckets, but we hope to remove this aggregation in a later version of the system. The numbers are then normalized by the number of packets in the session to create a value between 0 and 1.

The three normalized numbers are calculated as follows:

$$F_{Good} = \frac{C_{exp} + R_{exp}}{total.}$$

$$F_{Neutral} = \frac{C_{old} + R_{old} + C_{val} + R_{val} + C_{unx} + R_{unx}}{total.}$$

$$F_{Bad} = \frac{CR_{unk}}{total.}$$

These three inputs per protocol all lie in the range [0,1]. This then generates a fifteen dimensional vector in the range [0,1]. At first it may seem counter-productive to dump the three middle levels of "unexpectedness" together into one "neutral" category. But it turns out that they are rare enough in legitimate traffic that a much larger data set would be required to train the neural network.

For the Neural Network we utilized "NevProp 3.0," a developed product from the University of Nevada at Reno [6] that is widely popular. The neural network produces five outputs, on the range [0,1], one for each protocol. This represents the prediction that the session was actually a session of that protocol. Usually, all five numbers are very near zero, (< 0.01) or— only one is near 1 (> 0.99) and the remainder are near zero. These represent the prediction that a session is of that protocol, or of none of the given protocols.

Why Utilize A Neural Network? Neural Networks have certain training time costs, and do not offer the same guarantees of reliability that strictly computational methods might. Furthermore, it may seem that we only need a map from the bucket populations to the predictor outputs. This would simply be five functions from $[0,1]^2 \rightarrow [0,1]$. (While there are three categories of "Good", "Neutral", and "Bad", they must add to 1, giving two degrees of freedom.) However, it turns out this is misleading.

Each protocol's bucket counts *do not* independently determine their own protocol predictions. Instead, low or high values on one input will influence each output. This has been verified experimentally, by creating a script which simply passes artificially generated random inputs to neural net to calculate the independence of inputs and outputs. The function is thus instead $[0,1]^{10} \rightarrow [0,1]^5$. There may be methods of finding a good function to carry out this mapping but a neural network converges rapidly to one with relatively short training time, in the special case of this program. Also, the relatively small number of nodes (15 input, 15 hidden, 5 output) means a moderate amount of data is sufficient for training, and the Lincoln Labs data set was highly satisfactory.

Scalability A short discussion about scalability was moved to Appendix B.

3.4 Outputs & Alarms

The neural network outputs a five element vector of floating-point numbers from 0 to 1, which represent the prediction that the given session is of a particular protocol. Two common forms are anticipated. If a session is very clearly belonging to one protocol, then a value near one is anticipated for that protocol, and values near zero for the other four protocols. If the session is not of any of the five protocols, then values near zero are expected everywhere. Alternatively, if one protocol is masquerading as another, effectively, one might see a low but non-zero value for the false protocol, and near 1.0 for the true protocol.

FLOWHUNT generates four principal alarms and can be programmed to flag sessions for other uses. The alarms are all consequences of the [0,1] predictions of protocol membership (one for each protocol) that each session receives after going through the neural network.

Low Absolute Score: This occurs when a session is on the "official" port for a protocol in the system, but scores less than 0.500 on that port. An example of this would be that an FTP session is taking place on port 80. The score of the HTTP state-machine (if the neural network is operating as intended) would be approximately 0.0 and the FTP session would be approximately 1.0, creating a Low Absolute Score Alarm.

Low Relative Score: This signifies when a session is on the "official" port for a protocol in the system, but the ratio of its score to that of the sum of the scores for all protocols is less than 0.500. For example, it scored higher on another port. This alarm detects situations where two protocol-predictions are near each other, even though the session scored "acceptable" on the protocol that matches its port number. For example, an FTP transaction on port 110 (POP3) might cause this result. This is usually concurrent with the above alarm.

High Absolute Score: This is a result of a session scoring above 0.500 on a protocol, when it is NOT on the official port of that protocol. In the example from the Low Absolute Score Alarm, the FTP-state machine should score 1.0 on that port 80 session.

Unknown Protocol: This occurs when a session goes between two ports, neither of which are "official" ports for a protocol in the system, (or optionally on a list of "legal exceptions.") The assumption is a tightly controlled military network, with a restrictive firewall and only approved software on-board, using a particular subset of the TCP/IP family of protocols.

Sessions can also be flagged. This means that a libpcap-format transcript is retained in a directory for archiving or for human analysis. All alarming sessions are flagged by default, but it is easy to add other flagging requirements. For example, an office might want to keep permanent `telnet` logs, especially if that protocol is not normally permitted.

4 Results

The primary measure of an IDS is naturally the extent to which it detects attacks. Also critical is keeping the false alarm rate low and throughput high. Finally it is also important to note whether the system's settings are stable or they are sensitive to small fluctuations in the inputs. All these criteria are quantified below.

4.1 Lincoln Labs 1998

In 1998 DARPA contracted out to Lincoln Labs at MIT to create a test data set to evaluate current IDS technology. The result was a seven-week training set, fully annotated, and a two-week test data set. This data set is documented in Michael Kendall's Master's Thesis [8]. Since all attacks were present in the second week, we used the second week of the test data set as our test case. The seven training weeks, as well as other sources, such as a US military command, provided data to train the neural network to arrive at its current weights.

We can categorize the 35 attacks in the Lincoln Labs Set into the following basic categories. The most important distinction for a specification-based IDS is whether the attack is violative or non-violative to the design parameters of the protocols used. In other words, is the attack traffic legal activity in the protocol?

Those that are violative are characterized by the target, which is either the Operating System (including the Windowing System), an Internet Service, or an Application Program. The Operating System category is further divided into two halves, those targeted at network components and those targeted elsewhere.

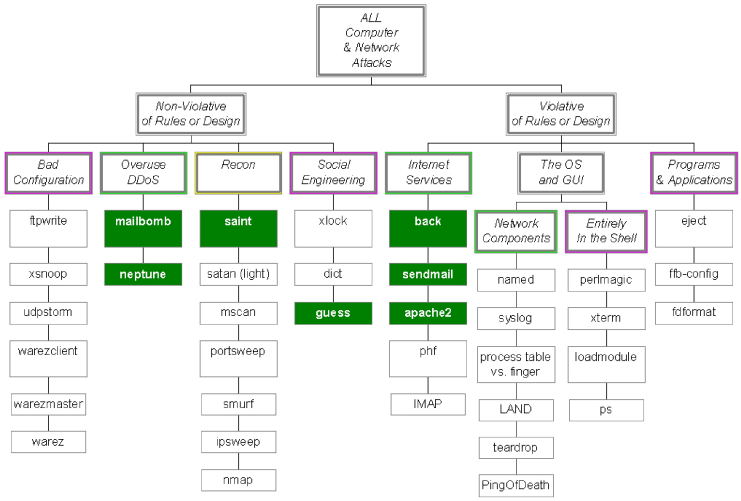

Figure 2: A Taxonomy of the Attacks in the Lincoln Labs Data

4.2 Capabilities

If an attack was detected once, it was detected each time it existed. Naturally, this is expected, as it would be highly odd for an attack to violate a specification of a protocol only part of the time. Thus attacks are either visible or invisible. However, some attacks only existed 2–3 times in the data set, and in some circumstances, detailed below, ("guess", "mailbomb") the 100% detection rate would not be anticipated in the general case if the formatting of the attacks was done by a human. A far more detailed analysis of all 36 can be found in Appendix A of the this paper, while an overview is given here.

Score	Count	Frequency	Score	Count	Frequency
0.00	121,336	80.24 %	0.45...0.54	10	0.01 %
0.01...0.04	8	0.01 %	0.55...0.64	0	0 %
0.05...0.14	52	0.03 %	0.65...0.74	0	0 %
0.15...0.24	0	0 %	0.75...0.84	0	0 %
0.25...0.34	2	0 %	0.85...0.94	0	0 %
0.35...0.44	0	0 %	0.95...0.99	2	0 %
			1.00	29,814	19.72 %

Table 1: Neural Network Output Distribution & Frequency

Those that Could Never be Detected Some of the attacks listed in the data sets are due to very poor configuration choices, such as allowing guest accounts ftp access. These are not attacks in the normal sense however, because the events that take place are actually valid under the settings that exist (even if those settings are chosen by the administrator unwittingly.) For example, if the guest account is configured to allow files to be stored in the guest directory, it is not an attack if this happens. Other attacks may be social engineering attacks, such as xlock. Here, a user has made a publicly executable version of xlock, modified to capture the target's password. When the target physically walks away from the console, the attacker runs his version of xlock, at the console. When the target returns, it appears as though xlock is running normally, and the target enters his/her password. Clearly, these are not detectable by any network-based intrusion detection scheme. The attacks that can be classed as non-attacks are seven, namely "ftpwrite", "xsnoop," "warezclient," "warezmaster," "warez," "xlock," and "dict."

Other attacks occur entirely within the operating system like "phf," "perlmagic," "xterm," "loadmodule," "ps," "eject," "ftb-config," and "fdformat." Since these are attacks which originate in the shell, and target the operating system, these are not visible to network-based intrusion detection systems. Ruling out these 8 attacks, plus the 7 from the above, 21 of them remain.

Network-Based Attacks The attacks "mailbomb," "neptune," "saint," "guess," "back," "sendmail" and "apache2" are detected by FLOWHUNT. One can see that these attacks mostly fall in the categories of either overusing or targeting an Internet service.

The attacks "named," "syslog," "ps-finger," "IMAP," and "mscan" use protocols (DNS, syslog, finger, IMAP, and DNS) other than the five selected (http, nntp, ftp, pop3, smtp), so FLOWHUNT cannot attempt to detect them. However, if a state-machine were created, and the neural-network enlarged and retrained, it is hoped they would be detectable. This assumption is based on the success of FLOWHUNT versus the seven attacks against the protocols whose state-machines are defined. Saint, the lighter-weight version of satan, happened to not use any of the five state-machined protocols in the test cases of the data set, but this would not be true in general. Also, satan was detected by FLOWHUNT, so one can be confident that saint would be visible also, if it trespassed into the five monitored protocols.

Of the 8 remaining attacks, all of them occur at the transport layer. The attacks "LAND," "portsweep," and "nmap" all use TCP, and it would be easy to develop a specification model for pure TCP based on SYNs, ACKs, FINs, et cetera... The attacks "udpstorm," and "teardrop," use UDP, which is not even fed into FLOWHUNT. Likewise neither is ICMP, with the attacks "smurf," "PingOfDeath", and "ipsweep."

4.3 Stability

In Table 1 is the distribution of neural network outputs over the week of test data, for both flagged and unflagged sessions. While this shows signs of possible over-fit, an execution on a network of actual traffic, quite large and entirely unrelated to the Lincoln Labs network, was extremely successful in producing very few alarms. One cannot tabulate detection results in that case, as it is unknown if that network was intrusion-free, and therefore any comparison of false alarm rates would be difficult.

Day:	Monday	Tuesday	Wednesday	Thursday	Friday	Total
Sessions:	40,755	45,245	47,653	52,836	42,334	228,843
False Alarms:	22	7	20	16	8	73
Invalid Traffic:	0	0	0	0	56	56
Total Unlisted:	22	7	20	16	54	129
False Alarm rate (ppm:)	540	115	419	303	189	319 ± 141
Unlisted Alarm rate (ppm:)	540	115	419	303	1559	587 ± 298

Table 2: False Alarm Rates

*This represents sessions that exist on ports 21, 25, 80, 110, 119, 8000, and 8080, namely sessions claiming to be FTP, SMTP, HTTP, NNTP, POP3 or high-port HTTP.

It is still remarkable that 99.96 % of the outputs are exactly 0.00 or 1.00. This could reflect the binary nature of the question "is this session using that protocol." Since the actual question is binary, it makes sense that the answer is binary.

Moreover, these results show that the system is not sensitive to small variations in the thresholds of alarm settings, as it would be quite rare (4×10^{-4}) for an output to be near the threshold value, commonly 0.500. This makes any potential variation of that setting (to find the optimal) an unneeded experiment.

Also note than in an ideal session, there will be four protocols with a response of 0.0, and one with a response of 1.0. This leads to the basic distribution of about 80% at 0.0 and 20% at 1.0. Moreover, an attack or non-conformal session might score 0.0 in all categories, thus 1.0 is slightly rarer than ideal, and 0.0 is slightly more common.

4.4 False Alarm Rates

Not all alarms corresponded to known and listed attacks. Upon termination of the experiments, at first all alarms that did not match attacks listed in Kendall's thesis were considered false. However, upon analysis, one was found to be clearly an instance of apache2. Therefore, an attempt was made to classify non-listed alarms as either false alarms, or invalid traffic. However, we list the original results as well.

The rate of false alarms, after correcting for invalid traffic, is 319 alarms per million sessions, which is extremely manageable in comparison to traditional IDS methods. Looking at this from the perspective of alarms per day, the network in question, which was exercising heavy traffic for the number of machines involved, would experience 14.6 false alarms per day, or one every 58.4 minutes (the traffic ran from 0900 to 2400 hours). Certainly no such small network would be, in actual use, as bombarded by such a diversity of attacks as the IDS testbed network, however this can be thought of as a model of a medium sized network of 500 users.

4.5 Speed

These experiments were performed on a Sun Microsystems Ultra 60 with the Solaris operating system. The age of the machine may have had an impact on the speed of processing. Nonetheless, 40,725 sessions in one example were processed in 2 hours and 4 minutes. (This is a sample representing one day of data.) Averaged, this is 5.47 sessions per second. Using a more modern machine, 103 to 120 sessions per second (based off of the best machines in the SpecInt95 and equivalent specs for SPEC CPU2000, in May 2004) could be anticipated.

4.6 An Alternate Configuration

As mentioned above a session is sent to each state machine for analysis. Optionally, it could be sent to the state machine corresponding to its own port number, and only upon failure, to the remainder of the

state machines. At first it would seem as though this could remove the capability to detect one protocol masquerading as another. However, it is still expected that the masqueraded session would score low in the protocol it tries to be. There were no such examples in the Lincoln Labs data set, but in manual experiments this was the case each time.

Since the vast majority of sessions are non-intrusive, given the current number of protocols (5) the speed would be five times faster[2]. This would also establish scalability in the number of protocols, where there would be no cost for additional protocols, compared to linear cost presently. (See Appendix B.) If such a change were adopted, the expected speed would then be 27 sessions per second for our Ultra 60, or 514 to 588 sessions per second for the best of the CPU2000 Spec. (Again for May 2004.)

4.7 Novel Attacks

The detection of an attack, known or otherwise, occurs when the attack transits the state of a session outside the space of allowed activities defined by the specification. It is therefore reasonable to believe that some novel attacks will be detected by a specification based system. Just as a buffer overflow against a known vulnerability requires (most likely) the transmission of data which does not fit the model of a correct protocol transaction, so should the buffer overflow against an unknown vulnerability.

Its not clear, of course, how one could test a system to see that most or even many novel attacks would be detected, but the designers of the specifications used to create the specifications (i.e. state machines) did not take knowledge of specific attacks into account, (at least not deliberately.) Certainly, the designers were not experts in the details of all the attacks in the data sets provided. Therefore its reasonable to hope that the performance of FLOWHUNT against the attacks in the data set reflects typical behavior against novel attacks in general. Without a general model of "novel attacks", which would be absurd, it is unclear how a more precise test could be executed.

Alternative Applications A short discussion on using FLOWHUNT for non-intrusion detection purposes (fault-tolerance and exfiltration detection) has been moved to Appendix C of the this paper.

5 Comparison with Sekar

Sekar's IDS, as described in [17] (published in November 2002), is network- and specification-based, like FLOWHUNT. Each host and router has a specification, given in terms of the packets it receives and transmits. The system in [17] operates on the transport layer and further down, (i.e. the network and data link layers.) Meanwhile, FLOWHUNT operates on the application layer. The impact on detection classes is critical, as some attacks are invisible at some layers, and plainly visible at others. Before embarking on the comparison, it should be duly noted that [17], also explored automated responses, specifically in the scenario of an e-mail worm/virus.[3] Sekar's paper [17] used the 1999 Lincoln Labs Data set, which included all 36 the 1998 attacks, except three of a particular form.[4] Also, 22 new attacks were added, giving a total of 55.

Attacks in Common There were 33 attacks present in both the 1998 and 1999 data sets.

- Five attacks, (Apache2, Back, Mailbomb, Neptune, Satan) are visible to both FLOWHUNT and Sekar's system.

[2]Connections on unknown ports could simply alarm directly, or could be sent to all state machines with a slight speed loss.

[3]The denial-of-service problem has been a serious challenge for Intrusion Detection systems, since it is possible to generate a very large number of valid queries, and thus create a denial-of-service, without breaking the protocol specification in any way. Their use of timing data is probably better classified as anomaly-based detection rather than specification-based, but the results are impressive.

[4]Three attacks (Warez, Warezmaster, Warezclient,) are in the 1998 set but absent from the 1999 set. These attacks were classified by our taxonomy as being "bad configuration" rather than attacks per se. Their omission from the next version of the data set tends to reinforce this conclusion, that they were not good examples of Computer Network Attack.

- Five attacks detected by Sekar, (Ipsweep, Mscan, Ping-of-Death, Smurf, Portsweep) are in protocols (ICMP, DNS, ICMP/PING, ICMP, various TCP/UDP) not present in the FLOWHUNT system, therefore it is not a surprise that FLOWHUNT fails to detect them. However, this reinforces our hopes that the FLOWHUNT-concept could detect attacks against these protocols if it were expanded to cover them.
- Three attacks, (Saint, Guess, SendMail), were visible to FLOWHUNT, but not visible to Sekar. This reflect FLOWHUNT's use of the Application layer, as these attacks are not visible at lower layers.
- There are no attacks visible to Sekar's system that FLOWHUNT cannot detect that are within FLOWHUNT's protocol set.
- The remaining 20 attacks are invisible to both systems.
- Of the 22 attacks in 1999 but not in 1998, only one attack (Queso,) was detected by Sekar. However, Queso is extremely similar to nmap, which operates at the transport layer and not the application layer.
- The false alarm rate of Sekar's system is "less than 10 per day", whereas FLOWHUNT has 14.6 false alarms per day—about the same.

Future Work A short discussion on Future Work and possible improvements has been moved to Appendix E of the this paper.

6 Conclusions

Specification-based intrusion detection has indeed proven to be a practical method of intrusion detection. The false alarm rate is outstandingly low, measured in parts-per-million. The rate of detection for attacks in the protocol set is 100%. FLOWHUNT operates at a reasonable speed, and can be a background process on most UNIX systems, or run on a small "monitor" on a LAN, without any interference with the user. Finally, the ability to detect novel attacks is an absolute advantage, that is indispensable for Defense and other users who cannot tolerate any intrusion—novel or otherwise.

References

[1] R. Bace. *Intrusion Detection.* Macmillan Technical Publishing, 2000.

[2] I. Balepin, S. Maltsev, J. Rowe, and K. Levitt. "Using Specification-Based Intrusion Detection for Automated Response." *Sixth Symposium on Recent Advances in Intrusion Detection Conference (RAID'03).* 2003.

[3] Y. Cai. "A Specification-Based Approach for Intrusion Detection." Master's Thesis, University of Iowa. Dec 1998.

[4] S. Cheung, and K. Levitt. "A Formal Specification-Based Approach, for Protecting the Domain Name System." *Proceedings of the International Conference on Dependable Systems and Networks.* 2000.

[5] R. Cunningham, R. Lippmann, D. Fried, S. Garfinkle, I. Graf, K. Kendall, S. Webster, D. Wyschogrod, and M. Zissman. "Evaluating Intrusion Detection Systems without Attacking your Friends: The 1998 DARPA Intrusion Detection Evaluation." *SANS 1999.*

[6] P. Goodman. *NevProp Software, version 3.* University of Nevada, Reno: 1996

Available at:

http://www.scs.unr.edu/nevprop

[7] J. Haines, L. Rossey, R. Lippmann, and R. Cunningham. "Extending the 1999 Evaluation." *DARPA Information Survivability Conference and Exposition, (DISCEX'01).* 2001.

[8] K. Kendall. "A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems." Master's Thesis, Massachusetts Institute of Technology, 1998.

[9] C. Ko, P. Brutch, J. Rowe, G. Tsafnat, and K. Levitt. "System Health and Intrusion Monitoring Using A Hierarchy of Constraints." *Fourth Symposium on Recent Advances in Intrusion Detection Conference (RAID'01).* 2001.

[10] C. Ko, G. Fink, and K. Levitt. "Automated Detection of Vulnerabilities in Privileged Programs by Execution Monitoring." *Proceedings of the 10th Annual Computer Security Applications Conference (ACSAC'94).* 1994.

[11] C. Ko, M. Ruschitzka, and K. Levitt. "Execution Monitoring of Security-Critical Programs in Distributed Systems: A Specification-based Approach." *Proceedings of the 1997 IEEE Symposium on Security and Privacy (SP'97).* 1997.

[12] R. Lippmann, R. Cunningham, D. Fried, I. Graf, K. Kendall, S. Webster, and M. Zissman. "Results of the DARPA 1998 Offline Intrusion Detection Evaluation." *Second Symposium on Recent Advances in Intrusion Detection Conference (RAID'99).* 1999.

[13] R. Lippmann, I. Graf, D. Wyschogrod, S. Webster, D. Weber, and S. Gorton "The 1998 DARPA/AFRL Off-Line Intrusion Detection Evaluation." *First Symposium on Recent Advances in Intrusion Detection Conference (RAID'98).* 1998.

[14] R. Sekar, Y. Cai, and M. Segal. "Specification-Based Approach for Building Survivable Systems." *Proceedings of the 21st NIST National Information Systems Security Conference (NIST-NCSC'98).* 1998.

[15] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou. "Specification-based Anomaly Detection: A New Approach for Detecting Network Intrusions." *Proceedings of the 9th ACM Conference on Computer and Communications Security, (CCS'02).* 2002.

[16] C. Tseng, P. Balasubramanyam, C. Ko, R. Limprasittiporn, J. Rowe, and K. Levitt. "A Specification-based Intrusion Detection System for AODV." *ACM Workshop on security of Ad Hoc and Sensor Networks (SASN '03.)* 2003.

[17] P. Uppuluri, and R. Sekar, "Experiences with Specification-based Intrusion Detection." *Fourth Symposium on Recent Advances in Intrusion Detection Conference (RAID'01).* 2001.

[18] 1998 DARPA/Lincoln Labs/MIT Intrusion Detection Evaluation Data Set.
Available at: http://www.ll.mit.edu/IST/ideval/data/1998/1998_data_index.html

[19] 1999 DARPA/Lincoln Labs/MIT Intrusion Detection Evaluation Data Set.
Available at: http://www.ll.mit.edu/IST/ideval/docs/1999/attackDB.html

[20] RFC959: File Transport Protocol.
Available at: http://www.w3.org/Protocols/rfc959/Overview.html

A Specific Attacks and Detections

The following is a detailed list of each attack in the data set, as well as FLOWHUNT's detection performance versus that attack, and possible reasons why.

1. Violative
   (a) Target is the OS/windowing system.
      i. **The Target is a Network Service of the OS/Windowing System.** This is a common target, and 6 out of 35 of the attacks in the LL data set fall into this category. Given that the protocols are in a specification-based system, all these attacks should be discoverable.

      "named": No, because this uses DNS, and DNS is not in the system yet.

      "syslog": No, because this uses a service that is not one of the 5 protocols used.

      "process table": No, because this implementation uses finger as its "target", but if another protocol had been chosen from the 5 used, it would be detected.

      "land": No, because this attack occurs entirely at the transport layer, by sending a SYN packet with identical source and destination IP addresses.

      "teardrop": No, because it uses UDP.

      "Pingofdeath": No, because it uses ICMP.

      ii. **The Target is not a Networking Component of the OS/Windowing System.** This also implies that the attack is entirely in the shell, or the file system. Any network-based system, whether specification, or anomaly based, will be unable to detect these attacks. Signature based systems will also only detect the most un-stealthy implementations of these attacks, because their signature can be masked by trivial scrambling. Four of the 35 Attacks are in this category.
      Any network-based IDS cannot see these attacks. These attacks start, work and end entirely in the shell. Therefore, they are entirely invisible, except possibly that their signatures could be detected in transit. However, this is trivially circumvented with encryption. Therefore no network-based IDS can defend against these.

      "Perlmagic"

      "xterm"

      "loadmodule"

      "ps"

   (b) **Target is an Application Program (not network based).** This is very similar to Operating System/Windowing System attacks that are entirely in the shell. No network-based system on can detect these attacks if they are performed with any scrambling or obfuscation.

      Any network-based IDS cannot see these attacks. See previous.

      "fdformat"

      "eject"

      "ffb-config"

   (c) **The Target is a Network Service** Naturally, these attacks are ideal for network-based IDSs. The attacks should be identifiable, given that the protocols involved are in the IDS. This is also an important category, with 5 of the 35 attacks.

      "back" Yes. The attack consists of sending an http get request with far too many backslashes, which crashes the system.

      "sendmail" Yes. While this is a DoS, and *could* be done within the confines of the protocol, the traffic is not quite exactly crafted and an alarm occurs.

"apache2" Yes. This is a series of http requests to an apache server with far too many http headers in the request packet. This causes the server to crash, though the RFC of the protocol does not specify a limit.

"phf" No. This attack is against the CGI script running on the web server. Analysis cannot be done on that level unless a specification were available for all the CGI scripts on the target network.

"IMAP" No, because this uses the IMAP protocol, not one of the five protocols in the system.

2. Non-Violative

   (a) Bad Configuration (making the attack legal)

   "ftp-write" Here the system administrator has made the ftp guest directory world-writable. The hacker simply writes a .hosts file, and connects again, and can execute shell commands. This is perfectly valid activity, given the unfortunate choices by the system administrator, and so cannot be detected.

   "warez-client, warez, warez-master" Here the user has made the guest directory of an FTP server open (world-writable), and several users are depositing and removing their files. Again, since the attack is not illegal in the protocol, a specification-based IDS is unable to detect this.

   "xsnoop" The attacker connects to the user's XServer and requests to be informed of all KeyPress events. This is not usually possible unless the permissions are configured to allow it, or have a default of allowing connections from all hosts—in which case this is perfectly valid activity given the configuration.

   "udpstorm" This attack is the author's favorite. A forged packet is sent with the victim as both sender and receiver. The one port is "echo" which repeats back what is sent to it, and the other is "chargen" repeats back a random set of characters. Thus packets bounce between these ports continuously, causing a denial-of-service. These are correct uses of these protocols, so the specification-based class of IDSs would be unable to see them. Note, there is no practical use of these protocols, and the ports should be closed on any well-configured machine. However, one could specifically forbid these protocols by not including them in the IDS, and then they could be detected (as an unknown protocol alarm.)

   (b) Social Engineering

   "guess" This attack is a repeated attempt at guessing the "guest" FTP password. The packets of this attack were not well-formed, and thus were detected by FLOWHUNT. This at first appears to be a flaw in the data set, and thus not a useful detection. However, unless a malicious user manually guesses the passwords, it is reasonable to conclude they will write a script to automate the process and generate the needed packets. There is a probability that the packets will not be forged exactly to spec, and therefore, those also would be detected by FLOWHUNT. Also note that the responses to a bad password guess are considered "error conditions" and are not in the "good" bucket, so the score for this session will not be perfect.

   "dict" In this attack, guesses were made against the `telnet`, POP3, and FTP passwords of users. In this case, the guesses were well-formed, and no detection occurred. As above, the bad guesses are considered error conditions, and the scores will not be perfect.

   "xlock" In this attack, a malicious user waits until the target physically leaves a machine, while logged-in. The attacker has a fake version of xlock on their own machine, in a world-readable-executable file.

   This program is custom-written, and appears to be xlock but instead forwards the password of the victim back to the malicious user. The attacker runs their fake version of xlock from the victim's keyboard, and when the victim returns, he/she types in their password. This attack occurs at the "person" level and cannot be detected by an Intrusion Detection System, short of User Profiling or biometrics.

(c) Reconnaissance & Denial of Service

"nmap" The ubiquitous NMAP is a classic tool that maps target networks, by among other things, sending pings and malformed TCP packets. This all takes place primarily at the transport layer, and thus is invisible to FLOWHUNT in its current form. However, this is strong motivation to include transport layer analysis in FLOWHUNT.

"satan" Satan is a tool designed to probe a target via various grades of bombardment with packets to test vulnerabilities. The setting used in this example was "light" and so by coincidence, no HTTP, FTP, POP, NNTP or SMTP, packets were used. However, if other settings were used, it is likely this attack would have been detected.

"saint" Saint is a cleaner, newer and more friendly version of satan, but is essentially the same concept. In this case, several packets of the 5 protocols we target were used, and therefore the attack was detected by FLOWHUNT. This represents a more aggressive setting.

"neptune" The neptune attack floods a machine with SYN TCP packets, which leaves a large number of half-open sockets. Eventually, the target machine runs out of sockets. This cannot be detected at any level other than the transport layer, and so is currently undetectable but is strong motivation to add transport layer capability to FLOWHUNT.

"portsweep" This is actually a collection of scans of various methods using TCP and UDP to find open ports on a target machine. This is further motivation to expand to the transport layer.

"mailbomb" Much like "guess," this script crafted traffic automatically, rather than through the usual methods. It is a long series of rapid sendmail commands, designed to overwhelm the server, and thus be a Denial-of-Service attack. The packets were not well-formed, and the attack was detected.

"smurf" The smurf attack is a distributed denial of service which floods the target machine with ICMP packets. Since TCP is not involved, FLOWHUNT cannot detect these attacks.

"ipsweep" This also uses ICMP, to see what machines exist and what machines do not, by their responses to pings. Again, TCP is not involved so FLOWHUNT cannot operate.

"mscan" This attack uses DNS, and so is not visible to the FLOWHUNT system.

(d) Other– No other attack falls in this category but the other three of "Protocol Abusing" are not collectively exhaustive.

B Scalability

If sessions are kept and scored only upon completion, and if no session-to-session correlation needs to be known or determined (which in this case is true), it can be concluded that scoring one session is entirely independent of scoring another. This makes the system entirely scalable.

More specifically, if a very high bandwidth link creates far more sessions than can be analyzed by a single system, then sessions could be dispatched from a queue to any number of systems for analysis. The only requirement is that the sessions be stored as fast as they are incoming.

On the other hand, there may be times when such communication or correlation between sessions is desired. (For example, some sort of "system-wide alarm level" or detecting DoS by noticing repeated packets from the same source.) This still is possible if the assumption is made that packets with similar IP address (at least in the network portion) are related.

The proposed splitting pattern, if such correlations were desired, is as follows. Each session is duplicated, and sent to a machine/process based on the first two octets of its IP address, both sender and receiver. Thus if two senders or if two receivers are on the same Class B, or Class C, or $1/256^{th}$ subnet (or smaller) of a Class A, then they will be processed on the same machine/process. (Naturally this produces two copies of each packet, one at the receiver's address's machine, and one at the sender.)

Thus, any information gained in the analysis of one session of such a pair would be immediately available, without delay, for the next session of such a pair. There need not be any high speed interprocess communication.

Also note that such a splitting scheme would allow for between 1 and 65,536 processing nodes. Adjusting for the duplication of effort for duplicating each session, this can be a speed up of 32,768, if desired, over one process/machine.

C Alternative Applications

While FLOWHUNT was designed as an Intrusion Detection system, it has other applications in the security arena, as outlined below.

C.1 Fault Tolerance

FLOWHUNT can also be used as a fault-detection tool, to detect improper or odd, behavior of one party in network communication. These errors can be due to reasons other than attack. For example, during testing on a lab network which included a Windows machine, Internet Explorer crashed, which was immediately preceded by sending strange data to the intranet web server on Port 80. This generated a FLOWHUNT alarm.

C.2 Exfiltration Detection

Once an attack is complete, the attacker may wish to extract data from the target system. This can be done overtly, such as reading a data file in `emacs` over `telnet`, or FTPing data out. However, exfiltration is likely to be stealthy, particularly if the attacker is concerned with network monitoring or logging.

Not all exfiltration is due to attackers. Various programs such as Napster (in its era), Kazaa and other file sharing systems can be detected operating unauthorized in business (or even Defense) networks. These systems, and others such as instant messaging protocols, will sometimes masquerade on Port 80.

Finally, it should be noted that if the exfiltration from a network is mitigated, the intelligence gathering impact of very difficult-to-detect "easter eggs"[5] is also mitigated. Of course, this does not address easter egg ability to trigger denial-of-service or other undesirable events.

D A History of FLOWHUNT

This project began as a packet-verifier tool or PV, designed in the period 1996–1998, by various members of the Defensive Information Operations team of the Information Security Research Office of the National Security Agency. A stream of sniffed packets would be passed into verifiers, which would determine if the packet conformed to the definitions of the data-link, network, transport and application layers for the particular protocols involved. While originally intended for off-line log analysis, PV was made on-line (real time) as computational power of the typical desktop machine increased. It could be used to highlight the precise packets relating to an intrusion detected by other programs. Also it could be used to ensure that network software was operating correctly within the parameters of the defined protocols.

This set of tools was very sucessful. The natural next step was to convert PV (now renamed PEAVEY) into an Intrusion Detection System of its own. More specifically, if one chooses to define an intrusion as a violation of protocol specifications, then it is natural to verify compliance with those specifications as a means of detecting intrusions. This research effort undertaken by a senior neural networks expert in the Defensive Information Operations (DIO) branch of the NSA Secure Systems Research Office (in the Research Directorate) in 1999 and 2000.

[5] Malicious software with deliberately coded flaws, to enable later exploitation and exfiltration remotely.

Having been laid aside for a year, this project was inherited by Gregory Bard in the summer of 2001, and renamed FLOWHUNT when applied to real world data and brought into development as a proof-of-concept tool. The experiments in this paper took place in the fall of that year. The rather severe international events in September of 2001 contributed to the delay in publication, and eventually lead to the cancelation of this project.

This work was all done at taxpayer expense, and abandoned. The interested reader, regardless of citizenship, is encouraged and entitled to request documents relating to FLOWHUNT from the NSA via the Freedom of Information Act.

E Future Work

The immediate opportunities for improvement are the following:

- Expand the IDS to the transport layer by making a general TCP state-machine. This will eliminate many attacks from the list of invisibles, by detecting violations of sequence numbering, flag usage and other odd behaviors. This work would make the following attacks potentially visible: "land", "nmap", many parts of "satan/saint", "neptune", and "portsweep." An excellent scheme is recommended in [17].

- Expand the suite of available protocols to include other common applications (SSH, `telnet`, IMAP, etc. . .)

- Add the ability to examine UDP usage. Some concept of "session" must be artificially created here, because UDP is connectionless, but perhaps a conversation between two machines within a particular bound of time might be a "pseudo-session." This would attempt to detect "udpstorm", and "teardrop", as well as parts of "satan" and "saint."

- A state machine for ICMP would be also useful. Here no real concept of session exists, except perhaps ping and response. Therefore a session would be defined to be a single packet by itself, possibly with a response if one exists within a certain time-limit. But, still there could be detections made, and perhaps tabulations kept to detect scans. This would attempt to detect "pingofdeath", "smurf", and "ipsweep," as well as parts of "satan", and "saint."

- An aggregation method or system is needed for alarms. If a denial-of-service attack occurs, such as a mailbomb, then a flood of alarms occurs at the IDS node. This would be highly annoying to the watch-person, as well as a potential DOS to the IDS itself.

- It would be an interesting experiment to attach a Signature-Based IDS such as SNORT to this system as an adviser. If a session comes back as alarming, the notification would include an addition string, which would be some response from SNORT about the nature of the attack. Hopefully our system can detect some attacks not in SNORT (otherwise why build it?) but the nomenclature and signature library already developed by the SNORT community should be leveraged.