Detecting Novel Attacks by Identifying Anomalous Network Packet Headers

Matthew V. Mahoney and Philip K. Chan
Department of Computer Sciences
Florida Institute of Technology
Melbourne, FL 32901
{mmahoney,pkc}@cs.fit.edu

Florida Institute of Technology Technical Report CS-2001-2

Abstract

We describe a simple and efficient network intrusion detection algorithm that detects novel attacks by flagging anomalous field values in packet headers at the data link, network, and transport layers. In the 1999 DARPA off-line intrusion detection evaluation test set (Lippmann et al. 2000), we detect 76% of probes and 48% of denial of service attacks (at 10 false alarms per day). When this system is merged with the 18 systems in the original evaluation, the average detection rate for attacks of all types increases from 61% to 65%. We investigate the effect on performance when attack free training data is not available.

1. Introduction

An intrusion detection system (IDS) monitors network traffic, operating system events, or the file system to detect unauthorized attempts to access a system. The two most common detection techniques are signature detection, which looks for characteristics of known attacks, and anomaly detection, which looks for deviations from normal behavior, signaling a possibly novel attack. Forrest et al. (1996) showed that novel buffer overflow attacks on UNIX processes can be detected because they generate unusual sequences of operating system calls. A variety of machine learning techniques can be used to learn what is "usual". For example, a system described by Ghosh et al. (1999) uses a neural network trained to recognize normal system call sequences extracted from Solaris Basic Security Module (BSM) data and Windows-NT audit logs.

However, it is unusual to apply machine learning to network intrusion detection. The COAST survey of network IDSs (Crosbie and Price 2000) shows that most systems are rule based (similar to a firewall), requiring that the user or network administrator specify what type of traffic is allowed. For example, Bro (Paxson 1998) requires that the operator use a specialized language to describe allowable network packets. EMERALD (Newmann and Porras 1998, Porras and Valdes 1998) has a component which collects network statistics from (presumably) attack-free traffic, automating the model building somewhat, but it still requires the user to specify which statistics to collect. ADAM (Barbará, Wu, and Jajodia) is fully automated in this respect, but monitors only IP addresses, ports, and the TCP flags. Our system extends this idea to the other packet header fields.

The rest of this paper is organized as follows. In section 2, we describe how we believe attacks generate anomalies in network traffic. In 3, we describe packet header anomaly detection (PHAD), a simple and efficient system that models the set of allowable values for each field of the data link, network, and transport layer protocols. In 4, we test the system using the 1999 DARPA off-line intrusion detection test set (Lippmann et al. 2000) to show that we detect most probes and many denial of service attacks. In 5, we improve on these results by using clustering rather than a hash function to represent the set of allowable field values. In 6, we merge our results with the outputs of the 18 original systems that participated in the 1999 evaluation to show an increase in the detection rate for attacks of all types. Obtaining attack-free data for training is not always practical, so in Section 7 we investigate the effects of attacks in this data. In 8 we discuss the results and future work.

2. How Attacks Generate Anomalies

We believe that attacking network traffic contains anomalies for four reasons. First, attacks that exploit bugs in the victim's software must somehow be unusual, because any bug that is evoked by normal traffic would have long since been detected and fixed. Second, an attacker might generate anomalies through carelessness, for example, putting unusual but legal values into the TCP/IP header fields when the attack requires programming with raw sockets. Third, probes are necessarily anomalous because the attacker is seeking information that is known to legitimate users, who would normally not guess passwords or try to contact nonexistent IP addresses and ports. Fourth, an attacker may deliberately insert unusual data to confuse the IDS (Horizon, 1998).

Kendall (1999) describes a number of denial of service (DoS) attacks that exploit bugs in some TCP/IP stack implementations. For example.

- *Teardrop.* The attacker sends fragmented IP packets with fragments that overlap or have gaps, causing the victim to crash.
- *Land*. The attacker sends a TCP packet with a spoofed source address and port matching that of the destination, causing a crash.
- *POD (ping of death).* The attacker sends a fragmented IP packet that reassembles to one more than the maximum size allowed by the protocol (65,535 bytes), causing a crash.
- *Dosnuke*. The attacker sends a packet containing urgent data (URG flag set) to the Windows netbios port, causing Windows to crash.

Other attacks might open a shell to an unauthorized user (remote to local, or R2L), or may allow a local user to gain root or administrative privileges (user to root, or U2R). Our system is not designed to detect these types of attacks. R2L attacks that exploit bugs in remote servers, such as buffer overflow vulnerabilities, typically use application layer protocols, which we do not monitor. U2R attacks, which in UNIX often exploit bugs in *suid root* programs, are difficult to detect from network traffic because the input typically comes from user commands or a local file. It is difficult to monitor user commands from a telnet session, and in any case the attack could be hidden either by launching it from the local console or by using a secure shell to encrypt the session. We do not detect data attacks (violations of a written security policy, such as copying secret files), because our system has no knowledge of that policy.

3. Packet Header Anomaly Detection

Our packet header anomaly detector (PHAD) is trained on attack-free traffic to learn the normal range of values in each packet header field at the data link (Ethernet), network (IP) and transport (TCP, UDP, ICMP) layers. For the most part, the meanings of the fields are irrelevant. The system only has syntactic knowledge of the protocols, enough to parse the header, but no more. For example, it does not reassemble IP fragments or TCP streams, nor does it do any special processing for bad checksums, missing or duplicate TCP packets, etc. The only exception is that the system does compute the IP, TCP, UDP, and ICMP checksums, and replaces these fields with the computed values (normally FFFF hex).

To simplify implementation, we require all fields to be 1, 2, 3, or 4 bytes. If a field is larger than 4 bytes (for example, the 6 byte Ethernet address), then we split it into smaller fields (two 3-byte fields). We group smaller fields (such as the 1-bit TCP flags) into 1 byte field.

During training, we would record all values for each field that occur at least once. However, for 4-byte fields, which can have up to 2^{32} values, this is impractical for two reasons. First, this requires excessive memory. Second, there is normally not enough training data to record all possible values, resulting in a model that overfits the data. To solve these problem, we record a reduced set of values either by hashing the field value modulo a constant H, or by clustering them into C contiguous ranges. These variants are denoted as follows.

- PHAD-H1000 (hashed modulo 1000)
- PHAD-C32 (32 clusters)
- PHAD-C1000 (1000 clusters)

It turns out (in Sections 4 and 5) that the selection of H or C doesn't matter much because the fields which are the best predictors of attack usually have only a small range of values, and we could simply list them.

For each field, we record the number r of "anomalies" that occur during the training period. An anomaly is simply any value which was not previously observed. For hashing, the maximum value of r is H. For clustering, an anomaly is any value outside of all of the clusters. After observing the value, a new cluster of size 1 is formed, and if the number of clusters exceeds C, then the two closest clusters are combined into a single contiguous range.

Also for each field, we record the number n of times that the field was observed. For the Ethernet fields, this is the same as the number of packets. For higher level protocols (IP, TCP, UDP, ICMP), n is the number of packets of that type. Thus, $p = r/n$ is the estimated probability that a given field observation will be anomalous, at least during the training period. This estimate is also consistent with PPMC, one of the better models used to predict novel values in data compression algorithms (Bell, Witten, and Cleary, 1989).

Table 3.1 below shows the model that results after training PHAD-H1000 on 7 days of attack free network traffic (week 3 of the inside tcpdump files) from the 1999 DARPA IDS evaluation test set (Lippmann et al. 2000). The first column gives the name of the field and its size in bytes. The second column gives r and n. The third column is a partial list of the r observed values for that field after hashing. Also present, but not shown, is the time of the last observed anomaly in each field.

During testing, we fix the model (n, r, and the list of observed values). When an anomaly occurs, we assign a field score of t/p, where $p = r/n$ is the estimated probability of observing an anomaly, and t is the time since the previous anomaly in the same field (either in training or earlier in testing). The idea is that events that occur rarely (large t and small p) should receive higher anomaly scores.

Finally, we sum up the scores of the anomalous fields (if there is more than one) to assign an anomaly score to the packet.

$$\text{Packet score} = \Sigma_{i \in \text{anomalous fields}}\, t_i/p_i = \Sigma_i\, t_i n_i/r_i \qquad (1)$$

We assume a nonstationary model in which the best predictor of future events is the time since the event last occurred. This accounts for the factor *t*; as *t* increases, the probability of the event drops, so the anomaly score rises. For attacks (or benign events) consisting of many anomalous packets during a short period of time, we do not need to flood the user with alarms after the first detection.

Field (bytes)	**p = r / n**	**Hashed values**
Ether Dest Hi (3)	9/34909810	186, 192, 215...
Ether Dest Lo (3)	12/34909810	9, 88, 215...
Ether Src Hi (3)	6/34909810	186, 192, 219...
Ether Src Lo (3)	9/34909810	88 257 268...
Ether Protocol (2)	4/34909810	48, 54, 310, 864
IP Header Len (1)	1/34669966	69
IP TOS (1)	4/34669966	0, 8, 16, 192
IP Pkt Length (2)	1000/34669966	0, 1, 2, 3... 999
IP Frag ID (2)	1000/34669966	0, 1, 2, 3... 999
IP Flag/Offset (2)	2/34669966	0, 384
IP TTL (1)	10/34669966	2, 32, 60... 255
IP Protocol (1)	3/34669966	1, 6, 17
IP Checksum (2)	1/34669966	535
IP Src Addr (4)	851/34669966	0, 2, 3, 4...
IP Dest Addr (4)	853/34669966	0, 2, 3, 4...
TCP Src Port (2)	1000/27010151	0, 1, 2, 3...
TCP Dest Port (2)	1000/27010151	0, 1, 2, 3...
TCP Seq (4)	1000/27010151	0, 1, 2, 3...
TCP Ack (4)	1000/27010151	0, 1, 2, 3...
TCP Hdr Len (2)	2/27010151	80, 96
TCP Flags (1)	9/27010151	2, 4, 16, 17...
TCP Window (2)	1000/27010151	0, 1, 2, 3...
TCP Chksum (2)	1/27010151	535
TCP Urg Ptr (2)	2/27010151	0, 1
TCP Option (4)	2/1612632	36, 112
UDP Src Port (2)	1000/7565939	0, 1, 2, 3...
UDP Dest Port (2)	1000/7565939	0, 1, 2, 3...
UDP Length (2)	128/7565939	25, 27, 29...
UDP Chksum (2)	2/7565939	0, 535
ICMP Type (1)	3/90288	0, 3, 8
ICMP Code (1)	3/90288	0, 1, 3
ICMP Chksum (2)	1/90288	535

Table 3.1 PHAD-H1000 model of attack-free traffic.

4. Results on the DARPA IDS Test Set

In 1999 the Defense Advanced Research Projects Agency (DARPA) developed an off line test set for evaluating intrusion detection systems (Lippmann et al. 2000). The set consists of a 5 week run of four main "victim" machines (SunOS 4.1.4, Solaris 2.5.1, Linux Redhat 4.2, and Windows NT 4.0) on a simulated network (Ethernet, TCP/IP) with hundreds of hosts and an Internet connection through a Cisco 2514 router. This setup simulates an imaginary Air Force base. Eight participating organizations were provided with three weeks of training data, consisting of all network traffic (*tcpdump* files) just inside and outside the gateway, audit logs, nightly system file dumps from the four main victims, and BSM data from the Solaris machine. A few attacks are against machines other than the four main victims (e.g. the router or DNS server), in which case the only evidence is network traffic. The training data consists of two weeks of attack free traffic (weeks 1 and 3), used to train anomaly detection systems, and one week in which there are 43 attack instances, drawn from a set of 64 attack types, used to train signature detection systems. Most of the exploits were taken from published sources such as rootshell.com or the Bugtraq mailing list archives, and are described in (Kendall, 1999). The attacks are labeled with the date, time, victim IP address, and a brief description of the attack.

After several months, the eight developers were provided with two weeks (weeks 4 and 5) of test data containing 201 unlabeled instances of the 64 attacks, some of them modified to make them harder to detect. Each of the 18 systems (more than one could be submitted) produced a list of detections by victim IP address, date, time, and a score from 0 to 1, where higher numbers indicate greater confidence in the detection. Each system was evaluated by its detection-false alarm (DFA) curve, which is the number of correct detections at a given a false alarm rate, plotted by taking all alarms with scores above a threshold. An alarm is considered true if it correctly identifies any of the hosts involved in the attack and the time of any part of the attack within 60 seconds. Multiple detections of the same attack are counted only once. After the evaluation, the test data, attack labels, and the results from the 18 systems were made available to promote the development of new techniques in intrusion detection.

We trained each of the PHAD variants on week 3 (7 days attack free) of the inside tcpdump files, then tested the system on weeks 4 and 5 of the inside tcpdump files. We did not make any use of the outside traffic, because the inside data contains evidence of attacks both from inside and outside the network, although we miss outside attacks against the router. We did not use the week 1 (attack free) data in order to reduce computational requirements, and we did not use week 2 (labeled attacks) because we do not look for any attack-specific features. We did not use any of the BSM, audit logs, or file system data.

We used roughly the same evaluation method for our system as was used in the original evaluation, counting an attack as detected if there is at least one alarm within 60 seconds of the attack, but counting all false alarms separately. Our evaluation method differs from DARPA's in how it sorts tie scoring alarms (not specified by DARPA) in order to achieve a desired false alarm rate. Initially we tried breaking ties randomly, but this resulted in scores much lower than the official results reported by Lippmann et al. (1999) for some systems. We found that this was due to these systems reporting hundreds of alarms for a single attack (or benign event) and using binary scoring (always 1). We found that we could improve the detection rates for these systems by ranking the alarms according to the time since the previously reported alarm for the same victim IP address, so that the first alarm in each burst got the highest ranking. Then we would remove any alarm if it fell within 60 seconds of a higher ranked alarm for the same address. For example, if there were four alarms at times 1, 2, 5, and 7 seconds, then these would be ordered 1, 5, 7, 2, and the last three would be discarded since they are within 60 seconds of alarm 1.

DARPA allowed extended time limits for detecting DoS attacks (3 minutes for *selfping*, 15 minutes for the others) so that host-based systems could detect a reboot or restart. Our evaluation kept the cutoff at 60 seconds. Also, DARPA made manual corrections due to labeling errors and false alarms caused by simulation artifacts, which we did not duplicate because the details were unavailable.

To test our evaluation method, we ran it on the raw output data of two of the other high scoring participants in order to duplicate the results reported by Lippmann for a false alarm (FA) rate of 10 per day (100 over 10 days). We observed detection rates 4% to 5% higher than the official values for the sets of attacks that these systems were designed to detect. We believe this error is partly due to the use of time-ranked sorting of alarms, which should not help PHAD because tie scores are rare.

We categorized the 201 attacks in the DARPA test set according to the protocol exploited, based on the brief descriptions provided by the test set and by Kendall (1999). All of the 37 instances of 9 probing attacks except two instances of *ls* exploit protocols at the transport layer or below, so PHAD should detect them. Of the 65 instances of 18 DoS attacks, PHAD should detect 11 types (40 instances). We should also detect 3 instances of one R2L attack, the backdoor *netbus*. We will refer to these 78 instances of 27 attack types as *in-spec* attacks. However, had we participated in the original DARPA evaluation, we would not have been permitted to make such a fine distinction between detectable and undetectable attacks, as not all attacks appeared in the training set. DARPA allowed participants to classify their systems by the type of attack detected (DoS, probe, R2L, U2R, data), evidence examined (inside tcpdump, outside tcpdump, BSM, audit logs, file system dumps), and operating system (Solaris, SunOS, Linux, NT). Had we participated, we would have chosen the set of all DoS and probe attacks with inside tcpdump evidence as the best fit to our system.

Our unofficial evaluation of PHAD-H1000 at 10 FA/day detects 67 of 201 attack instances (33%), of which 53 of 78 (68%) are in-spec by our unofficial classification. We detect 29 of 37 probes (78%) and 26 of 65 DoS attacks (40%), for a combined 55 of 102 (54%) DoS/probe attacks.

We must caution that any comparison to the performance of the original participants would be biased in our favor. We had access to the test data, and even though we did not use it to train the system or write attack-specific techniques, simply having this data available to test our system helps us. For example, we knew, and the original participants did not, that there is more evidence of attacks in the inside network traffic than the outside traffic. Furthermore, we wrote our own evaluation program, which we found to be optimistic.

Nevertheless, we should point out that a 50% detection rate is considered good. The 18 systems submitted by eight organizations used a variety of techniques: signature and anomaly detection, rule based and machine learning, analyzing network traffic, BSM, audit logs, and file system dumps (Barbará, et al.; Ghosh et al. 1999; Lindqvist and Porras 1999; Neumann and Porras 1999; Porras and Valdes 1998; Sekar and Uppuluri 1999; Tyson et al. 2000; Valdes and Skinner; Vigna, Eckmann, and Kemmerer 2000; Vigna and Kemmerer 1999). Table 4.1 shows the official results of the top 4 systems in the original evaluation reported by Lippmann.

System	**In-Spec Detections**
Expert 1	85/169 (50%)
Expert 2	81/173 (47%)
Dmine	41/102 (40%)
Forensics	15/27 (55%)

Table 4.1. Official detection rates (out of the total number of attacks that the system is designed to detect) at 10 FA/day for top systems in the 1999 DARPA evaluation (Lippmann et al. 2000, Table 6).

We examined the top 20 scoring packets for PHAD-H1000 in detail. Starting with the highest score (FA indicates false alarm):

1. FA - fragmented TCP header, legal but unusual, probably due to a misconfigured machine. Normally, large IP packets are fragmented to 576 bytes (Internet) or 1500 bytes (Ethernet) because of the packet size limitations of the data link layer. This packet was fragmented to 8 bytes (the smallest possible), fragmenting the 20 byte TCP header. The program detected that the last 12 bytes were missing (e.g. the

checksum), because we make no attempt to reassemble fragmented IP packets.

2. *Teardrop* - a fragmented UDP header resulted in an anomalous header size field.
3. *Dosnuke* - the nonzero URG pointer was anomalous.
4. FA - same as 1.
5. FA (*arppoison*) - anomalous Ethernet source address. In the *arppoison* attack, a local sniffer spoofs a reply to the *ARP-who-has* packet. ARP is used to resolve IP addresses to Ethernet addresses. The attack causes the victim to incorrectly address packets, so that they are not received. This packet does not count as a detection because the DARPA scoring algorithm requires the IP address of the victim. Since an ARP packet is not IP, this information is not available in the packet.
6. FA - TOS = 0x20. This TOS (type of service) value indicates a high priority IP packet, a normal response to an SNMP request to a router. However, most systems ignore the TOS, so this field is usually 0.
7. *Portsweep* - a fragmented TCP header resulted in anomalies in the fields past the fragmentation point, in particular, the checksum. This probe sends a packet to each well known port (1-1024) to see which ones are listening. The reason for the fragmentation is not clear, possibly carelessness by the attacker or an attempt to confuse the IDS that backfired.
8. *UDPstorm* - UDP checksum error, possibly due to carelessness by the attacker. A udpstorm attack is started by sending a UDP packet to the echo server on one victim with the spoofed source address and port of the echo or chargen server of the other victim. The result is that they echo each other endlessly and waste network bandwidth. The actual storm was not detected.
9. FA (*arppoison*) - unusual destination Ethernet address in an ordinary HTTP request packet from the victim. (This detection would succeed if we reported the source IP address instead of the destination).
10. *Pod* (ping of death) - fragmented ICMP echo request packet. Some TCP/IP stacks will crash when they receive a fragmented IP packet whose total size is larger than 64K, the maximum in the IP protocol specification. Normally an ICMP packet would not be large enough to require fragmentation.
11. *Dosnuke* - nonzero URG pointer.
12. FA (*arppoison*) - unusual Ethernet source address.
13. FA - TOS = 0xC8 (high priority, high throughput) in an ICMP TTL expired message.
14. FA - unusual Ethernet destination address in an NTP (network time protocol) request.
15. FA - TCP checksum error in a FIN (close connection) packet.
16. FA - unusual Ethernet source address in an ordinary ARP packet.
17. *Portsweep* - FIN without ACK. This is a stealth technique to prevent the probe from being logged. Normally, a FIN (connection close) packet to an unopened connection will simply be dropped. Thus, any port *not* listening will send a RST packet, from which the attacker can determine which ports *are* listening.
18. FA - fragmented TCP header, same as 1.
19. *Portsweep* - TTL = 44. This could be an artifact of the simulation. It appears that initial TTL values were usually set to 32, 64, 128, or 255, then decremented at most 4 times (once per hop). In reality, 20 hops (from 64) would not be unusual.
20. FA - TOS = 0x20, normal SNMP response.

We have observed four ways in which attacks can generate anomalies.

- The anomaly exploits a bug that usually causes no problem because the input is normally rare (*dosnuke, teardrop, pod*).
- The attacking program has a bug (*udpstorm*).
- An anomaly meant to hide an attack exposes it instead (*portsweep*).
- The victim generates anomalies as a symptom of a successful attack (*arppoison*).

None of the attacks were discovered using anomalous IP addresses or ports, the usual way of detecting probes. As Table 3.1 shows, anomalous IP addresses would generate low scores due to their high r values (851-853). Port number anomalies would never occur, since all H = 1000 possible values occurred in training.

4.1. Run-Time Performance

PHAD inspects 54 bytes of each packet header (more if IP or TCP options are present). For each of the up to 25 fields (depending on the protocol), the program hashes the value and stores it in a bit vector (during training), or checks whether the bit is present. The program also computes IP, TCP, UDP, and ICMP checksums, which requires reading the application layer payload as well. The algorithm is simpler than a typical TCP/IP stack, so we would expect it to be faster as well.

On A Sparc Ultra 5-10, our implementation of PHAD-H1000 processes 2.9 Gbytes of training data (tcpdump files) and 4.0 Gbytes of test data in 30 minutes, or about 2 minutes per day of simulation time.

Memory usage is negligible. The model consists of 33 fields with a vector of H = 1000 bits each, or about 4 Kbytes.

5. Clustering Field Values

In section 4, we used a hash function to reduce the model size to conserve memory, and more importantly, to avoid overfitting the training data. We got good performance because the important fields for intrusion detection have a small *r*, so that hash collision are rare for these fields. However, hashing is a poor way to generalize continuous values such as TTL or IP packet length when the training data is not complete. A better representation would be a set of clusters, or continuous ranges. For instance, Instead of listing all possible hashes of the IP packet length (0, 1, 2,..., 999 for *r* = 1000), we list a set of ranges, such as {28-28, 60-1500, 65532-65535}.

We modified PHAD to store a list of clusters for each field, with a maximum of *C* clusters. Whenever an anomalous value is observed in training, *r* is incremented and a new cluster containing only the observed value is created, for instance, 28-28 if a value of 28 is observed. Then if the number of clusters exceeds *C*, the two closest clusters are merged. The distance between clusters is the smallest difference between two cluster elements. In the example above, the closest pair is 28-28 and 60-1500 (distance 32), so if *C* = 2, then these would be merged to form a new cluster, 28-1500.

The choice of *C* is a tradeoff between overgeneralizing (small *C*) and overfitting the training data (large *C*). We tried *C* = 32 and *C* = 1000, calling these models PHAD-C32 and PHAD-C1000, respectively.

On the DARPA test set, C32 and C1000 both outperform H1000. The results are summarized in table 5.1. The first two columns give the number of probe and DoS detections. The last two columns give the number of in-spec detections (exploiting the transport layer or below according to our unofficial classification) and the total number of detections at 10 FA/day.

Tables 5.2 and 5.3 list the unofficial detection rates for PHAD-C32, the best performing system, for all probe and DoS attacks out of the total number of instances of each type of attack, at 10 FA/day. Out-of-spec attacks (according to our unofficial classification) are shown in parenthesis, with the application layer protocol that those attacks exploit. Table 5.4 shows all R2L, U2R, and data attacks. All of these except *netbus* are out-of-spec.

PHAD	**Probe/37**	**DoS/65**	**In-spec/78**	**All/201**
H1000	29 (78%)	26 (40%)	53 (68%)	67 (33%)
C32	28 (76%)	31 (48%)	56 (72%)	72 (36%)
C1000	29 (78%)	28 (43%)	55 (71%)	70 (35%)

Table 5.1. Unofficial detection rates for PHAD at 10 FA/day

Probes Detected by PHAD-C32	**Detected**
illegalsniffer	2/2
ipsweep	4/7
ls (DNS)	(0/2)
mscan	1/1
ntinfoscan	2/3
portsweep	14/15
queso	3/4
resetscan	0/1
satan	1/2
Total Probes	**28/37 (77%)**

Table 5.2. Probes unofficially detected by PHAD-C32. Protocols shown in parenthesis denote the application layer that is exploited in out-of-spec attacks.

DoS Attacks Detected by PHAD-C32	**Detected**
apache2 (HTTP)	(1/3)
arppoison	0/5
back (HTTP)	(0/4)
crashiis (HTTP)	(1/8)
dosnuke	4/4
land	0/2
mailbomb (SMTP)	(2/4)
neptune	3/4
pod	4/4
processtable	1/4
selfping (UNIX shell)	(0/3)
smurf	5/5
syslogd	3/4
tcpreset	0/3
teardrop	3/3
udpstorm	2/2
warezclient (FTP)	(0/3)
warezmaster (FTP)	(1/1)
Total DoS	**31/65 (48%)**

Table 5.3. DoS attacks unofficially detected by PHAD-C32. Parenthesis denote out-of-spec attacks.

Other Attacks Detected by PHAD-C32	**Detected**
R2L dict (FTP, telnet, POP3)	(3/7)
R2L named (DNS)	(1/3)
R2L netbus	3/3
R2L netcat (DNS)	(1/4)
R2L ppmacro (Powerpoint)	(1/1)
R2L sendmail (SMTP)	(1/2)
R2L xlock (X)	(1/3)
U2R casesen (NT shell)	(1/3)
U2R sechole (NT shell)	(1/3)
Total R2L, U2R and Data	**13/99 (13%)**

Table 5.4. Other attacks unofficially detected by PHAD-C32. All R2L, U2R, and data attacks except *netbus* are out-of-spec.

Lippmann (1999, Table 4) lists 21 hard to detect attacks including 5 probes and 5 DoS. These are attacks for which no system in the original evaluation detected more than half of the instances. Table 5.5 shows that we improve on four of these unofficially. We did not improve on any out-of-spec attacks.

Probe/DoS Hard to Detect Attack	**Best detection (Lippmann 1999)**	**PHAD-C32 (unofficial)**
stealthy ipsweep	0/3	**1**
ls	1/2	(0)
stealthy portsweep	3/11	**11**
queso	0/4	**3**
resetscan	0/1	0
arppoison	1/5	0
dosnuke	2/4	**4**
selfping	0/3	(0)
tcpreset	1/3	0
warezclient	0/3	(0)

Table 5.5. PHAD-C32 unofficial detections of hard to detect probes and DoS attacks. (Parenthesis denote out-of-spec attacks).

An attack is classified as stealthy if the attacker takes steps to hide it from the IDS. In the case of probes (ipsweep and portsweep), this means slowing down the attack to more than one minute between packets. PHAD is immune to this technique because spreading out the attack actually increases the anomaly score due to the factor *t*, the time between anomalies.

The inside tcpdump data for week 4 day 2 is missing, but we did not adjust our scores for that. The only in-spec attack in this data is one instance of *land*, and we missed the other one.

We did not determine how the out-of-spec attacks were detected. We believe that some of these may be coincidental, especially for attacks with long durations. Some could also be due to simulation artifacts caused by generating the attack in a different environment than the one used to simulate the background traffic.

5.1. Run Time Performance

Clustering has a higher run time complexity than hashing. By storing a sorted list of clusters, lookup time is $O(\log C)$ by binary search. Worst case complexity is $O(C)$, the time to find the closest pair to merge when an anomaly is detected in training, but this occurs only rarely. In practice, our implementations of C32 and C1000 run at the same speed as H1000.

Memory requirements are minor. The C1000 model requires a list of $2C$ = 2000 4-byte numbers for each of 33 fields, for a total of about 264 Kbytes.

Neither the *H* nor *C* variants of PHAD allocate memory at run time, so they are immune to DoS attacks on the IDS that try to exhaust memory. However the *C* variants have different average and worst case run times, making them vulnerable to CPU runtime attacks. This vulnerability exists only during the training period.

6. Merging Detections

One of the goals of the DARPA evaluation is to improve the overall detection rate by combining the results of multiple systems that detect different classes of attacks. This should improve overall coverage, but since we must also combine the false alarms from different systems, it does not necessarily mean that the detection rate will be improved at a given false alarm rate.

To that end, we measured the detection rates for the 18 systems participating in the 1999 evaluation, using the raw alarm data, both alone and in combination with the alarms from our systems. To combine multiple systems, we took equal numbers of the highest scoring alarms from each system until a threshold of 100 false alarms (10 per day) was reached. Recall that we broke ties by sorting the alarms by the time since the previous alarm, with the longest intervals first, and then we removed duplicates that occurred within 60 seconds of a higher ranked alarm. When combining systems, we merged after the time-ranked sorting, but before removing duplicates, so that when two IDSs reported the same attack, we were left with only one alarm. For example, suppose we are given the two lists of alarms in Table 6.1, and the simulation starts at time 0:00.

System A

ID	Time	Victim	Score
1	1:00	pascal	0.7
2	1:03	pascal	0.6
3	1:09	pascal	0.5
4	1:16	hume	0.4
5	1:18	hume	0.3

System B

ID	Time	Victim	Score
1	1:00	pascal	1.0
2	1:02	pascal	1.0
3	1:10	pascal	1.0
4	1:15	hume	0.5
5	1:16	hume	1.0

Table 6.1. Hypothetical output of two IDSs.

In System A, we choose the alarms in the order 1, 2, 3, 4, 5, because the scores takes precedence over everything else. In system B, we choose the alarms in the order 5, 1, 3, 2, 4, because the time since the previous equally scored alarm against the same victim in the first four cases is 1:16, 1:00, 0:08, and 0:02. To merge these systems, we take alarms alternately from each system in the chosen order, i.e. 1A, 5B, 2A, 1B, 3A, 3B, 4A, 2B, 5A, 4B. Then we remove alarms 1B (duplicate of 1A) and 4A (duplicate of 5B).

We exhaustively evaluated all 2^{19} - 1 = 524,287 combinations of the 18 original systems and PHAD-C32 to find the set that detects the most attacks, whether in-spec or not, using the unofficial evaluation method just described. The official evaluation excluded out-of-spec detections in order not to penalize systems for missing attacks that they were not designed to detect. Our goal now is not to penalize the merged system for detecting attacks that it was supposed to miss. The results are shown in Table 6.2.

System	**Detections/201 (unofficial)**
PHAD-C32	72
Best combination without PHAD	123 (61%)
Best combination with PHAD	131 (65%)

Table 6.2. Unofficial detection rates, including out-of-spec detections, at 10 FA/day for merged combinations of 18 IDSs in the 1999 DARPA evaluation and PHAD-C32.

The best combination without PHAD consists of 3 high scoring systems from the original evaluation, unofficially detecting 123 of 201 attacks, or 61% including out-of-spec attacks. The best combination with PHAD-C32 includes two of these three plus two others, for a total of 5 systems. It unofficially detects 131 attacks, or 65%.

We should mention that our unofficial rescoring of the participant data found a significant number of out-of-spec detections in many systems that were not counted in the official results. The highest number unofficially detected by any system was 97.

7. Attacks in the Training Data

A practical problem in implementing any anomaly detection system is how to keep the attack-free model up to date as new hardware and software is added, possibly changing the traffic statistics. We could put the IDS back into training mode periodically, but if any attacks occur during this time, they would be missed. Not only that, but the anomalies that they generated would be part of the model, so that similar attacks would be missed later.

One approach to this problem might be to divide the training set into smaller pieces, say, one day, and merge multiple detectors each trained on a different piece. If an attack occurs during some of the training days, then the other detectors should still catch it. We could keep the model updated by rotating components, say, by having 7 components each trained on a different day of the week, and using 6 for detection while the seventh is being trained.

However, we found experimentally that this method would not work. First, we divided the 7 day training period (week 3, days 1-7) into one-day sets to train 7 PHAD-C32 models. These unofficially detect 62, 57, 59, 56, 59, 71, and 39 of 201 attacks during weeks 4 and 5, or an average of 61.9. But when we merge the sorted results, we unofficially detect only 31 attacks, compared to 72 when a single model was trained on all 7 days at one. We also tried merging pairs of models, and the results were almost always worse then the two components. The reason may be that we are generating more false alarms without detecting more attacks.

Next, we tested the effect of using training data with attacks in it. As long as the attacks are of a different type than those in the training data, then we should be able to detect them. An important question is, if there are *m* attacks in the training data, then how many attacks will we miss in the test data?

We tested PHAD-C32 on the 10 attack days using only the previous day's traffic as training. That means that for all but the first day, the training data would contain attacks. The first day (week 4 day 1) was trained on week 3 day 7. Since the DARPA test set contains 20 attacks per day drawn from 64 types, we would expect that any given attack in the test data has a (9/10)(20/64) = 28% chance of having occurred during the training period. Thus, if there is no generalization between attacks, then we should detect 28% fewer attacks than the average of 62 when using one day of attack-free training data. We actually observed 36 attacks by this method, or 42% less, suggesting that there is some generalization between the anomalies generated by different types of attacks. More specifically, each attack in the training data reduces by 42/28 = 1.5 the number of attack types that can be detected.

8. Concluding Remarks

PHAD differs from most network IDSs and firewalls in that it lacks semantic knowledge of the protocols that it analyzes (other than checksums). PHAD uses only syntactic knowledge to parse the header into fields, and then figures out which fields are important. Surprisingly, IP addresses and ports are not among them (they have large *r* values). PHAD's lack of semantic knowledge is not a disadvantage. For instance, its inability to reassemble IP fragments leads to the inadvertent detection of some attacks that deliberately fragment the TCP header to try to hide themselves. The IDS is an inviting target, so the simplicity of the algorithm should make it easier to analyze implementations for vulnerabilities.

We hypothesized that attacks generate anomalies because they are necessary to exploit bugs (possibly bugs in the IDS), or because of carelessness or lack of knowledge by the attacker. We found examples of these, and also anomalies generated by victims after the attack succeeds. All but this last type are fundamentally different from the types of anomalies studied by Forrest. Anomalies in system calls are due to the program under attack executing code either supplied by the attacker (as in a buffer overflow) or code which was never properly tested. In a network IDS, we look for untested *inputs* rather than untested *states*.

We avoid direct comparison of PHAD to the original participants in the DARPA evaluation because we had the advantage of having access to the test data during development. Although this data is not used to train the system, having it available does introduce a bias because it could influence our design and experimental protocol. Nevertheless, we believe that our system is generic enough that it would have ranked high if we had participated in the original evaluation. We unofficially detect 77% of probes, and 48% of DoS attacks, and 72% of attacks that exploit the protocols we analyze.

Since we detect four attack types that were missed by all other systems (ipsweep, portsweep, queso, dosnuke), we would expect that merging our system with the others would result in an increase in total coverage. Indeed, that is what we found. Merging PHAD with the 18 original systems increases the total coverage from 61% to 65% of all attacks according to our evaluation. Merging does not always help. We also found that a weak system can sometimes drag down a stronger one, and that merging identical systems trained on different sets does not work.

In a practical system, it is difficult to obtain attack-free training data reliably. We investigated the effects of including attacks in the training data, and estimate that each attack makes the system vulnerable to about 1.5 attack types. We also investigated shortening the training period from 7 days to 1 day to reduce the number of training attacks, and saw only a small average drop in the detection rate, from 72 to 62 for PHAD-C32.

We are concerned that the test data is not a realistic simulation. For instance, there are no checksum errors in the training data, and other fields such as TTL have unusually small sets of values. Floyd and Paxson (2001) discuss many of the problems of simulating the Internet. Tests in real systems are necessary, of course, but these have the disadvantage that the tests cannot be replicated. The DARPA IDS evaluation is not perfect, but it is a valuable resource for the systematic study of IDSs.

We had experimented with a number of variations of PHAD, none of which worked as well at the two variations described here. In one experiment, we compared the distribution of field values in the training and test data over a time window. This detected some high volume attacks that the other systems missed, but overall had lower performance. Another variation looked at anomalies across pairs of fields, but this had a slightly lower detection rate than H1000 and was much slower.

Many R2L attacks and some DoS attacks exploit faulty implementations of application layer protocols such as HTTP or SMTP. We plan to apply the principles of PHAD to detect anomalies in these protocols. Our approach will be the same; apply some minimal syntactic knowledge and let the system learn the rest of the protocols from the training data.

Acknowledgments

This research is partially supported by DARPA (F30602-00-1-0603). We wish to thank Richard Lippmann for help by reviewing earlier drafts of this paper.

References

Bell, Timothy, Ian H. Witten, John G. Cleary, "Modeling for Text Compression", ACM Computing Surveys (21)4, pp. 557-591, Dec. 1989.

Barbará, D., N. Wu, S. Jajodia, "Detecting novel network intrusions using Bayes estimators", George Mason University, to appear.

Crosbie, Mark, and Price, Katherine, "Intrusion Detection Systems", COAST Laboratory, Purdue University, 2000, http://www.cerias.purdue.edu/coast/intrusion-detection/ids.html

Floyd, S. and V. Paxson, "Difficulties in Simulating the Internet." To appear in IEEE/ACM Transactions on Networking, 2001. http://www.aciri.org/vern/papers.html

Forrest, S., S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for Unix processes", Proceedings of 1996 IEEE Symposium on Computer Security and Privacy. ftp://ftp.cs.unm.edu/pub/forrest/ieee-sp-96-unix.pdf

Ghosh, A.K., A. Schwartzbard, M. Schatz, "Learning Program Behavior Profiles for Intrusion Detection", Proceedings of the 1st USENIX Workshop on Intrusion Detection and Network Monitoring, April 9-12, 1999, Santa Clara, CA. http://www.cigital.com/~anup/usenix_id99.pdf

Horizon, "Defeating Sniffers and Intrusion Detection Systems", Phrack 54(10), 1998, http://www.phrack.com

Kendall, Kristopher, "A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems", Masters Thesis, MIT, 1999.

Lindqvist, U., and P. Porras, "Detecting Computer and Network Production-Based Expert System Toolset (P-BEST)", Proc. 1999 IEEE Symposium on Security and Privacy, Oakland CA.
http://www.sdl.sri.com/emerald/pbest-sp99-cr.pdf

Lippmann, R., et al., "The 1999 DARPA Off-Line Intrusion Detection Evaluation", Computer Networks 34(4) 579-595, 2000.

Neumann, P., and P. Porras, "Experience with EMERALD to DATE", Proceedings 1st USENIX Workshop on Intrusion Detection and Network Monitoring, Santa Clara, California, April 1999, 73-80, Website:
http://www.sdl.sri.com/emerald/index.html
Paper: http://www.csl.sri.com/neumann/det99.html

Paxson, Vern, "Bro: A System for Detecting Network Intruders in Real-Time", Proceedings, 7'th USENIX Security Symposium, Jan. 26-29, 1998, San Antonio TX.
http://www.usenix.org/publications/library/proceedings/sec98/paxson.html

Porras, P., and A. Valdes, "Live Traffic Analysis of TCP/IP Gateways", Networks and Distributed Systems Security Symposium, 1998.
http://www.sdl.sri.com/emerald/live-traffic.html

Sekar, R., and P Uppuluri, Synthesizing Fast Intrusion Prevention/Detection Systems from High-Level Specifications, Proceedings 8th Usenix Security Symposium, Washington DC, Aug. 1999.

Tyson, M., P. Berry, N. Williams, D. Moran, D. Blei, "DERBI: Diagnosis, Explanation and Recovery from computer Break-Ins", 2000, http://www.ai.sri.com/~derbi/

Valdes, Alfonso, and Keith Skinner, "Adaptive, Model-based Monitoring for Cyber Attack Detection", SRI International, http://www.sdl.sri.com/emerald/adaptbn-paper/adaptbn.html

Vigna, G., S. T. Eckmann, and R. A. Kemmerer, "The STAT Tool Suite", Proceedings of the 2000 DARPA Information Survivability Conference and Exposition (DISCEX), IEEE Press, Jan. 2000, 46-55.

Vigna., G., and R. Kemmerer, "NetSTAT: A Network-based Intrusion Detection System", Journal of Computer Security, 7(1), IOS Press, 1999.
http://citeseer.nj.nec.com/vigna99netstat.html