An Analysis of the 1999 DARPA/Lincoln Laboratories Evaluation Data for Network Anomaly Detection

Matthew V. Mahoney and Philip K. Chan
Dept. of Computer Sciences Technical Report CS-2003-02
Florida Institute of Technology
Melbourne, Florida 32901
{mmahoney, pkc}@cs.fit.edu

ABSTRACT

We investigate potential simulation artifacts and their effects on the evaluation of network anomaly detection systems in the 1999 DARPA/MIT Lincoln Labs off-line intrusion detection evaluation data set. A statistical comparison of the simulated background and training traffic with real traffic collected from a university departmental server suggests the presence of artifacts that could allow a network anomaly detection system to detect some novel intrusions based on idiosyncrasies of the underlying implementation of the simulation, with an artificially low false alarm rate. The evaluation problem can be mitigated by mixing real traffic into the simulation. We compare three anomaly detection algorithms, SPADE, PHAD, and LERAD, on simulated and mixed traffic. On mixed traffic they detect fewer attacks, but the explanations for these detections are more plausible.

1. INTRODUCTION

The DARPA/MIT Lincoln Labs (LL) off-line intrusion detection evaluation data set [17] is probably the most widely used set for developing and testing intrusion detection systems. Prior to the development of the two sets in 1998 and 1999, researchers had to laboriously construct their own tests using exploits against live targets under controlled conditions. Such results were normally not reproducible because the background data (network traffic, audit logs, etc.) would contain private data that could not be released. The LL data, which simulates a network with multiple targets under attack by published exploits, solves the privacy problem by simulating the background network traffic using a mix of public and synthesized data and custom software to make it appear as if the traffic originates from a much larger collection of hosts and users.

The LL data is useful because it allows a variety of methods to be tested against a wide range of exploits and targets. The 1999 set alone consists of 5 weeks of network traffic, audit logs, system call (BSM) logs, nightly file system dumps, and directory listings collected from targets running four different operating systems (Solaris, SunOS, Linux, and Windows NT) attacked 244 times using variations of 58 published exploits ranging from probes and denial of service attacks to root shell exploits and backdoors. Systems can detect these attacks using either host based or network based methods, and either signature detection (modeling known attacks) or anomaly detection (modeling normal behavior to detect new attacks). These sets have been used to test a number of important systems [4, 6, 7, 12, 16, 18, 20, 21, 22, 24, 30, 32, 33, 34, 35, 36, 37]. In addition, the 1998 network traffic is the basis of the 1999 KDD cup machine learning competition [5], which had 25 participants, and it continues to be used to test intrusion detection methods [28, 38].

We are concerned with the realism of the background Internet traffic, which is important to the proper evaluation of network anomaly detection systems which use this traffic to model normal (non-hostile) behavior. Internet traffic is vastly complex, and very difficult to simulate properly [10]. Some post-evaluation work on the 1999 LL data suggests the presence of simulation artifacts that allow hostile traffic to be distinguished based on idiosyncrasies of the simulation. For example, during development of PHAD [20], it was discovered that many attacks can be detected simply because the packets have a TTL value of 253. It was concluded that this was probably due to the network configuration used in the simulation. A similar problem occurred with the TCP window size field in NETAD [18].

The most serious questions are raised by the large number of attacks detected by anomalous source (remote) addresses by NETAD, and two other systems, ALAD [22] and LERAD [21]. Source addresses are monitored by most network intrusion detection systems, and form the basis of firewall policy, along with destination addresses and port numbers. An anomaly would seem to indicate a novel user, which for a password protected service such as SSH or telnet, would be considered suspicious. However most of the attacks detected by these systems in the LL data are on public services: HTTP, SMTP, DNS, and anonymous FTP. This does not make sense.

McHugh [23] criticized the LL data because of the unrealistically high rate of malicious attacks, a compromise needed to keep the data set to a reasonable size. Although some have suspected there are problems with the LL background traffic (as suggested by anonymous reviewers of our other papers), we are not aware of a detailed analysis of this data.

The rest of this paper is organized as follows. In Section 2 we review the LL evaluation and related work in network anomaly detection, traffic collection and synthesis. In Section 3 we introduce a simple anomaly detection system called SAD, intended to motivate our approach to finding and correcting artifacts. In Section 4, we statistically compare the LL background data with some real traffic, which we find to be less predictable and "dirtier" at many protocol levels. In Section 5 we propose a strategy for making the LL evaluation more realistic by mixing real traffic into the simulated off-line data. In Section 6 we test the mixed data on three systems, SPADE [13], PHAD, and LERAD, and show that most of the questionable detections are removed. In Section 7, we conclude.

2. RELATED WORK

Our goal is to examine differences in the LL (simulated) and real background traffic from the point of view of network anomaly detection systems. We review some of those systems, how the LL

evaluation was conducted, and problems with collecting or synthesizing background traffic.

2.1. The 1999 LL Evaluation

The 1998 and 1999 LL data sets were originally used in blind evaluations before the data was released. In 1999, eight groups participated, submitting 18 different systems. Participants were given 3 of the 5 weeks worth of data in advance in order to develop their systems. One week (week 2) had 43 labeled instances of 18 of the 58 test attacks. The other two weeks (weeks 1 and 3) were attack-free background traffic which could be used to train anomaly detection systems. The data included traffic from two sniffers (inside and outside the Internet router), audit logs, nightly file system dumps and directory listings, and Solaris system call traces (BSM).

Four months later, the developers were given the two weeks of test data (weeks 4 and 5) and asked to use their systems to identify the 201 attacks, some of which were modified to be stealthy to defeat signature detection systems (e.g. by slowing down a port scan or disguising suspicious commands). Participants had to identify each attack by time (within 60 seconds) and target IP address, and report a numeric score or ranking indicating a confidence level in the alarm. Systems were evaluated by the number of attacks detected out of the number of in-spec attacks (those they are designed to detect based on the data they examine) at a scoring threshold allowing 100 false alarms. The best systems used a combination of methods, both host and network based, and both signature detection (modeling known attacks) and anomaly detection (modeling normal behavior to detect new attacks). The top four systems [17] detected 40% to 55% of those attacks, as shown in Table 1.

System	In-spec attacks	Detected at 100 FA
Expert 1	169	85 (50%)
Expert 2	173	81 (47%)
Dmine	102	41 (40%)
Forensics	27	15 (55%)

Table 1. Top 4 results from the 1999 LL evaluation [17].

2.2. Traffic Collection and Synthesis

The LL developers put much effort into making the background traffic appear as realistic as possible. Traffic was generated using custom software running on a small number of hosts to emulate hundreds of programmers, secretaries, managers, and other types of users running common UNIX or Windows applications on thousands of hosts and websites. The distribution of services (web, mail, telnet, FTP, etc.) was matched to the actual distribution measured on a small Air Force base network in 1998. Traffic rates are highest during normal working hours, as with real traffic. Email messages were taken from public mailing lists or synthesized using English word bigram statistics. Some traffic that was too complex to synthesize was generated manually.

Nevertheless, it is extremely difficult to simulate the Internet accurately. The difficulty comes not just from modeling human behavior, but also from modeling thousands of versions of various operating systems, servers, and clients. Much of the "crud" found in real traffic [25] could originate from faulty versions of this software. For example, in real traffic that we collected on a university departmental server, we have found reserved TCP flags set, fragmented IP packets with the "don't fragment" flag set, TCP retransmissions with inconsistent payloads, invalid application protocol commands, undocumented protocols, and so on.

A solution to this problem would be to collect real traffic rather than synthesize it. However, this raises privacy issues. Public collections of real traffic such as the Internet Traffic Archive [27], and the University of New Mexico data set [10], are stripped of much of the data useful to anomaly detection, including all of the application payload data.

2.3. Network Anomaly Detection

Many older network anomaly detection systems, such as those surveyed by [3], are based on modeling normal user behavior to detect unauthorized users. To that end, these systems model features that are controlled by the user, such as the destination IP address and service (port number), and attributes that identify the user, e.g. the source address. Unusual combinations of these attributes are often hostile. For example, attempted accesses to nonexistent ports might signal a port scan. An unusual source address connecting to an authenticated service such as SSH might indicate a password guessing attack. ADAM [31], NIDES [2], eBayes [35], and SPADE are all systems that model addresses and ports. For example, one mode of SPADE assigns an anomaly score to inbound TCP SYN packets of 1/P(source address, destination address, destination port), i.e. higher scores to packets with combinations of these three attributes that appear less frequently.

User modeling differs from host based anomaly detection systems that model *program* behavior to detect when a server or operating system component has been compromised. Forrest [9] showed that such systems make unusual sequences of system calls when attacked. This can happen in a buffer overflow attack where the compromised program is executing code supplied by the attacker, but another possibility is that the attacker is exploiting a vulnerability in a rarely used feature of the program. Because vulnerabilities are bugs, they are most likely to occur in poorly tested code. Thus, another approach to anomaly detection is to look for unusual inputs to a program that would invoke seldom used features. For example, attacks such as *teardrop* and *pod* (ping of death) exploit errors in IP packet reassembly code that cause the target to crash when it receives fragmented IP packets that cannot be reassembled properly. Because IP fragmentation is rare, an anomaly detector that flags all fragmented traffic, whether legitimate or not, is likely to detect these attacks, albeit at the cost of some false alarms.

Systems like PHAD and LERAD are program modelers. They differ from user modelers in two respects. First, they greatly extend the number of attributes monitored in order to cover many of the protocols that might be exploited. PHAD (Packet Header Anomaly Detector), which monitors both inbound and outbound packets, models 34 fields in the Ethernet, IP, TCP, UDP, and ICMP headers. LERAD (LEarning Rules for Anomaly Detection), which monitors inbound client to server TCP streams, models addresses, ports, length, duration, opening and closing TCP flags, and the first 8 words in the application payload.

Second, in order to cope with the bursty nature of network traffic over a wide range of time scales [15, 26], these systems use a time-based model rather than a frequency-based model. In a frequency based model such as SPADE, the probability $P(A = v)$ that attribute A has value v is estimated by n_v/n, where n_v is the number of times v is observed, and n is the total number of observations. The assumption is that rare events are more likely to

be hostile. Thus, these systems assign an anomaly score of $1/P(A=v) = n/n_v$ or something similar.

PHAD, ALAD, LERAD, and NETAD model *novel* events, where $n_v = 0$ in training. They use the anomaly score tn/r where t is the time since a novel value was last observed, n is the total number of observations, and r is the number of anomalies that occur in training, i.e. the size of the set of allowed values. For instance, given the training sequence *ababc*, there are $n = 5$ observations and $r = 3$ allowed values (*a*, *b*, and *c*). Given the test sequence *aadad*, only the two *d*'s would generate anomalies. (We assume separate training and test phases, although the system could remain in training mode at all times like SPADE). The last anomaly before the first *d* is *c*, so $t = 3$. The last anomaly before the second *d* is the first one, so $t = 2$. Thus, the first anomaly score would be 3*5/3 = 5, and the second would be 2*5/3 = 3.33. If the sum of anomaly scores over all attributes exceeds a threshold, then an alarm is signaled.

Note that tn/r has the form of an inverse probability $1/P$. First, the average rate of anomalies in training is r/n, thus this model assumes that this rate will continue. However, because the set of allowed values is not allowed to grow after training, we need another term to discount repeat occurrences of the same anomaly. This term is t. This is a good model of bursty behavior, and also quite intuitive. The assumption is that the probability of an event is inversely proportional to the time since it last occurred.

3. SIMULATION ARTIFACT DETECTION

To motivate our investigation of simulation artifacts in the LL data, we develop a very simple anomaly detection system that could not possibly work. We call our system SAD (Simple Anomaly Detector, available at [19]). SAD examines only inbound TCP SYN network packets (destination address 172.16.x.x). It looks at just one byte of the packet, specified as a parameter, for example the TTL field (time to live – an 8-bit counter used to expire packets caught in misconfigured router loops). During training, SAD records which of the 256 possible values are seen at least once and which are not. During testing, it detects an anomaly if this byte has a value that was never seen in training. If there have been no other anomalies in the last 60 seconds, then it outputs an alarm with a score of 1 warning that the packet is hostile.

We train SAD on the inside sniffer traffic from week 1 and test it on week 2, which contains 43 attacks. This data (with truth labels) was available in advance to the original participants in the 1999 evaluation. We evaluate SAD (using EVAL [19], our implementation of the LL detection criteria) and identify several promising SAD variations, which we define as any variation that detects at least one attack for every 10 false alarms (Table 2, second column). Then we evaluate these variants on the actual test data by training them on inside week 3, and testing on weeks 4 and 5, which contain 177 of the 201 attacks used in the published evaluation results. Almost all of these variations would have done well in this evaluation (Table 2, third column). The best SAD variant, which examines the third byte of the source IP address, detects 79 of 177 attacks (45%), with 43 false alarms. This result is competitive with the top systems in the original evaluation.

However, these results are misleading. To test how SAD might behave in a real network, we mix the 146 hours of training traffic from week 3 and 197 hours of test traffic from weeks 4-5 with equal durations of (presumably attack-free) traffic collected from a university departmental server on a large network. We mix the traffic by shifting the timestamps to make it appear as if the web server is part of the home network. No other fields (e.g. IP addresses) are changed. The mixed traffic contains 154,057 simulated and 125,834 real inbound TCP SYN packets.

As we should expect, the results are quite poor (Table 2, last column). Very few attacks are detected, and the false alarm rate is much higher. But a more detailed analysis shows that these results make more sense. For example, on the simulated data SAD detects source address anomalies in attacks on public web servers (*apache2*, *back*, *crashiis*, and *phf*), mail servers *(mailbomb*, *sendmail*),. DNS (*ls_domain*), and anonymous FTP (*guessftp*, *warez*), where novel addresses should be normal. However, on the mixed data, the only attack detected is *neptune*, which spoofs the source address with an unassigned portion of the IP address space (10 or 11 in the first byte). Likewise, most of the other packet header anomalies detect only attacks that require the attacker to write (usually arbitrary) values into those fields.

But why did SAD do so well in the first place? In the next section, we compare the simulated training traffic with our real traffic to shed some light on this question.

SAD Byte Det/FA	Wks 1-2	Wks 3-5	Mixed
IP packet size, low byte	4/0	15/2	0/1
TTL	25/36	24/4	5/43
Source IP address, 1st byte	13/7	64/41	4/0
Source IP address, 2nd byte	13/7	67/42	0/0
Source IP address, 3rd byte	16/15	79/43	0/0
Source IP address, 4th byte	17/14	71/16	0/0
Source port, high byte	2/0	13/0	0/0
Destination port, high byte	4/24	4/0	4/1664
Destination port, low byte	5/6	0/0	0/0
TCP header size	4/0	15/2	0/5
TCP window size high byte	5/1	15/2	7/112
TCP window size, low byte	3/1	7/1	4/29
TCP options, bytes 1, 2, 3	4/4	15/2	0/1
TCP options, byte 4	4/4	15/2	0/255

Table 2. SAD detections and false alarms (Det/FA) for variants that do well on the 43 attacks in weeks 1-2 of the 1999 LL IDS evaluation inside sniffer traffic. Detections and false alarms are shown for weeks 1-2 (43 attacks), weeks 3-5 (177 attacks) and for weeks 3-5 mixed with real traffic.

4. SIMULATED VS. REAL TRAFFIC

In this section, we compare the LL training data (inside sniffer weeks 1 and 3) with real traffic collected from a similar environment, a university Solaris machine which is the main server for the CS department, with several faculty user accounts and serving several thousand web pages. We look for differences in the distributions of attributes that an anomaly detection system might monitor. We examine many traffic types, but especially the types most commonly exploited: inbound TCP client to server traffic.

We are primarily interested in differences that could lead to evaluation errors in typical anomaly detection systems. One type of error could occur if values that appear only in hostile traffic in simulation actually occur in normal, benign traffic. This would cause those attacks to be missed. Another type of error could occur if rare or novel values occur at a higher rate in real traffic because there are a wider range of values. This would either lead

to a higher false alarm rate, or possibly some adaptive process to turn off the rule, resulting in a lower detection rate.

We collected two sets of data from the server, a small set consisting of two traces of one million packets each (several hours), and a larger set of 100 million packets sampled over 10 weeks. The large set was filtered to allow quicker analysis by extracting only the 1.6 million most interesting packets: truncated inbound client to server sessions.

4.1. Analysis of Packets and Protocols

The simulated and real networks are similar in that there are two routers, one to a large local network with hundreds of hosts, and a second to the Internet. However, the real network differs in that it uses an Ethernet switch rather than a hub, so only traffic to and from the local host is visible. Also, our data was collected 2-3 years after the LL data was synthesized, during which time some new protocols probably came into use and others were expanded. There are some dynamically assigned IP addresses, and a portmapper service to assign ports for RPC and NFS, which are not found in the simulation. The only TCP application protocols that are found in sufficient quantity to allow comparison are HTTP, SMTP, and SSH. Some traffic is blocked by a firewall, such as unreachable ports and IP packets with options. Although this probably blocked most port scans (there were probably many), we did find some apparently malicious HTTP and SMTP traffic in the real data.

The traffic was collected on www.cs.fit.edu. The smaller set consists of two traces of 1 million packets each, collected on Nov. 4, 2001 from 17:40 to 04:43 local (Eastern) time the following day (11 hours), and Nov. 6, 2001 from 10:25 to 12:05 (100 minutes). Packets were truncated to 68 bytes. Table 3 compares the distribution of protocols with that of inside week 3 in the LL set. In general, the real traffic is more complex in that there are more protocols present at every level. At the transport layer there is somewhat more TCP and ICMP and less UDP, but all three are represented in sufficient quantities to allow comparison (and later, mixing).

Attribute	LL inside week 3	Real, unfiltered set
Packets, n	12,814,738	2,000,000
Ethernet. protocols	4 (IPv4, ARP, hub test, loopback)	45 (many undocumented)
IP protocols	3 (TCP, UDP, ICMP)	6 (also OSPFIGP, IGMP, PIM)
IP packets	99.2% of Ethernet	94.8% of Ethernet
TCP packets	83.4% of IP	94.6% of IP
UDP packets	16.4% of IP	3.5% of IP
ICMP packets	0.056% of IP	0.268% of IP
TCP protocols (in order of descending session frequency)	HTTP, SMTP, FTP, telnet, ssh, finger, auth, epmap, printer	HTTP, printer, POP3, NFS, SMTP, RMI, IMAP, nbsession, 42239, ssh, auth, dsp, 4045, X-font, portmap
UDP protocols (by descending packet frequency)	DNS, NTP, router, nbname, nbdatagram, syslog	756, NTP, portmap, DNS, syslog, nbdatagram, nbname, isakmp, xdmcp

Table 3. Protocols found in LL inside sniffer week 3 and in the unfiltered real data set.

4.2. Analysis of Fields

In this section, we compare the simulated training data from the inside sniffer weeks 1 and 3 with the larger set of real traffic collected over 10 weeks. To reduce the workload of analyzing these huge data sets, we filtered them to extract just the data that a network intrusion detection system would most likely monitor: the initial portions of inbound client to server requests.

4.2.1. Data Set

Most of our analyses are based on the large sample of 100 million packets. This consists 50 traces of 2 million packets each collected on Monday through Friday over 10 weeks from Sept. 30 through Oct. 25 and Nov. 4 through Dec. 13, 2002. Each trace was started at 00:01 local time and ended when 2 million packets were collected, usually about 10 to 15 hours later. Packets were truncated to 200 bytes (134-146 bytes of TCP payload).

To reduce the volume of data to a manageable level, the large sample set was filtered. This filter removes the following data, leaving 1,663,608 packets (1.6%).

- All non-IP packets.
- All outbound packets. A packet is inbound if the destination address is 172.16.x.x or 192.168.x.x (simulated eyrie.af.mil) or exactly the IP address of the real server (163.118.135.1).
- UDP packets to high numbered ports (over 1023), which are normally server responses back to clients.
- TCP ACK, SYN-ACK, FIN-ACK, and PSH-ACK packets unless within the first 100 payload bytes of a SYN packet (i.e. only the first 100 bytes of an inbound client request are passed, and none of the server's response).
- Any packet where more than 16 have been passed to the same IP address/port/protocol (TCP/UDP/ICMP) combination in the last 60 seconds. A 4K hash table is used, so there are a small number of drops due to collisions.

Most of the data of interest remains present after filtering. Filtering both weeks 1 and 3 of the simulated inside sniffer traffic reduces this set from about 20M packets to about 1.6M packets.

To compare application protocols, we reassemble TCP streams from the filtered traffic. Because these packets are truncated, we use only the first 134 bytes of the first payload packet. For interactive protocols such as SMTP, this method only allows the first inbound command to be captured. However, this is sufficient for our analysis.

4.2.2. Measurements

We are primarily interested in the rate of novel values in each attribute that an anomaly detection system might monitor. The higher this rate, the higher the false alarm rate will be, and the greater the chance that a genuine anomaly will later be masked. We define the following four statistics:

- r – the number of observed values.
- r_1 – the fraction of r consisting of values seen exactly once.
- r_h – the fraction of values seen for the first time in the second half of the data.
- r_t – the fraction of data needed for r to reach half its final value.

For example, given the sequence ABCABD, $r = 4$ (the size of the set {A,B,C,D}), $r_1 = 2/4 = 0.5$ (C and D occur once), $r_h = 1/4$ (only D is seen for the first time in the second half), and $r_t = 2/6$

(because we observe $r/2 = 2$ letters in the first 2/6 of the sequence).

The statistic r is significant because it directly counts novel events in the training data (which would be false alarms if they occurred during testing), and is also used to compute the anomaly score in time-based systems (i.e. tn/r). For Poisson processes (where events are independent), r_1 is a Good-Turing [11] estimate of the probability that the next value will be novel. For network processes which are bursty with long range dependencies, r_1 is usually an underestimate. However r_h and r_t measure the rate of novel values directly, either over the second half of the data (r_h) or the second half of the novel values (r_t). Because there are gaps in the data collection, we use packet counts rather than real time to compute the fraction of data seen.

The r_h and r_t statistics give us two points on the growth curve of r over time. For many attributes, r will grow rapidly at first, and then level off as all of the possible values are observed. We are interested in both the initial growth rate, given by r_t, and the recent growth rate, given by r_h. Often r_h will be 0, so we need r_t to make meaningful comparisons in cases of slowly growing r.

If an attribute has a Zipf distribution [39], then $r_1 = r_h = r_t = 0.5$, and r grows without bound at a constant rate. A Zipf distribution is a special case of a power law or Pareto distribution, which occurs in many natural processes, for example, the distribution of words in English, city populations, file sizes, or website requests [1, 14]. We find that many network attributes in the real traffic, but not in the simulated traffic, are approximately Zipf, for example, client addresses and client versions.

For binary attributes, we list the percentage of occurrences. We consider it significant if an event occurs at any rate in one set but never in the other. For continuous attributes we list the range of values, although it is unclear when a larger range becomes significant.

4.2.3. Comparison at Low Level Protocols

We first compare the training traffic (inside sniffer weeks 1 and 3) with the large, 10 week data set, both after filtering. In most of the attributes we examined, the rate of anomalies is higher in the real traffic, as indicated by higher values of r, r_1, r_h and r_t (listed as four consecutive values in Table 4a), even after taking into account the larger size of the real data set. Where the difference is significant (a somewhat subjective judgment), the higher values are highlighted in italics. These fields include the Ethernet source address, TTL, TOS, TCP options, UDP destination port, and ICMP type.

The following binary events occur only in the real traffic: fragmented IP packets (with the "don't fragment" flag set), TCP and ICMP checksum errors, nonzero bits in TCP reserved fields and reserved flags, and nonzero data in the urgent pointer when the URG flag is not set. These events are present even after removing TCP packets with bad checksums.

For all continuous attributes we measured, the range is higher in real traffic. This includes packet size, UDP payload size, TCP header size, urgent pointer, and window size. However it is difficult to judge the significance of these differences based on range alone.

Most attributes are less predictable in real traffic than in simulation. However the situation is opposite for TCP ports. The rate of novel values is lower in the real traffic. Most of the simulated TCP ports are high numbered FTP data ports negotiated during FTP sessions. The real traffic has a much lower rate of FTP sessions. Also, some real ports may be blocked by the firewall.

r, r_1, r_t, r_h	**Simulated**	**Real**
Ethernet source addr	8, 0, 0, .00001	*76, .01, .11, .03*
IP source addr	1023,.26, .71, .73	27632, .08, .53,.53
IP destination addr.	32, 0, 0, .0002	1, 0, 0, 0
TCP header size	2, 0, 0, .000003	*19, .16, .05, .024*
ICMP types	3, 0, 0, .001	*7, .14, .14, .16*
TTL	9, 0, .1, .00002	*177, .04, .12, .023*
TOS	4, 0, 0, .0003	*44, .07, .64, .53*
TCP dest port	*8649,.35, .66, .65*	32855,.001,.002,.3
TCP flags	8, 0, 0, .00002	13, 3, 0, .00009
TCP options 4 bytes	2, 0, 0, .00002	*104, .22, .31, .18*
UDP dest port	7, 0, 0, .0001	*31, .52, .55, .45*

Percent	**Simulated**	**Real**
Packets	n = 658,801	n = 1,663,608
IP options	None	None
IP fragments	0	*0.45%*
Don't fragment (DF)	52% set	90% set
DF set in fragment	No fragments	100% bad
IP checksum	No errors	No errors
TCP checksum	No errors	*0.017% bad*
UDP checksum	No errors	No errors
ICMP checksum	No errors	*0.020% bad*
TCP reserved flags	Always 0	*0.093% bad*
TCP reserved field	Always 0	*0.006% bad*
Urgent data, no flag	None	*0.022% bad*

Range	**Simulated**	**Real**
IP packet size	(38-1500)	(24-1500)
TCP window size	(0-32737)	(0-65535)
TCP header size	(20-24)	(20-48)
Urgent pointer	(0-1)	(0-65535)
UDP packet size	(25-290)	(25-1047)

Table 4. Comparison of inside sniffer weeks 1 and 3 (simulated) with 10 weeks of real traffic after filtering (real). (a) r, r_1, r_h and r_t for discrete attributes, (b) percent true for binary attributes, (c), ranges of continuous attributes.

In Table 5 we compare inbound TCP SYN packets in the simulated and real traffic. This exposes some potential artifacts that were not apparent in the larger set of all filtered packets. The most striking difference is in IP source addresses. The number and rate of novel addresses is thousands of times higher in real traffic than in simulation. This is *not* the case when UDP and ICMP traffic (or outbound TCP) is included.

Other differences include TCP options (which determine packet size and TCP header size) and window size. Every inbound TCP SYN packet uses the exact same four TCP option bytes, which set the maximum segment size (MSS) to 1500. In reality, the number of options, their order, and the option types and values varies widely.

Window size (used to quench return traffic) is allowed to range from 0 to 65535. The full range of values is seen only in real traffic. Simulated traffic is highly predictable, always one of several values. A difference in range is also observed in source

ports (selected randomly by the client) and high numbered destination ports (often negotiated). One other type of anomaly seen only in real traffic is a nonzero value in the acknowledgment field, even though the ACK flag is not set.

We have now accounted for all of the SAD attributes that detect attacks. They all appear to be artifacts. These fall into two categories. Either the simulated range is too low, allowing detections of attacks that would otherwise be masked (e.g. window size, source port), or the rate of false alarms is too low (e.g. source address, TTL, TCP options). This does *not* mean that SAD did not legitimately detect any attacks. For example, it detects *neptune* by an anomalous source address that does not appear in real traffic either.

Attribute	**Simulated**	**Real**
Packets, n	50650	210297 + 6 errors
Source address r, r_1, r_h, r_t	29, 0, .03, .001	*24924, .45, .53, .49*
Dest addr, r	17	1 (163.118.135.1)
Src port, r	13946 (20-33388)	45644 (21-*65534*)
Dest port,r	4781 (21-33356)	1173 (13-*65427*)
IP pkt size, r	1 (44, 4 option bytes)	*8 (40-68)*
TCP options,r	1 (MSS=1500)	*103 in first 4 bytes*
Window size,r	7 (512-32120)	*523 (0-65535)*
TCP ack	Always 0	*0.02% bad*

Table 5. Comparison of simulated and real inbound TCP SYN packets (excluding TCP checksum errors).

4.2.4. Comparison at Application Protocols

We compare HTTP requests in the simulated data (weeks 1 and 3) with 10 weeks of real traffic. Because the real packets were truncated to 200 bytes (usually 134-146 bytes of payload), we examine only the first 134 bytes in both sets. Table 6 summarizes the differences we found.

Inbound HTTP Requests	**Simulated**	**Real**
Number of requests, n	16089	82013
Different URLs requested, r, r_1	660, .12	*21198, .58*
HTTP versions, r	1 (1.0)	2 (1.0, 1.1)
Commands (GET, POST...), r	1 (GET)	*8*
Options, r	6	*72*
User-agents, r, r_1	5, 0	*807, .44*
Hosts, r	3	13

Table 6. Comparison of HTTP requests in simulated traffic (inside weeks 1 and 3) and 10 weeks of real traffic.

There are two simulated web servers (hume and marx). However, the one real web server receives more traffic and has more web pages. The distribution of real URLs is approximately Zipf, consistent with findings by Adamic [1]. A characteristic of a Zipf distribution is that about half of all values occur exactly once. The simulated URLs are distributed somewhat more uniformly. Many of the singletons are failed requests which were simulated by replacing the last 4 characters of the file name (e.g. *html*) with *xxxx*.

There is a huge disparity in the number of user-agents (client types). The simulated traffic has only five, all versions of *Mozilla* (Netscape or Internet Explorer). Real web servers are frequently accessed by search engines and indexing services. We found the top five user-agents in the real data to be (in descending order) *Scooter/3.2, googlebot/2.1, ia_archiver, Mozilla/3.01*, and *http://www.almaden.ibm.com/cs/crawler*. They also have a Zipf distribution.

The only simulated HTTP command is GET, which requests a web page. The real traffic has 8 different commands: GET (99% of requests), HEAD, POST, OPTIONS, PROPFIND, LINK, and two malformed requests, *No* and *tcp_close,*. There is also a much wider variety of options, although some of these are due to the introduction of HTML/1.1. Nevertheless there is wide variation in capitalization and spacing. In the simulated traffic, HTTP options invariably have the form *Keyword: value*, with the keyword capitalized, no space before the colon and one space afterwards. This is usually but not always the case in real traffic. Furthermore, we occasionally find spelling variations, such as *Referrer:* (it is normally misspelled *Referer:*) or the even more bizarre *Connnection:* with three n's. Some keywords are clearly malformed, such as *XXXXXXX:* or ~~~~~~~:. A few requests end with a linefeed rather than a carriage-return and linefeed as required by HTTP protocol. Finally there are some requests which are clearly suspicious. We found 33 requests similar to the following two examples.

GET /scripts/..%255c%255c../winnt/system32/cmd.exe?/c+dir
GET /MSADC/root.exe?/c+dir HTTP/1.0

Undoubtedly this did not accomplish much on a UNIX host.

We look only briefly at SMTP (mail) and SSH (secure shell). These are the only other TCP application protocols besides HTTP that exist in sufficient quantity in both data sets to do a useful comparison. Like HTTP, we once again find that real traffic is "messy", high in benign anomalies. Table 7 summarizes the results.

Inbound Request	**Simulated**	**Real**
SMTP requests, n	18241	12911
First command, r	2	*7*
HELO hosts, r, r_1	3, 0	*1839, .69*
EHLO hosts, r, r_1	24, .04	*1461, .58*
No initial HELO or EHLO	0	*3%*
Lower case commands	0	*0.05%*
Binary data in argument	0	*0.1%*
SSH requests, n	214	666
SSH versions, r, r_1	1, 0	*32, .36*

Table 7. Comparison of inside sniffer weeks 1 and 3 with 10 weeks of real inbound SMTP and SSH requests.

A normal SMTP session starts with HELO or EHLO (echo hello), but these are optional. In the simulated traffic, every session starts with one of these two commands. However, about 3% of real sessions start with something else, usually RSET, but also QUIT, NOOP, EXPN, or CONNECT. About 0.2% of real commands are lower case. One command (EXPN root) is suspicious.

The number of simulated remote hosts sending and receiving mail (arguments to HELO and EHLO) is clearly unrealistic. This is also reflected in the small number of source IP addresses in general. The simulated traffic has one malformed command, an

EHLO with no argument. The real traffic does too, and a variety of other malformed arguments, including binary strings (1-21 bytes, probably too short to be a buffer overflow). The host name arguments are roughly Zipf distributed, with over half appearing only once.

An SSH session opens with the client version string. The simulated traffic uses a single client version. In real traffic there are many versions, again Zipf distributed.

4.3. Potential Source of Artifacts

In the previous section we saw that many attributes have a wider range of values (higher *r*) in real traffic than in simulation, and a higher growth rate (r_1, r_h, and r_t), which would make them harder to model. Why is this?

One possibility is that the various traffic sources (hardware, software, and people) are modeled incorrectly with *r* too small and static, i.e. too predictable. A second possibility is that individual sources display the correct ranges of values, but their timing is wrong, resulting in an incorrect growth rate for *r*. A third possibility is that the individual sources are correct in both values and timing, but there are too few sources to simulate the diversity of real traffic.

4.3.1. Source Hosts

To investigate the first possibility, we compare single sources in the simulated and real traffic. In the first two columns of Table 8, we compare TCP SYN packets originating from the simulated Solaris host (*pascal*) with the real host, which also runs Solaris. We see nearly identical behavior. Both sources produce highly predictable values, especially for attributes such as window size, TTL, and TCP options, fields that we previously identified as artifacts.

In the rightmost two columns, we compare inbound TCP SYN packets, which represent aggregate sources. In this case, we see (as before) that *r* is higher in real traffic. The simulated aggregate traffic bears a greater resemblance to the simulated single source than to real aggregate traffic. This effect is most noticeable for TCP options and packet size (Ethernet, IP and TCP header), but also for TTL and window size.

r (values)	Sim out	Real out	Sim in	Real in
Packets	n=3165	n=6932	n=29263	n=7063
Ether size	1 (60)	1 (58)	1 (60)	*6 (60-82)*
IP length	1 (44)	1 (44)	1 (44)	*6 (44-68)*
TOS	1 (0)	1 (0)	3(0,8,16)	4 (0-3)
DF	1 (1)	1 (1)	2 (0-1)	2 (0-1)
TTL	1 (255)	1 (255)	7	*65*
Header	1 (24)	1 (24)	1 (24)	*6 (24-48)*
Window size	2 (8760-24820)	1 (24820)	7 (512-32120)	*40 (512-65535)*
Urg ptr	1 (0)	1 (0)	1 (0)	2 (0-1738)
Options	1 (MSS)	1 (MSS)	1 (MSS)	*20*

Table 8. Number (r) and range of values in outbound (from Solaris) and inbound TCP SYN packets in week 3 and the small real data set.

4.3.2. Self-Similarity

Although the evidence suggests that the source of simulation artifacts is too few sources, we cannot yet rule out a lack of burstiness or self-similarity as another source. We know that many types of network events tend to occur in bursts separated by long gaps, regardless of time scale (self-similarity) [15, 26]. If individual sources were instead modeled as Poisson processes (independent events, lacking long gaps), then in an aggregate process we would expect to see all possible values within a short period of time. On the other hand, if individual sources produced bursts of events with long gaps, then *r* would continue to grow as new sources are seen for the first time. Furthermore, because gaps can be arbitrarily long, *r* should grow throughout any arbitrarily long trace.

We first observe that both the simulated and real traffic are bursty. When we examine the distribution of intervals between successive events, we find that the distribution is heavy tailed compared to a Poisson distribution, i.e. lots of short and long gaps. We found this to be true for many packet types: Ethernet, ARP, ICMP, UDP, TCP, TCP SYN, HTTP SYN, and SMTP SYN. Although these are all aggregate processes, the results hold because the sum of Poisson processes is a Poisson process, but the sum of self-similar processes tends to remain self-similar.

We also measured the Hurst parameter [15], a measure of how rapidly the burstiness "smoothes out" as the time scale increases. One measure of burstiness is the standard deviation of events per time interval, e.g. packets per second or packets per minute. As we increase the time scale by a factor of M (e.g. 60), we would expect the relative standard deviation to decrease by a factor of $M^{1/2}$ for a Poisson process. For a purely self-similar process, there is no decrease at all, i.e. M^0. A process is said to have a Hurst parameter of H if the standard deviation decreases by a factor of M^{1-H}, where H = 0.5 indicates a Poisson process and H = 1 indicates a purely self-similar process. Many network events have Hurst parameters of 0.7 to 0.9.

We compared Hurst parameters in the simulated and real traffic for all of the packet types mentioned above. We measured H by computing the relative standard deviation of the event rate at sampling intervals of M = 1, 10, 100, and 1000 seconds. We then estimated the Hurst parameter between successive values of M as

$$H_{M\text{-}10M} = 1 + \log_{10}((\sigma_{10M}/\mu_{10M}) / (\sigma_M/\mu_M))$$

where $H_{M\text{-}10M}$ is the Hurst parameter estimate between sampling rate M and 10M, σ_M is the sample standard deviation for samples over M seconds, and μ_M is the sample mean. Because measuring H requires a contiguous trace, we compared each of the two small traces of one million packets with two real traces taken from the first million packets of simulated week 3, days 1 and 7.

We found that in most cases, the simulated traffic has similar or higher Hurst parameters than real traffic. This is true for both individual and aggregate processes. Table 9 shows two examples, where we compare outbound and inbound TCP SYN packets from the first of two real traces with the first million packets of week 3, day 1. Although the number and rate of simulated outbound packets is small, we obtained similar values with longer traces.

	Sim out	Real out	Sim in	Real in
Packets	235	5892	2265	5387
Packets./sec.	.0108	.148	.104	.135
H, 1-10 sec.	.547	.610	.594	.670
H, 10-100 s.	.614	.484	.630	.694
H, 100-1000 s.	.719	.617	.694	.747

Table 9. Packets, packet rates, and Hurst parameter estimates over the range 1 to 1000 seconds for outbound (from the Solaris host) and inbound TCP SYN rates for simulated and real traffic.

This evidence seems to rule out the first two possibilities that individual sources are modeled incorrectly either in range of values or in timing. Instead, we conclude that a likely source of artifacts is that there are too few independent sources of traffic to duplicate the complexity of Internet traffic.

4.4. Summary

We found what appear to be simulation artifacts at every layer of the protocol stack from the data link layer to the application layer. These are of two types. First, the simulated data lacks "crud", anomalous but mostly benign events that might trigger false alarms during testing, such as checksum errors, IP fragments with the "don't fragment" flag set, TCP retransmissions with inconsistent or partially overlapping payloads, or data in the TCP reserved fields, urgent pointer, or acknowledgment field when the corresponding flags are not set. These anomalies make up about 0.01% of real packets.

Second, many attributes that have a small and static range of values in simulation would actually have a much wider range in practice (often with a power law or Zipf distribution), and this range would grow at a constant rate, introducing a huge disparity in the rate of novel events. The problem occurs in Ethernet and IP protocols and source addresses, TOS, TTL, TCP options and window size, HTTP and SMTP commands and arguments, and HTTP and SSH client versions. The problem is especially severe for source IP addresses for inbound (but not outbound) TCP SYN packets. This attribute is monitored by nearly all network intrusion detection systems. The simulated rate of novel addresses (and thus the false alarm rate) is too low by a factor of many thousands.

Finally, we compared individual traffic sources in the simulated and real traffic, and these appear to be modeled correctly, both in range of values and in timing. We believe the problem is due to too few sources.

5. EVALUATION WITH MIXED TRAFFIC

The evidence presented in Sections 3 and 4 suggest a problem with the LL data. However, this data was generated at great expense and could not easily be replaced. We would prefer a solution that fixes the data as it exists now, rather than require that a new set be created.

We believe it is impractical to synthesize Internet traffic accurately due to its vast complexity. However, we believe that attacks can be – and for the most part were – simulated correctly. Thus we propose to use real traffic (with all the usual implications about privacy and security) as background and training data, but to continue to use the labeled, simulated attacks as before.

Our proposal is to add real traffic to the LL data to make it appear as if it were being sent and received during the simulation. We believe it is not necessary to remove the simulated background traffic because the combination should be similar (in the statistical sense of Section 4) to the real traffic alone. To see this, let A_S be the set of values of attribute A seen in simulation up to the present time, and let A_R be the corresponding set of values seen in real traffic. Then the set of values A_M seen in merged traffic would be at all times:

$$A_M = A_S \cup A_R$$

Note that the r statistic for attribute A_S, which we denote r_S is simply $|A_S|$. Likewise, we define $r_R = |A_R|$ and $r_M = |A_M|$. Therefore, we have at all times:

$$\max(r_S, r_R) \le r_M \le r_S + r_R$$

In cases where we suspect r is an artifact, we have $r_S << r_R$, and therefore $r_M \approx r_R$, so removing the simulated traffic would have little effect. Furthermore, because this is true at all times, r_M and r_R would have similar growth rates.

A problem can occur when A_R is too small or empty, i.e. there is little or no real traffic of types where A is defined to mix with the simulation. In this case, $r_M \approx r_S$, and the artifact, if there is one, would not be removed. One such example is the destination address of incoming traffic, where there are $r_S = 16$ simulated hosts and $r_R = 1$ real host. We are unable to test whether the destination address is an artifact in the simulation (although we have no reason to believe that it would be). Other untestable attributes are those of FTP and telnet payloads, because there is little FTP and no telnet traffic in our real data. (Remote login and FTP are available only via the SSH protocol).

We wish to evaluate network anomaly detection systems on mixed data. Our approach is as follows. First, we analyze the system to determine which attributes are monitored. Then we test the simulated and real data to determine which attributes are present in the simulation, but absent or rare in the real data. Then we either modify the system to ignore these attributes (e.g. remove rules for FTP and telnet), or we modify the real data to remove the discrepancy (e.g. modify destination IP addresses in the real traffic). We illustrate the process with three systems, SPADE, PHAD, and LERAD.

5.1. Data Preparation

For our mixed traffic, we use the same large, filtered data set as described in Section 4. We have 579 hours of traffic, which is more than enough to mix into the 146 hours of traffic in inside sniffer week 3 plus the 198 hours in weeks 4 and 5. We mix the traffic in a 1:1 ratio, i.e. one hour of simulated traffic is mixed with one hour of real traffic. Other ratios would be possible by stretching or compressing the real traffic, but we do not do this.

We mix the traffic to make it appear as if all of the collected data occurs during the simulation. We do this by adjusting the time stamp of the first real packet to match the time stamp of the first simulated packet, then maintain the relative times of the other real packets, excluding gaps in the two collections. This is illustrated in Figure 1. Time reads from left to right.

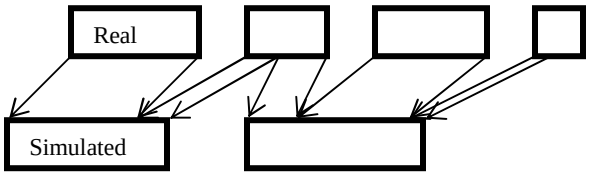

Figure 1. Mapping real time into simulation time when there are gaps in collection in both data sets.

The real traffic consists of 50 traces, divided into 10 weeks. We mix these into weeks 3 (training), 4, and 5 (test) of the inside sniffer data to prepare three mixed data sets, which we label A, B, and C as shown in Table 10. Prior to mixing, both the simulated and real traffic are filtered as described in Section 4.2.1 to pass only truncated and rate limited inbound client to server requests. We denote the unmixed data (after filtering) as set S.

Set	Training data	Test data
S	LL inside week 3	LL inside weeks 4-5
A	S + real weeks 1-3	S + real weeks 4-7
B	S + real weeks 4-6	S + real weeks 7-10
C	S + real weeks 7-9	S + real weeks 1-4

Table 10. Mixed data sets used for evaluation. All data is filtered.

The results for SAD in Section 3 were obtained with sets S and C. However sets A and B give results similar to C. In this case, filtering has little effect because most inbound TCP SYN are passed through.

5.2. Algorithm Preparations

In this section we describe how we modify SPADE, PHAD, and LERAD to meet the requirement that it not test any attributes where $r_R << r_S$. We can do this either by modifying the data (SPADE), the algorithm (LERAD), or determining that no modification is needed (PHAD).

5.2.1. SPADE Modifications

SPADE is a frequency based model of inbound TCP SYN packet addresses and port numbers. It has four probability modes (0-3), which assign scores to every such packet as follows:

0. $1/P(SA, SP, DA)P(SA, SP, DP)/P(SA, SP)$
1. $1/P(DA, DP, SA, SP)$
2. $1/P(DA, DP, SA)$
3. $1/P(DA, DP)$ (default)

where the joint probabilities are based on counts up through the current packet, and SA, SP, DA, and DP denote source address, source port, destination address, and destination port, respectively.

All probability modes include the destination address (DA), which implies that SPADE would build separate models for the simulated and real hosts. To prevent this from happening, we randomly replace all real destination IP addresses with one of the four main targets in the simulation (pascal, hume, marx, or zeno), making it appear as if the real traffic were on these hosts instead.

5.2.2. PHAD Modifications

PHAD is a time-based global model of packet header fields. If any packet (inbound or outbound, client or server) displays a value never seen in training, then PHAD assigns a score of $\Sigma\ tn/r$, where t is the time since the previous anomaly, n is the number of training packets, and r is the number of allowed values, and the sum is over all of the anomalous attributes.

There are 34 attributes corresponding to the various 1 to 4 byte fields in the Ethernet, IP, TCP, UDP, and ICMP packet headers. The conditions for these fields to be present, is that the packet be of the corresponding type. From Table 3 (Section 4.1) we see that all five types of packets are available in comparable quantities in both the simulated and real traffic. Thus, no rules need to be removed from PHAD.

5.2.3. LERAD Modifications

LERAD is a time-based model, like PHAD, assigning a score of $\Sigma\ tn/r$ to novel attribute values. It creates conditional rules of the form

$$\text{if } A_1 = v_1 \text{ and } A_2 = v_2 \text{ and ... then } A_k \in \{v_k, v_{k+1}, \ldots v_{k+r-1}\}$$

where A_1 through A_k are arbitrary attributes and the v_i are values. The rules are randomly selected such that they are always satisfied in training and have high n/r.

LERAD models inbound TCP streams from client to server. The attributes are date, time, single bytes of the source and destination address, source and destination ports, TCP flags of the first, next to last or last packet, duration, length, and the first 8 words in the application payload.

There are many potential rules that could exclude real traffic, for example "if DA = *pascal* and DP = FTP then ...". Rather than modify LERAD to avoid such rules, we modify it to record the number of simulated and real training instances that satisfy the rule antecedent, then weight each rule by the fraction of real traffic when computing the anomaly score. This has the effect of removing rules that depend only on the simulated traffic.

5.3. Evaluation Criteria

We evaluate SPADE, PHAD, and LERAD on the LL data with and without real traffic mixed in. This serves two purposes. First, we wish to know if we successfully removed the artifacts. Second, we wish to predict how these systems would work on real traffic. Although the rate of attacks in the LL data is artificially high (except possibly for probes), we can still use these results to estimate the probability of detecting an attack given any false alarm rate (e.g. 10 per day), on the type of traffic that we add to the data.

To test whether artifacts are removed, we look at each attack and the attributes that lead to its detection with and without mixing. If, based on the attack's description, the detection is suspect, then we would expect it to be missed when real traffic is added. For example, we would expect that HTTP or SMTP attacks detected by source address in simulated traffic would be missed in mixed traffic. However, if the feature is genuine, for example, *neptune's* forged source address, then the attack would still be detected, although it could still be missed due to a higher false alarm rate. In general, we will use the following somewhat subjective guidelines to determine whether a detection is legitimate.

- Source address is legitimate for denial of service (DOS) attacks that spoof it, or if the attack is on an

authenticated service (e.g. telnet, auth, SSH, POP3, IMAP, SNMP, syslog, etc), and the system makes such distinctions. FTP is anonymous in the LL data, so we consider it public.

- Destination address is legitimate for probes that scan addresses, e.g. *ipsweep*.
- Destination port is legitimate for probes that scan or access unused ports, e.g. *portsweep*, *mscan*, *satan*. It is debatable whether it is legitimate for attacks on a single port, but we will allow them.
- TCP state anomalies (flags, duration) are legitimate for DOS attacks that disrupt traffic (*arppoison*, *tcpreset*), or crash the target (*ntfsdos*, *dosnuke*).
- IP fragmentation is legitimate in attacks that generate fragments (*teardrop*, *pod*).
- Packet header anomalies other than addresses and ports are legitimate if a probe or DOS attack requires raw socket programming, where the attacker must put arbitrary values in these fields.
- Application payload anomalies are legitimate in attacks on servers (usually R2L (remote to local) attacks, but may be probes or DOS).
- TCP stream length is legitimate for buffer overflows.
- No feature should legitimately detect a U2R (user to root) or Data attack (security policy violation).

5.4. Evaluation Procedure

We use the EVAL program (available at [19]) to test whether an attack is detected. EVAL uses the same criteria as the original 1999 LL evaluation as described in [17]. However we have had to make some assumptions where the description is ambiguous.

EVAL counts an attack as detected if there is at least one alarm that correctly identifies the target IP address (or one target if there is more than one) and if the alarm occurs within 60 seconds of any point within any segment of the attack. Attack segments are derived from the start times, durations, and destination addresses from the LL master detection truth table [17], when (with three exceptions), the destination is a local address (172.16.x.x or 192.168.x.x in the LL set). The three exceptions allowed by EVAL are two instances of *httptunnel* and one instance of *portsweep* which have no segments with a local destination address. In these cases, the local address from the LL master identification list [17] is used instead.

Systems are evaluated based on the number of "in-spec" attacks detected for a given number of false alarms as the scoring threshold is varied. An attack is in-spec if the system is designed to detect it, based on the attack type, input data, target operating system, and whether the attack is new or stealthy. Each attack instance during week 4 or 5 is labeled with this information so that the scoring is unambiguous. For this paper, we consider an attack in-spec if there is evidence for the attack in the inside sniffer data on which we test it. There are 177 such instances out of 201 in weeks 4 and 5.

An alarm is counted as a false alarm if it falls outside of all attack segments by more than 60 seconds, whether or not that segment is in-spec. If an alarm occurs during two overlapping attacks, then EVAL counts both attacks as detected. For a given false alarm rate, EVAL sets the threshold as low as possible without exceeding that rate. In other words, if the limit is 100 false alarms, then detections between the 100'th and 101'st highest scoring false alarms are included. In case of tie scores, alarms are ranked in the order they are input, with the first alarm ranking highest.

It is often possible to decrease the number of false alarms without affecting detections by consolidating bursts of alarms. We use the program AFIL.PL (also available at [19]) to do this prior to all our evaluations. AFIL.PL divides the alarm sequence into one-minute intervals, then discards any alarm if there is a higher scoring alarm in the same interval that identifies the same target.

6. EVALUATION RESULTS WITH MIXED TRAFFIC

Based on the evaluation criteria and procedures described in Section 5, we obtained the following results for SPADE, PHAD, and LERAD.

6.1. SPADE

We tested SPADE version v092200.1, which is built into SNORT 1.7 Win32 [29]. We used sets S, A, B, and C, which were further modified as in Section 5.2 to randomly substitute real destination addresses with simulated ones. All SPADE options were set to their default values, and all SNORT rules other than SPADE were turned off. SPADE does not have separate training and test modes, so we ran it on weeks 3 through 5 continuously, discarding all alarms in week 3. SPADE uses an adaptive threshold with various parameters to control alarm reporting. However we used the raw score reported by SPADE instead. The default threshold allows thousands of false alarms so we do not believe that any were lost.

Results are shown in Table 11 for each of the four probability modes. We used a threshold of 200 false alarms rather than 100 because the numbers are low. SPADE detects about half as many attacks at 100 false alarms.

SPADE detections at 200 FA	S	A, B, C
0: P(SA, SP, DA)P(SA, SP, DP)/P(SA, SP)	6	6, 6, 7
1: P(DA, DP, SA, SP)	1	0, 0, 0
2: P(DA, DP, SA)	6	2, 1, 1
3: P(DA, DP) (default)	8	9, 8, 7

Table 11. Attacks detected by SPADE at 200 false alarms according to EVAL on filtered inside sniffer weeks 3-5 (S) and when mixed with real traffic (A, B, C) in each probability mode.

Probability modes 0 and 1 include the source port (SP), which is normally picked randomly by the client and would not be expected to yield useful information. The six attacks detected in mode 0 on set S are *insidesniffer*, *syslogd*, *mscan*, *tcpreset*, *arppoison*, and *smurf*. All but *mscan* are probably coincidental because the others generate no TCP SYN packets. However, all but *syslogd* target multiple hosts, increasing the likelihood of a coincidental alarm for one of the targets.

However modes 2 and 3 show the effects of mixing clearly. We have previously identified source address (SA) as an artifact. We now find that adding real data removes most of the detections from mode 2, which uses SA, but not from mode 3, which does not. On S, mode 2 detects *guest*, *syslogd*, *insidesniffer*, *perl*, *mscan*, and *crashiis*. By our previously mentioned criteria, *guest* (telnet password guessing) could legitimately be detected by SA,

and *mscan* (a probe for multiple vulnerabilities on a range of hosts) by destination address or port (DA or DP). We do not count *syslogd* or *insidesniffer* (no TCP traffic), *perl* (a U2R attack), or *crashiis* (an HTTP attack).

SPADE detects only *mscan* on all three mixed sets. On A, it also detects *portsweep*, which also can legitimately be detected by DP. Thus, our results are consistent with the claim that SA (but not DP) is an artifact and that the artifact is removed in mixed traffic. When real traffic is added, the fraction of legitimate detections goes from 2/6 to 1/1 (or 2/2).

6.2. PHAD

Although it is not necessary to modify PHAD to test it on mixed traffic, we modified it to include the TTL field. This field was removed in the original version because it was believed to be an artifact. We wish to see whether adding real traffic will remove it.

On sets S, A, B, and C, PHAD detects 76, 18, 32, and 25 in-spec attacks respectively at 100 false alarms according to EVAL. The median mixed result is for C, so we restrict our detailed analysis to S and C. However, A and B give similar results.

In Table 12, we list the attacks detected in sets S and C grouped by the attribute that contributes the most to the detection. We classify each detection as legitimate or not according to the criteria at the beginning of this section. We list the number of legitimate and total detections separated by a slash, then list the legitimate and non-legitimate detection types also separated by a slash. For example, out of the 36 attacks detected in S by TTL, we classify only the 7 instances of *portsweep* as legitimate, because this is the only attack which requires the attacker to fill in this field.

Attribute	Legitimate/Detected in S	In C
TTL	7/36: portsweep / apache2, back, casesen, crashiis, ffbconfig, guessftp, guesstelnet, ipsweep, mailbomb, named, neptune, netbus, netcat_breakin, ntinfoscan, ppmacro, sechole, smurf, yaga	1/2: portsweep / ffbconfig
Ether-DA	0/1: mscan	0/1: mscan
Dest. IP address	1/5: portsweep / ncftp, guesstelnet	0/3: ncftp
IP frag	7/8: pod, teardrop / insidesniffer	5/5: teardrop, pod
Source IP address	0/5: xlock, portsweep, ncftp, processtable, sendmail	0/0
Checksums	1/6: udpstorm / smurf, apache2	0/0
TCP flags	6/6: portsweep, queso	9/9: queso, portsweep, dosnuke
Urgent ptr	4/4: dosnuke	1/1: dosnuke
UDP DP	5/5: satan, portsweep, udpstorm	0/0
Packet size	0/0	1/2:satan/ncftp
Window	0/0	2/2: portsweep
Total	**31/76 (41%)**	**19/25 (76%)**

Table 12. Attacks detected by PHAD attributes at 100 false alarms in sets S and C.

When we add real traffic, the percentage of legitimate detections increases from 41% to 76%. Furthermore, of the 6 remaining detections we classify as not legitimate, 3 are destination address anomalies, which we are unable to remove because there is only one new destination address in the real traffic. One other (*ffbconfig*) is detected because it overlaps *portsweep* and is detected by the same alarm. If we ignore these 4 cases, then the fraction of legitimate detections increases to 19/21 or 90%.

6.3. LERAD

We modified LERAD to weight rules by the fraction of real traffic satisfying the antecedent in training. However, we find in practice that this weighting has very little effect. Almost all of the rules are satisfied by a significant fraction of real traffic, and the effect is to decrease the number of detections by less than 3%.

We also modified the TCP stream reassembly algorithm to handle truncated and filtered traffic as described in Section 4.2.1. The LL data contains complete packets, allowing streams to be reassembled completely. However, truncated TCP packets would leave gaps. Thus, we truncate the stream after the first 134 bytes of the first payload packet to match the maximum payload size of the filtered traffic. Our filter removes closing TCP flags. Therefore we also modify the TCP flag attributes to be the flags of the last three packets up through the payload, instead of the first and last two packets in the completely reassembled stream. The modified reassembly is also applied to the simulated traffic.

Attribute	Legitimate/Detected in S	In C
Source address	8/26: dict, guesstelnet, guest, sshprocesstable, sshtrojan / casesen, crashiis, fdformat, ffbconfig, guessftp, netbus, netcat_setup, perl, ps, sechole, sqlattack, xterm, warezclient, warezmaster	0/0
Dest. address	1/6: mscan / ncftp, guesstelnet	1/6: mscan / ncftp, guesstelnet
Dest. port	14/14: ftpwrite, guesspop, imap, ls_domain, satan, named, neptune, netcat, netcat_breakin	11/11: ftpwrite, ls_domain, satan, named, netcat, netcat_breakin,
Payload	22/29: apache2, back, crashiis, imap, mailbomb, ntinfoscan, phf, satan, sendmail / guesstelnet, portsweep, yaga	8/8: back, imap, ntinfoscan, phf, satan, sendmail
Duration	0/1: insidesniffer	0/0
Length	0/2: netbus, ppmacro	1/1: sendmail
TCP flags	4/9: dosnuke / back, loadmodule, sendmail	4/4: dosnuke
Total	**49/87 (56%)**	**25/30 (83%)**

Table 13. Attacks detected by LERAD at 100 false alarms on sets S and C.

LERAD uses a randomized rule generation algorithm. In 5 runs on set S, EVAL detects 87, 88, 80, 85, and 91 detections. (The lost of detections, compared to about 114 on unfiltered traffic, is due mostly to the loss of TCP flags and some of the payload). On one run each on sets A, B, and C, the modified

LERAD detects 29, 30, and 30 in-spec attacks. Thus, we confine our detailed analysis to the middle values of each group, the run detecting 87 attacks in S, compared with C. As with PHAD, we categorize each detection as legitimate or not, and list the results in Table 13.

Once again, we find that adding real data reduces the fraction of questionable detections. The fraction we allow as legitimate increases from 56% to 83%. Of the 5 questionable detections in C, all are detected by destination address, which is unaffected by the real traffic.

6.4. Results Summary

We tested SPADE, PHAD, and LERAD on mixed data, and found by an analysis of the detected attacks that suspected simulation artifacts were removed. In particular, two strong artifacts were removed: TTL from PHAD, and source address from SPADE and LERAD.

We also reached similar conclusions in tests on ALAD and NETAD, although we omit the detailed analysis. We modified both programs so that no rule depends exclusively on simulated data.

ALAD models TCP streams like LERAD, but uses fixed rather than learned rules. We modified these rules to remove dependencies on the destination address, which would distinguish the real traffic. We also removed rules for application payloads other than HTTP, SMTP, and SSH. We used LERAD's modified TCP stream reassembly algorithm. The result was to increase the fraction of legitimate detections, mostly by removing detections by source address.

NETAD models packets, like PHAD, but for several types such as inbound TCP, inbound TCP SYN, HTTP, SMTP, telnet, and FTP. We removed the telnet and FTP rules. Again, the fraction of legitimate detections was increased, mostly by removing source address and TCP window size anomalies.

The results are summarized in Table 14. The original number of detections is the number reported in the literature at 100 false alarms when trained on inside week 3 and tested on weeks 4-5 before the data is filtered or the algorithm is modified. For sets S and C we show the number of legitimate and total detections, and the percentage legitimate. Set C generally resulted in a number of detections between those of sets A and B, and is therefore the most representative of the mixed results. In every case, the fraction of legitimate detections increases when mixed data is used.

System	Orig	Legit/S (pct)	Legit/C (pct)
SPADE, mode 2		2/6 (33%)	1/1 (100%)
PHAD, no TTL	54	31/51 (61%)	19/23 (83%)
ALAD	59	16/47 (34%)	10/12 (83%)
LERAD (avg.)	114	49/87 (56%)	25/30 (83%)
NETAD	132	61/128 (48%)	27/41 (67%)

Table 14. Legitimate and total detections at 100 false alarms on sets S and C. The original results (orig) are the published results based on the unmodified algorithm on unfiltered data (inside weeks 3-5).

7. CONCLUSIONS AND FUTURE WORK

We analyzed the attack-free portions of the 1999 LL inside sniffer traffic by comparing it with one source of real traffic from the point of view of attributes that are important to anomaly detection. We discovered many attributes that have a small, fixed range in simulation, but a large and growing range in real traffic, in particular, remote client addresses, TTL, TCP options and TCP window size. The simulated traffic also lacks "crud", such as IP fragments, garbage data in unused TCP fields, bad checksums, and malformed application commands and arguments.

While these differences are of no consequence to a signature detection system, they could be to an anomaly detection system, which is designed to be sensitive to unusual events. The LL background underestimates the frequency of these events, which would otherwise generate false alarms in many systems. It also allows the systems to detect attacks based on idiosyncrasies that would normally be masked.

We propose solving these problems by adding real traffic to the simulation. This requires careful analysis to ensure that the system cannot distinguish the real traffic from the simulation, and that it does not monitor traffic types for which no real data is available. We did this for five systems and showed that many attacks that appear to be detected by suspected simulation artifacts are no longer detected when real traffic is added.

Although we tested only the 1999 inside sniffer traffic, we believe the problem exists in the outside traffic and the 1998 traffic because they were generated by the same methodology. We do not believe that the host based data (BSM, audit logs, etc.) are affected by artifacts because this data was generated by real hardware and software, rather than simulated.

Our analysis is based on a single source of real network traffic. Obviously every environment is different, so we must be cautious about drawing general conclusions. Our results need to be confirmed using other sources of real traffic. Also, we do not know the effects of changing the proportion of simulated and real traffic, which could be done by squeezing, stretching, or sampling the real data, or by using other sources with different rates. Our analysis assumes that the real traffic is attack-free, but we know that this is not the case. Finding and removing (or labeling) all of the hostile traffic would be difficult.

Evaluations using real traffic are usually not repeatable because privacy and security concerns usually prohibit the release of this data off-line. Furthermore, as hardware gets faster, software gets more complex, and new protocols are introduced, real traffic will look less and less like the LL data. It is more common now for data to be encrypted and inaccessible. These are difficult problems that remain to be solved.

Acknowledgments

This research is partially supported by DARPA (F30602-00-1-0603).

References

[1] L. A. Adamic, "Zipf, Power-laws, and Pareto - A Ranking Tutorial", http://ginger.hpl.hp.com/shl/papers/ranking/ranking.html (2002)

[2] D. Anderson,. et. al., "Detecting Unusual Program Behavior using the Statistical Component of the Next-generation Intrusion Detection Expert System (NIDES)", Computer Science Laboratory SRI-CSL 95-06 May 1995. http://www.sdl.sri.com/papers/5/s/5sri/5sri.pdf

[3] S. Axelsson, "Research in Intrusion Detection Systems: A Survey", TR 98-17, Chalmers University of Technology, 1999.

[4] D. Barbara, Wu, S. Jajodia, "Detecting Novel Network Attacks using Bayes Estimators", SIAM Intl. Data Mining Conference, 2001.
[5] C. Elkan, "Results of the KDD'99 Classifier Learning Contest", http://www.cs.ucsd.edu/users/elkan/clresults.html (1999)
[6] E. Eskin, "Anomaly Detection over Noisy Data using Learned Probability Distributions", Intl. Conf. Machine Learning, 2000.
[7] E. Eskin, A. Arnold, M, Prerau, L. Portnoy & S. Stolfo. "A Geometric Framework for Unsupervised Anomaly Detection: Detecting Intrusions in Unlabeled Data", In D. Barbara and S. Jajodia (editors), *Applications of Data Mining in Computer Security*, Kluwer, 2002.
[8] S. Floyd, V. Paxson, "Difficulties in Simulating the Internet", IEEE/ACM Transactions on Networking, 2001.
[9] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A Sense of Self for Unix Processes", Proc. 1996 IEEE Symposium on Computer Security and Privacy, 1996.
[10] S. Forrest, Computer Immune Systems, Data Sets and Software, http://www.cs.unm.edu/~immsec/data-sets.htm (2002).
[11] W. Gale, G. Sampson, "Good-Turing Frequency Estimation without Tears", Journal of Quantitative Linguistics 2.217-37, 1995.
[12] A. K. Ghosh, A. Schwartzbard, "A Study in Using Neural Networks for Anomaly and Misuse Detection", Proc. 8'th USENIX Security Symposium, Aug. 26-29 1999, Washington DC.
[13] J. Hoagland, SPADE, Silicon Defense, http://www.silicondefense.com/software/spice/
[14] B. A. Huberman, L. A. Adamic, "The Nature of Markets in the World Wide Web", http://ideas.uqam.ca/ideas/data/Papers/scescecf9521.html (1999)
[15] W. E. Leland,M. S. Taqqu, W. Willinger, D. W. Wilson, "On the Self-Similar Nature of Ethernet Traffic", ACM SIGComm '93, San Francisco, 1993.
[16] Y. Liao and V. R. Vemuri, "Use of Text Categorization Techniques for Intrusion Detection", Proc. 11th USENIX Security Symposium, 51-59, 2002.
[17] R. Lippmann, et al., "The 1999 DARPA Off-Line Intrusion Detection Evaluation", Computer Networks 34(4) 579-595, 2000. Data is available at http://www.ll.mit.edu/IST/ideval/
[18] M. Mahoney, "Network Traffic Anomaly Detection Based on Packet Bytes", to appear, Proc. ACM-SAC, Melbourne FL, 2003.
[19] M. Mahoney, Source code for PHAD, ALAD, LERAD, NETAD, SAD, EVAL and AFIL.PL is available at http://cs.fit.edu/~mmahoney/dist/
[20] M. Mahoney, P. K. Chan, "PHAD: Packet Header Anomaly Detection for Identifying Hostile Network Traffic", Florida Tech. technical report 2001-04, http://cs.fit.edu/~tr/
[21] M. Mahoney, P. K. Chan, "Learning Models of Network Traffic for Detecting Novel Attacks", Florida Tech. technical report 2002-08, http://cs.fit.edu/~tr/
[22] M. Mahoney, P. K. Chan, "Learning Nonstationary Models of Normal Network Traffic for Detecting Novel Attacks ", Edmonton, Alberta: Proc. SIGKDD, 2002, 376-385.
[23] J. McHugh, "Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory", Proc. ACM TISSEC 3(4) 262-294, 2000.
[24] P. G. Neumann, P. A. Porras, "Experience with EMERALD to DATE", Proc. 1st USENIX Workshop on Intrusion Detection and Network Monitoring, Santa Clara, CA, 1999, pp. 73-80
[25] V. Paxson, "Bro: A System for Detecting Network Intruders in Real-Time", Lawrence Berkeley National Laboratory Proceedings, 7'th USENIX Security Symposium, Jan. 26-29, 1998, San Antonio TX,
[26] V. Paxson, S. Floyd, "The Failure of Poisson Modeling", IEEE/ACM Transactions on Networking (3) 226-244, 1995.
[27] V. Paxson, The Internet Traffic Archive, http://ita.ee.lbl.gov/ (2002).
[28] L. Portnoy, "Intrusion Detection with Unlabeled Data Using Clustering", Undergraduate Thesis, Columbia University, 2000
[29] M. Roesch, "Snort - Lightweight Intrusion Detection for Networks", Proc. USENIX Lisa '99, Seattle: Nov. 7-12, 1999.
[30] A. Schwartzbard and A.K. Ghosh, "A Study in the Feasibility of Performing Host-based Anomaly Detection on Windows NT", Proc. 2nd Recent Advances in Intrusion Detection (RAID 1999) Workshop, West Lafayette, IN, September 7-9, 1999.
[31] R. Sekar, M. Bendre, D. Dhurjati, P. Bollineni, "A Fast Automaton-based Method for Detecting Anomalous Program Behaviors". Proceedings of the 2001 IEEE Symposium on Security and Privacy.
[32] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, S. Zhou, A. Tiwari and H. Yang, "Specification Based Anomaly Detection: A New Approach for Detecting Network Intrusions", ACM CCS, 2002.
[33] R. Sekar and P. Uppuluri, "Synthesizing Fast Intrusion Prevention/Detection Systems from High-Level Specifications", Proc. 8th USENIX Security Symposium, Washington DC, Aug. 1999,
[34] M. Tyson, P. Berry, N. Williams, D. Moran, D. Blei, "DERBI: Diagnosis, Explanation and Recovery from computer Break-Ins", http://www.ai.sri.com/~derbi/, April. 2000.
[35] A. Valdes, K. Skinner, "Adaptive, Model-based Monitoring for Cyber Attack Detection", RAID 2000, LNCS 1907, p80-92, Springer Verlag, 2000.
[36] G. Vigna, S.T. Eckmann, and R.A. Kemmerer, "The STAT Tool Suite", Proc. 2000 DARPA Information Survivability Conference and Exposition (DISCEX), IEEE Press, January 2000.
[37] G. Vigna and R. Kemmerer, "NetSTAT: A Network-based Intrusion Detection System", Journal of Computer Security, 7(1), IOS Press, 1999.
[38] K. Yamanishi, J. Takeuchi & G. Williams, "On-line Unsupervised Outlier Detection Using Finite Mixtures with Discounting Learning Algorithms", KDD, p.320-324, 2000.
[39] G. K. Zipf, *The Psycho-Biology of Language, an Introduction to Dynamic Philology*, M.I.T. Press, 1935.