Learning Models of Network Traffic for Detecting Novel Attacks

by Matthew V. Mahoney and Philip K. Chan

Florida Institute of Technology Technical Report CS-2002-08

{mmahoney,pkc}@cs.fit.edu

Abstract

Network intrusion detection systems often rely on matching patterns that are gleaned from known attacks. While this method is reliable and rarely produces false alarms, it has the obvious disadvantage that it cannot detect novel attacks. An alternative approach is to learn a model of normal traffic and report deviations, but these anomaly models are typically restricted to modeling IP addresses and ports, and do not include the application payload where many attacks occur. We describe a novel approach to anomaly detection. We extract a set of attributes from each event (IP packet or TCP connection), including strings in the payload, and induce a set of conditional rules which have a very low probability of being violated in a nonstationary model of the normal network traffic in the training data. In the 1999 DARPA intrusion detection evaluation data set, we detect about 60% of 190 attacks at a false alarm rate of 10 per day (100 total). We believe that anomaly detection can work because most attacks exploit software or configuration errors that escaped field testing, so are only exposed under unusual conditions.

1. Introduction

The internet is one of the most influential innovations in recent history. Though most people use the internet for productive purposes, some use it as a vehicle for malicious intent. As the internet links more users together and computers are more prevalent in our daily lives, the internet and the computers connected to it increasingly become more enticing targets of attacks. Computer security often focuses on preventing attacks using usually authentication, filtering, and encryption techniques, but another important facet is detecting attacks once the preventive measures are breached. Consider a bank vault, thick steel doors prevent intrusions, while motion and heat sensors detect intrusions. Prevention and detection complement each other to provide a more secure environment.

How do we know if an attack has occurred or if one has been attempted? This requires sifting through huge volumes of data gathered from the network, host, or file systems to find suspicious activity. There are two general approaches to this problem: signature detection (also known as misuse detection), where we look for patterns signaling well known attacks, and anomaly detection, where we look for deviations from normal behavior. Signature detection works reliably on known attacks, but has the obvious disadvantage that it is not capable of detecting new attacks. Though anomaly detection can detect novel attacks, it has the disadvantage that it is not capable of discerning intent. It can only signal that some event is unusual, but not necessarily hostile, thus generating false alarms.

Signature detection methods are well understood and widely applied. They are used in both host based systems, such as virus detectors, and in network based systems such as SNORT (Roesch, 1999) and BRO (Paxson, 1998). These systems use a set of rules encoding knowledge gleaned from security experts to test files or network traffic for patterns known to occur in attacks. As new vulnerabilities or attacks are discovered, the rule set must be manually updated.

How do security experts discover new unknown attacks? Generally, the experts identify something out of ordinary, which triggers further investigation. Some of these investigations result in discovering new attacks, while others result in false alarms. Identifying something out of ordinary is essentially anomaly detection. From their experience, security experts learned a model of normalcy and use the model to detect abnormal events. On the contrary, rather than learned, a model of acceptable behavior can also be specified by humans as well. For example, firewalls are essentially manually written policies dictating what network traffic is considered normal and acceptable.

We desire to endow computers with the capability of identifying unusual events similar to humans by learning from experience. Classical machine learning problems are classification tasks--given examples of different classes, learn a model that distinguishes the different classes. However, in anomaly detection, we are essentially given only one class of examples (normal instances) and we need to learn a model that characterizes and predicts the lone class reliably. Since examples of the other classes are absent, traditional machine learning algorithms are less applicable to anomaly detection.

Our investigation in this paper focuses on devising and evaluating machine learning algorithms that generate models for detecting anomalies. Particularly, we concentrate on anomalies in network traffic. Our approach is unique in three respects. First, we model the application payload, a more difficult problem than modeling just IP addresses and port numbers, as most network anomaly detectors do. Second, we use a nonstationary model, in which the time since an event last occurred is significant, and the frequency of occurrence is not. Third, we develop a randomized algorithm for finding the type of conditional rules that are most useful for anomaly detection. We test our system on the 1999 DARPA intrusion detection evaluation data set (Lippmann et al., 2000), which simulates a local network under attack.

The rest of this paper is organized as follows. In Section 2, we describe related work in anomaly detection. In Section 3, we describe the LERAD algorithm (learning rules for anomaly detection). In Section 4 we describe the 1999 DARPA intrusion detection evaluation data set. In Section 5, we test LERAD on this set and compare the results with two earlier versions that use fixed rules. In Section 6, we describe the attacks in the data set and analyze how they were detected. In Section 7, we summarize.

2. Related Work

Anomaly detection is a harder problem than signature detection because signatures of attacks can be very precise but what is considered normal is more abstract and ambiguous. Rather than finding rules that characterize attacks, we wish to find rules that characterize normal behavior. Since what is considered normal could be different in different environments, a distinct model of normalcy can be learned individually. Hence, in our approach, customization to individual environment is automated via machine learning. This contrasts to manually written polices of normal behavior that require manual customization at each environment. Moreover, since the models are customized to each environment, potential attackers would find them more difficult to circumvent than manually written policies which might be less customized due to inexperienced system administrators who do not change the default parameters and policies supplied by the vendors. Much of the research in anomaly detection uses the approach of modeling normal behavior from a (presumably) attack-free training set. Because we cannot predict all possible non-hostile behavior, false alarms are inevitable. Thus, the rules must also generate a score or ranking reflecting the probability of hostility, so that alarms can be prioritized.

The idea of anomaly detection is widely attributed to Forrest et al. (1996). Forrest reasons that our own immune system can provide ideas for more effective intrusion detection techniques. Part of our immune system functions by identifying unfamiliar foreign objects and attacking them. For example, a transplanted organ is often attacked by the patient's immune system because the organ from the donor contains objects different from the ones in the patient. To reduce and control rejection, doctors utilize drugs. Based on this observation, she found that when a vulnerable UNIX system program or server is attacked (for example, using a buffer overflow to open a root shell), that the program makes sequences of system calls that differ from the sequences found in normal operation. Forrest used n-gram models, i.e. recording sequences of n = 3 to 6 calls, and matching them to sequences observed in training. A score is generated when a sequence observed during detection is different from those stored during training. Other models of normal system call sequences have been used, such as finite state automata (Sekar et al., 2001) and neural networks (Ghosh et al., 1999). Notably, Sekar et al. (2001) utilize program counter information to specify states. Though the program counter carries limited information about the state of a program, its addition to their model differs from typical n-gram models that solely rely on sequences of system calls.

A host-based anomaly detector is important since some attacks (for example, inside attacks) do not generate network traffic. However, a network-based anomaly detector can warn of attacks launched from the outside at an earlier stage, before the attacks actually reach the host, than host-based anomaly detectors. Current network anomaly detection systems such as NIDES (Anderson et al., 1995), ADAM (Barbara et al., 2001), and SPADE (2001) model only features of the network and transport layer, such as port numbers, IP addresses, and TCP flags. Models built with these features could detect probes (such as port scans) and some denial of service (DOS) attacks on the TCP/IP stack, but would not detect attacks of the type detected by Forrest, where the exploit code is transmitted to a public server in the application payload.

Network anomaly detectors estimate the probabilities of events, such as that of a packet being addressed to some port, based on the frequency of similar events seen during training or during recent history, typically several days. They output an anomaly score which is inversely proportional to probability. Anomaly detectors are typically just one component of more comprehensive systems. NIDES is a component of EMERALD (Newmann and Porras, 1998), which integrates the results with host and network based signature detectors. ADAM is a Bayes classifier with categories for normal behavior, known attacks, and unknown attacks. SPADE is a SNORT plug-in.

Most current anomaly detectors use a stationary model, where the probability of an event depends on its average rate during training, and does not vary with time. However, using the average rate could be incorrect for many processes. Paxson and Floyd (1995) found that many network processes, such as the rate of a particular type of packet, have self-similar (fractal) behavior. Events do not occur at uniform rates on any time scale. Instead they tend to occur in bursts separated by large gaps on all time scales. Hence, it is not possible to predict the average rate of an event over a time window by measuring the rate in another window, regardless of how short or long the windows are. An example of how a stationary model fails in an anomaly detector would be any attack with a large number of events, such as a port scan or a flooding attack. If the detector correctly identifies each packet as anomalous, then the user would be flooded with thousands of alarms in a few minutes.

3. Learning Rules for Anomaly Detection (LERAD)

In this section, we introduce the three ideas that we believe are key to anomaly detection: First we extend the network traffic model to include a large number of attributes, including the application payload. Second, we introduce a nonstationary model, in which the probability of an event (an attribute having some value) depends on the time of its most recent occurrence, and not on its average frequency. Third, we introduce an efficient algorithm for selecting good rules for anomaly detection from a rule space that is exponentially large in the number of attributes.

The first two ideas were developed in two earlier versions of our current system, a packet header anomaly detector, or PHAD (Mahoney and Chan, 2000), and an application layer anomaly detector, or ALAD (Mahoney and Chan, 2001). LERAD extends these ideas by replacing the fixed set of rules with an algorithm for selecting them based on the training data. All of our systems were developed and tested on the 1999 DARPA off-line intrusion detection evaluation data set (Lippmann et al., 2000), by training them on attack free network traffic and evaluating them by the number of detected attacks at a given false alarm rate.

3.1. Extending the Attribute Set

Our first system, PHAD, extendes the four attributes normally used in network anomaly detection systems (source and destination IP address, source and destination port numbers) to 33. We simply divide up the Ethernet, IP, and transport headers (TCP, UDP, or ICMP) into fields of 1 to 4 bytes, as appropriate for each protocol. In testing, we discovered that many attacks could be detected because of unusual values in these fields. In addition to IP address anomalies, we found that some attacks generate unusually small packet sizes, unusual combinations of TCP flags (e.g. urgent data, missing acknowledgements, reserved flags), IP fragmentation, and unusual TCP options. In addition, we compute checksums and compare them with the checksum fields, and we found that some attacks that generate UDP or ICMP checksum errors. Surprisingly, we did not find any attacks that generate anomalous port numbers. This might be due to the global model we use. We simply record a set of allowed values (those seen in training) for each packet header field. We do not make this set conditional on other attributes, as we do in our later models.

Our next system, ALAD, extends our network model to the application layer. Instead of modeling single packets, as in PHAD, we model incoming TCP connections to the well known server ports (0-1023). Although this misses a few attacks that exploit IP, UDP, ICMP, or higher numbered ports (such as X servers), it does (or should) catch most attacks against servers, which usually use TCP.

ALAD also introduces conditional rules. With PHAD, we assigned a probability p that an attribute (field) X would have a particular value x, and then an anomaly score of $1/p$. In other words,

$$p = \Pr(X = x)$$

A more general (and useful) model would assign a probability to a set of attributes given that another set has some particular values. The most general form is,

$$p = \Pr(X = x, Y = y, ... \mid A = a, B = b, ...)$$

We call the condition $A = a, B = b,$ .... the *antecedent*, and the result $X = x, Y = y,$ ... the *consequent*.

ALAD uses five rule forms or models.

1. Pr(source IP address | destination IP address)
2. Pr(source IP address | destination IP address, destination port)
3. Pr(destination IP address, destination port)
4. Pr(TCP flags (first, next to last, and last packet) | destination port)
5. Pr(keyword | destination port)

These models were selected because they were found experimentally to give good results individually on the DARPA IDS evaluation, out of about 15 forms that we tried. The first four models are similar to conventional anomaly detectors. Since we monitor only incoming traffic, the destination means the server under attack. The first two rule forms model the set of users of a private (password protected) service, either on a per host and per server basis. The third is intended to detect probes, attempts to access nonexistent hosts or services. The fourth detects malformed or interrupted connections. The fifth models the application layer, and detects some malformed server requests. A *keyword* is the first word (delimited by spaces or tabs) on a line in the header part of the server request (indicated by a blank line separator). For example, the keywords we observe for port 80 (HTTP) in the DARPA training data are: *Accept-Charset:, Accept-Encoding:, Accept-Language:, Accept:, Cache-Control:, Connection:, GET, Host:, If-Modified-Since:, Negotiate:, Pragma:, Referer:, User-Agent:.*

We should point out a philosophical difference between PHAD and ALAD. With PHAD, we used a machine learning approach by selecting every conceivable attribute and letting the program figure out which ones are useful. It turned out that about a third of them were. With ALAD, we used an ad-hoc approach to select a few (conditional) rules from the huge space of possibilities. This proved useful (we showed that port numbers can detect attacks in a conditional setting) but there are probably many good rules that we did not think of. With LERAD, discussed in section 3.3, we continue to model TCP connections, but we return to the machine learning approach by selecting a large number of attributes, whether we think they are useful or not, and let the algorithm figure it out. But rather than select from the small set of unconditional rules, LERAD selects from the much larger space of conditional rules. The algorithm makes use of the special form for rules in a nonstationary model, used by all three of our systems, which we will describe first before returning to LERAD.

3.2. Nonstationary Event Modeling

Before the 1995 study by Paxson and Floyd, it was widely believed that network traffic could be modeled as a stationary process, i.e. independent of time. Although we may observe short term bursts of traffic of a particular type (say, an FTP data transfer), it was believed that these events would average out if our observation window were long enough. However, this is incorrect, which is unfortunate because stationary processes are easier to model than nonstationary ones, although the problem is not insurmountable.

To illustrate how nonstationary processes might be modeled, suppose that we observe a sequence of 20 events with the following outcomes, and we wish to predict what the next one will be.

11111111110000000000

If we assume that the source is memoryless (one type of stationary source), then we count ten ones and ten zeros, and estimate Pr(1) = 1/2. However, real systems have memory, and the sequence strongly suggests a change of state midway through the sequence. Therefore, Pr(1) should be much smaller than 1/2.

Intuition suggests that recent history is a more reliable predictor of the immediate future than events that happened longer ago. Suppose that we base our prediction on just enough of the history so that each outcome is observed at least once. Then our history is:

10000000000

By counting, we estimate Pr(0) = 10/11 and Pr(1) = 1/11, which agrees more closely with intuition. Another way of modeling this is to assume that the probability of an event is inversely proportional to the time t since it last occurred. For the event 1, we have $t = 11$, so again we have Pr(1) = 1/11.

Now let us ask a different question. We did not say that 0 and 1 are the only possible outcomes. What is the probability of a novel outcome, such as 5? In this case, we can examine the entire history of $n = 20$ events and observe that $r = 2$ of them are novel, i.e. the first event, 1, and the eleventh event 0. Based on this, we would have Pr(not 0 or 1) = r/n = 2/20. In fact, this is known as the PPMC method of estimating the probability of novel events in some data compression models (Bell, Witten, and Cleary, 1989). It is not the only model of novel events, and it has some shortcomings (e.g. when $n = 1$), but it is what we use in our systems.

Our nonstationary model makes two assumptions. First, by our example, we see that the frequency of an event is irrelevant, since it is only the time since the last occurrence that matters. Thus, by our assumption, all observed values are equally likely. For this reason, we are only interested in novel events, which have a lower probability. By our second assumption, we model this probability as r/n, where there are r unique values out of n observations. Thus our nonstationary model consists of r, n, and the set of observed values. For this example we have the set of observed values {0, 1}, and the values $r = 2$ and $n = 20$.

Now let us consider the case where we train our system on known attack-free data and test it on data possibly containing attacks. It might not be possible to guarantee real data to be attack free, but those are the conditions in the DARPA evaluation, so we will use them for now. Suppose that we train on the same 20 events as before, and then begin monitoring for attacks, and we observe the following.

11111111110000000000   0000022222

In this example, each of the "2" events are anomalous. We would assign an anomaly score of $1/p =$ n/r = 25/2 = 12.5 to the first "2", and 0 to the rest because the event is no longer novel at this point. However, this would not be correct because if "2" signals an attack, future attacks of the same type would be missed. To avoid this problem, we should not add "2" to the model. But by our nonstationary argument, we know that subsequent values of "2" are highly likely. If "2" is non-hostile (we don't know), then we would flood the user with false alarms.

The solution to this dilemma is to introduce the factor t, the time since the last anomaly, as we had originally. Thus, we assign an anomaly score of tn/r, where there are n training observations, r unique values observed, and t events since the last anomaly. In the above example, $t = 16$ for the first "2" (the previous anomaly was in training), and $t = 1$ for the others. The value $n/r = 20/2 = 10$ is fixed at the end of training. Thus, the first "2" has an anomaly score of 160, and the others have a score of 10. By setting the alarm threshold appropriately, only the first anomaly would generate an alarm.

It is possible that a packet or TCP connection could violate more than one rule. In this case, we add the anomaly scores:

Anomaly score = $\Sigma\ tn/r$

where the summation is over all rules that produce anomalies. This seems obvious and gives good results in practice, but there is no theoretical justification that we are aware of for using a summation. Since anomaly scores represent inverse probabilities, theory suggests we should multiply the scores if the rules are independent. We know that the rules are probably not independent, but without specifying the dependency more precisely, we cannot say that this is the optimal way to combine scores.

3.3. Learning Conditional Rules

In Section 3.1 we saw that conditional rules have the general form

Pr(X = x, Y = y, ... | A = a, B = b, ...)

In Section 3.2 we saw that the rules of interest are those which restrict X, Y, ... to a set of values observed in training. Also, we can reduce the consequent to one attribute if we treat sets of attributes like (X,Y) as single attributes, such as

P(X,Y = x,y | A = a, B = b, ...)

Thus, we can rewrite our rules to have the form:

If A = a, B = b, ... then X = x1, x2, x3...

where x1, x2, x3... are the values observed in training.

Our task is to find "good" rules of this form. For PHAD and ALAD, the rules were hand picked, either by eliminating the antecedent (in PHAD) or trying a few rules that we believed would give good results (ALAD). The reason we did this is that the rule space is huge. If there are m attributes, and each one has k_j possible values, $j = 1...m$, then the number possible rules is:

$$\Sigma_{\, i = 1 \ldots m} \, \Pi_{\, j = 1 \ldots m,\, j \neq i} \, (k_j + 1)$$

That is, for each attribute *i*, all of the remaining *m* - 1 attributes can appear in the antecedent with either one of the k_j allowed values for the j'th attribute, or a "don't care". As an example, for PHAD, with 33 attributes, ranging from 8 to 32 bits, there are about 8 x 10^{171} possible rules.

Fortunately the problem is not as hard as it seems. We know that a good anomaly rule is one that is rarely violated and that generates a high score when it is. We know that the anomaly score is *tn/r*, and that *n* and *r* can be determined from the training data. Therefore we want rules with high *n/r*. A rule has high *n* if a large number of training examples satisfies the antecedent. It has small *r* if the set of allowed values is small. Suppose that the rule "if A = a and B = b then X = x1, x2, or x3" has large *n* and small *r* (here *r* = 3), and pick a training example at random. If there are *N* training examples, then it is likely (probability *n/N*) that A = a and B = b. If we pick *two* examples at random and both satisfy the antecedent (which happens with probability n^2/N^2) , then since *r* is small, it is likely (probability ≥ 1/*r*) that the two values of *X* will be the same as well.

This suggests the following strategy. Pick two training examples at random. Then for each set of matching attributes, form rules where one attribute is the consequent and the others are conditions in the antecedent. For example, if A, B, and C match, with values *a*, *b*, and *c*, then we have rules like "if A = a and B = b then C = c", "if B = b and C = c then A = a", and so on. We can also consider subsets of the matching attributes, such as "if B = b then A = a", "B = b", and so on. If there are *m* matching attributes, then there are $m(2^{m - 1})$ possible rules, so we may want to consider strategies for limiting the number of rules generated when *m* is large.

It may turn out that some initial rules turn out to be poor when fully trained. For example, the rule "if A = 1 then B = 2" after training might be "if A = 1 then B = 2, 3, 4, 5..." with large *r*, or it may be that A is rarely 1 (small *n*). Fortunately, both cases can be determined fairly quickly by sampling the training set rather than testing every example to compute *n* and *r* exactly.

It may also turn out that after training that some rules are redundant. For example, suppose we have the following three rules:

- R1: A = 1
- R2: B = 2
- R3: if A = 1 then B = 2

Since A is always 1 (at least in training), there is no need to test for this condition in rule R3, so it is equivalent to rule R2. We could remove either of these rules and have the same constraints on the test data. Also, because rules R2 and R3 would generate anomalies at the same time (for example, if A = 1 and B = 3), this would affect the score. Ideally, each consequent should be counted only once (we believe).

Our strategy for removing redundant rules is to apply a coverage test to a sample of the training data. If a rule does not predict any values not already predicted by other rules, then we discard it. For instance, rule R2 predicts every value of B. Since rule R3 does not predict any additional values, we discard it. Since the rules that are discarded depends on the order in which we select them, we start with the rules with the highest *n/r* (estimated on the sample) first. In case of a tie, we pick the rule with fewer conditions in the antecedent first, in this case R2.

We mentioned that *r/n* is just one way to estimate the probability of a novel event, and not necessarily the best. In particular, it does not distinguish between the case where all of the novel events happen at the beginning of training and the case where the novel events are spread throughout the training data. In the latter case, we would expect the novel events to continue, generating a lot of false alarms. To eliminate these types of rules, we simply discard rules that generate anomalies near the end of the training period (e.g. in the last 10% of the data). Stated another way, we know that any anomaly in the training data is a false alarm, so we eliminate those rules that are expected to generate most of them.

3.4. The LERAD Algorithm

We are now ready to describe the LERAD algorithm.

1. Generate rules with initial *n/r* = 2/1 on L randomly selected pairs of training examples.
2. *Coverage test*: remove rules that do not predict any values (on a small sample S) not already predicted by rules with higher *n/r* (estimated on S).
3. Train on the full training set, removing any rule that generates anomalies in the last E examples.

As an example, suppose that we have the following training data, with examples T1 through T6. For the coverage test, we use T1-T4 as our sample S, although in general we would sample uniformly, which requires an extra pass through the training data. We let L = 1 and E = 1.

Example	**A**	**B**	**C**	**D**
T1 (sample)	1	2 (R2)	3	4
T2 (sample)	1	2 (R2)	3	5
T3 (sample)	1	3 (R1)	4	5
T4 (sample)	0	2 (R2)	3	5
T5	1	5	3	5
T6 (end)	1	3	3	5
T7 (start of test)	1	8	3	5
T8 (test)	1	8	3	5

Table 1. Example training data for LERAD (with values marked during the coverage test)

In step 1, we pick two training examples at random, say T1 and T2. These may be selected either from the S (T1-T4) or from the full set (T1-T6). In our implementation of LERAD, we generate rules by selecting up to 4 matching attributes in random order, and then generating one rule for each match, with the first match being the consequent and the remaining matches being added to the antecedent. For example, T1 and T2 have 3 matching attributes A, B, and C. If we select these in random order, say B, C, A, then the rules would be:

- R1: B = 2
- R2: If C = 3 then B = 2
- R3: If A = 1 and C = 3 then B = 2.

After selecting L pairs of examples, we apply the coverage test. In our example, R1-R3 are the only rules. We first train them on S, updating *n*, *r*, and the list of observed values. We then sort them by decreasing n/r, or by increasing number of conditions in the antecedent if equal.

- R2: if C = 3 then B = 2 (n/r = 3/1)

- R1: B = 2 or 3 (n/r = 4/2)
- R3: if A = 1 and C = 3 then B = 2 (n/r = 2/1)

Next we mark the values in S starting with the rule with highest *n/r*, which is rule R2. It marks column B for T1, T2, and T4. Rule R1 is next. It marks the only remaining value in column B not already marked, in T3. Rule R3 would have marked column B for T1 and T2, but because these are both already marked, the rule is removed. This leaves us with the following rules:

- R1: B = 2 or 3
- R2: if C = 3 then B = 2

After training on the full set (T1-T6) we have:

- R1: B = 2, 3, 5 (n/r = 6/3)
- R2: if C = 3 then B = 2, 3, or 5 (n/r = 5/3)

Because T6 is in the last E examples, then we remove rule R2 because the anomalous value (3) was observed there for the first time. This is not anomalous to rule R1 because the value 3 was already seen in T3. So our final set of rules is just {R1}.

Once we are done training and start testing, we assign anomaly scores of $\Sigma\, tn/r$ for each violated rule, where t is the time since the last anomaly for that rule, whether during training or testing, and the summation is over all rules that were violated. For example, suppose that T7 in Table 1 is the start of testing. This example violates rule R1, which has n/r = 6/3. The last anomaly for R1 occurred at time T5, so $t = 2$. Thus, the anomaly score is tn/r = 2(6/3) = 4. If we had kept rule R2 with $n/r = 5/3$ and $t = 1$ (since T6 is the last anomaly), then it would also be anomalous and generate a score of 1(5/3) = 1.67 for a total anomaly score of 5.67.

Since training has finished, we do not update the rules, but we do record that the last anomaly time for these rules is now T7. Thus, T8 is still anomalous, and rule R1 still has $n/r = 6/3$. However, t is now 1 (since T7 also violated R1), so the anomaly score is now 2 rather than 4.

3.5. PHAD, ALAD, and LERAD

Although we have focused on LERAD, this system evolved from our earlier work with PHAD (packet header anomaly detection) and ALAD (application layer anomaly detection). All three are nonstationary models, in that they assign an anomaly score of $\Sigma\, tn/r$ summed over each of the violated rules. The three systems differ in the attributes that they monitor, and the types of rules that they represent.

PHAD monitors 33 packet header fields for both incoming and outgoing traffic. There is a fixed rule for each field, which is a set of allowed values (or more precisely, a set of ranges if there are more than 32 distinct values). Thus, one rule might be "TOS = 0, 8, 16, or 192", where TOS is one of the fields in the IP header. In this example, n is the number of IP packets and r is 4. There are additional rules for Ethernet, TCP, UDP, and ICMP.

ALAD monitors the incoming side of TCP connections to well known server ports on the home network. ALAD has 5 fixed rule forms, although there may be many rules for each form. For instance, one form is "if destination port = x then keyword = $y1, y2, y3... yr$". There is one rule for each observed value of x in the training data. In this example, n is the number of times that port number x occurs in training, and r is the number of values of y. Because a connection may have several keywords (the first word on a line in the header), a single rule may be applied several times to a single connection. If more than one keyword is anomalous, then their scores are added.

LERAD monitors the same TCP connections as ALAD, but it uses the rule learning algorithm described in Section 3.4. Also, it monitors more attributes and parses the application payload differently. It just takes the first 8 words of the payload (some of which may be empty strings) as 8 attributes. Thus, each rule is applied only once in testing. A rule may have a conjunction of 0 to 3 conditions in the antecedent and exactly one attribute in the consequent. An example is "if F2 = AP then F1 = S or AS", where F1 and F2 are the first and next to last TCP flags in the connection, and A, P, and S are the ACK, PSH, and SYN flags. In this example, there is one condition in the antecedent, n is the number of connections where the antecedent F2 = AP is satisfied in training, and r is 2. Unlike ALAD, there is not a rule for every possible value of F2.

4. The DARPA IDS Evaluation

We developed and tested PHAD, ALAD, and LERAD using the 1999 DARPA intrusion detection evaluation data (Lippmann et al., 2000). The data is from a simulated local network for an imaginary air force base under attack as shown in Figure 1. The object is to detect as many attacks as possible given off-line data dumps of the network traffic and other data.

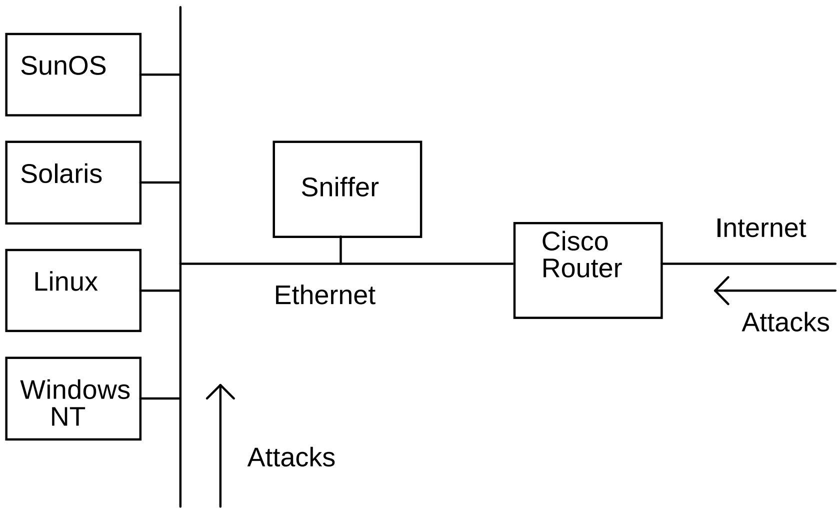

Figure 1. The 1999 DARPA IDS simulation.

In the original evaluation in 1999, eight organizations submitted 18 systems for testing. During the first phase, the participants were provided with 3 weeks worth of sniffed network traffic from inside and outside the local network (both sides of the router), audit logs and nightly file system dumps from four “victim” hosts running different unpatched operating systems, and BSM (Basic Security Module) logs from the Solaris machine, which traces system calls. The data consists of two weeks of attack-free background traffic for training anomaly detection systems (weeks 1 and 3) and one week of labeled attacks. (week 2) for testing and for training signature detection systems. Most of the attacks were taken from security mailing lists and cracker-oriented web sites to exploit known vulnerabilities, and are described by (Kendall, 1999).

After the systems were developed, they were tested on two additional weeks (10 days) of data in which the system was attacked 201 times, including 63 attacks of types not seen during week 2. Other attacks were varied to make them stealthy or hard to detect, for example, using slow port scanning or varying the exploit code to elude signature detection. The intrusion detection systems were evaluated by the number of attacks detected and number of false alarms. An attack is counted as detected if the system correctly reports the IP address of either the attacker or target, and the date and time of the attack within 60 seconds of any portion of the attack. The systems were required to output a score for each alarm so that a threshold could be used to vary the false alarm rate. Participants were allowed to declare before the evaluation what types of attacks they were designed to detect. Attacks are categorized by the host operating system, the type of data where evidence of the attack exists (inside or outside network traffic, audit logs, BSM, file system dumps), and type of attack (probe, denial of service (DOS), remote to local (R2L), or user to root (U2R)). Detections of attacks outside the declared categories do not count. Multiple detections of the same attack are counted only once. The results of the evaluation for the top four performing systems when the alarm threshold is set to 100 (10 per day) as reported by Lippmann are shown in Table 2. These systems may use a variety of techniques: both signature and anomaly detection, and both host and network based methods.

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

Table 2. Top official results of the 1999 DARPA IDS evaluation, showing number of detections out of the number of attacks that the system is designed to detect, at 100 false alarms (Lippmann et. al., 2000).

After the 1999 evaluation, the 5 weeks of data were released, including the truth labels for the 201 attacks (date, time, IP addresses, attack description) in weeks 4 and 5.

5. Experimental Results

We evaluated PHAD, ALAD, and LERAD by training them on 7 days of attack free inside traffic from week 3, and evaluating them on weeks 4 and 5 using the same criteria and false alarm rate as in the original DARPA evaluation. We also post-process the alarms by removing duplicates (same IP address) within 60 seconds of each other, keeping only the highest scoring alarm. This usually improves the performance of any IDS because it reduces the number of false alarms (which count individually) without reducing the number of detections (of which only one would count). All but a few attacks show some evidence in the inside network traffic, so to simplify the results we just report all detections. The inside network traffic for one day (week 4, day 2) is missing, so we omit the 12 attacks during this period, leaving 189 of the original 201. We also discovered by examining the test data that one attack (an *apache2* attack) was not labeled. We labeled this attack in our evaluation, bringing the number of attacks that our systems are designed to detect to 190.

The training data from week 3 is 7 days, containing about 12 million IP packets for PHAD, or 35,455 incoming server TCP connections for ALAD and LERAD. The test data (weeks 4 and 5) contain about 20 million IP packets, or 178,099 TCP connections (including incomplete connections during attacks).

Table 3 summarizes the parameters used for PHAD, ALAD, and LERAD, and the results.

Parameter	**PHAD**	**ALAD**	**LERAD**
Modeled data	All packet headers	Incoming server TCP connections	Incoming server TCP connections
Attributes	33 fields from Ethernet, IP, TCP, UDP, and ICMP packet headers	5: source and destination IP address, destination port, TCP flags, keywords	23: date, time, IP address bytes, ports, duration, length, 3 TCP flags, first 8 words
Rules	33 fixed, unconditional on each field	5 conditional fixed rule forms selected manually	Learned, conditional
Anomaly score	*tn/r*	*tn/r*	*tn/r*
Training instances, N	12 M	35 K	35 K
Test instances	20 M	178 K	178 K
Initial rule set			About 1100 from L = 1000 pairs
Rules after coverage test			77-85 using |S| = 100 samples
Final number of rules	33	5 forms	56-66 after discarding anomalies in last 10% (E = 0.1N = 3545).
Detected attacks (out of 190) at 100 false alarms	54 (72 including TTL artifact)	60	114.1 ± 2.3 (averaged over 10 runs)

Table 3. Parameters and results summary for PHAD, ALAD, and LERAD.

Table 4 shows how the number of attacks detected depends on the false alarm rate as the alarm threshold is varied. For rates up to 200 false alarms (20 per day), LERAD alone detects the most attacks. At higher rates, the best result is obtained by merging all three systems. To merge systems, we select equal numbers of the highest ranking alarms from each system and discard duplicate alarms. An alarm is considered a duplicate if there is a higher ranked alarm identifying the same target IP address and the time is within 60 seconds. Figure 2 shows the same results graphically.

False Alarms	**PHAD**	**ALAD**	**LERAD (avg.)**	**PHAD + ALAD**	**PHAD + LERAD**	**ALAD + LERAD**	**All 3**
5	5	10	4	6	3.2	**7**	6
10	9	13	**15.2**	14	9.2	17	11.6
20	21	19	**48**	21	22.8	25.8	23.6
50	33	42	**86.6**	44	54.6	59	49.8
100	54	60	**114.6**	73	89.4	93.6	85.2
200	56	66	**117.4**	96	110.6	107.8	**117.4**
500	56	72		108	123.6	120.4	**134.8**
1000	56			110	133.4	126	**144**
2000	62			111	142.2		**147**
5000	86			125	146.8		**151.6**

Table 4. Unofficial number of attacks detected (out of 190) by PHAD, ALAD, LERAD on the 1999 DARPA IDS evaluation test data, and their combinations as the false alarm rate is varied. Blank entries indicate where no measurement was taken because our implementations did not generate enough false alarms, i.e. LERAD generates less than 200 false alarms.

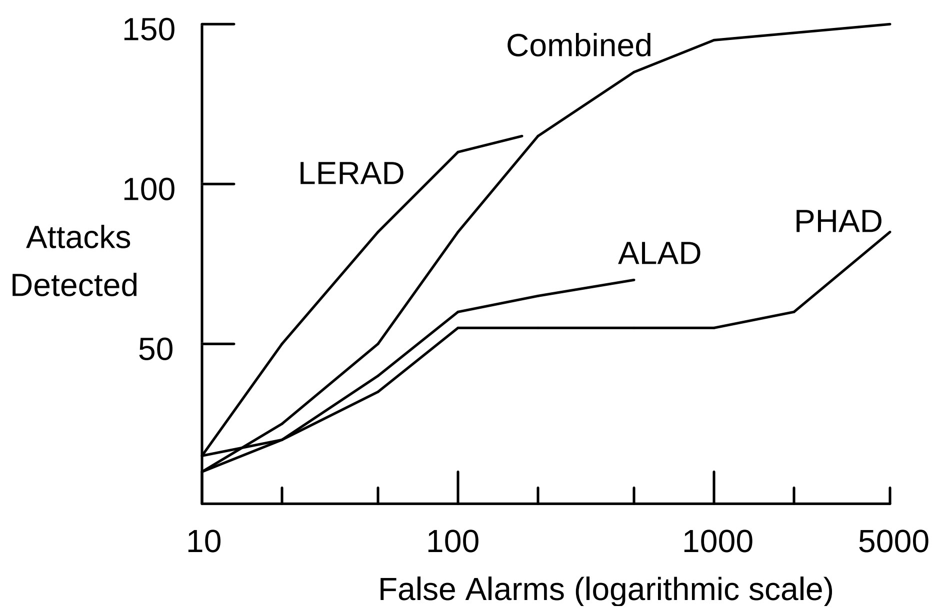

Figure 2. Detection-false alarm graph of Table 4.

5.1. PHAD Results

The complete results for PHAD and ALAD are described in our other papers (Mahoney and Chan, 2001, Mahoney and Chan, 2002), so we only summarize the results here. PHAD initially detected 72 of 190 attacks, or 38%, at 100 false alarms (10 per day). We examined the anomalies that generated the alarms, and discovered that the most common anomaly was in the TTL (time to live) field, most likely a simulation artifact. The TTL field is a counter that is decremented each time an IP packet is routed in order to prevent infinite routing loops. The DARPA evaluation used real hosts for the target, but simulated background traffic for the local network and Internet by modifying IP addresses from locally synthesized traffic. We believe the TTL anomalies were the results of the real attacking hosts being a different real distance from the targets than the real background generating hosts.

When we remove the TTL field from PHAD, it detects 54 of 190 attacks, or 28% (with 100 false alarms). As expected, many of these attacks exploit implementation flaws at the network and transport layers. Most of these are probes or denial of service (DOS) attacks. The most common anomalies resulting in detections occur in the IP fragment field, TCP flags, and IP source and destination addresses. Of the 33 fields besides TTL, 12 generate anomalies that detect at least one attack.

5.2. ALAD Results

ALAD (Mahoney and Chan, 2002) was tested alone and in combination with PHAD. Because PHAD and ALAD differ in the types of attacks they detect, it is possible to merge the results into one larger system that detects more attacks than either one by itself. ALAD detects 60 attacks by itself (at 100 false alarms) and 73 when combined with PHAD without TTL. The two systems are combined by taking equal numbers of top scoring alarms from each system and removing duplicates (within 60 seconds). The systems can be combined because ALAD is designed to detect mostly R2L attacks in the application payload. An R2L (remote to local) attack is where the attacker exploits a vulnerability in a server to execute commands on the target, although not necessarily as root. Usually this is accomplished by sending a malformed server request, such as a buffer overflow, in the application payload.

5.3. LERAD Results

LERAD is the most successful of our anomaly detection systems, detecting an average of 114 out of 190 attacks, or 60%, at 100 false alarms (10 per day). It detects a broad range of attacks (probe, DOS, R2L, U2R). Because LERAD examines only TCP, it misses a small number of attacks that exploit IP, UDP or ICMP protocols. Other than these, LERAD detects most of the attacks detected by PHAD and ALAD.

Our version of LERAD extracts 23 features from each incoming TCP connection: date, time, last two bytes of the destination IP address as two attributes (the first two are fixed, since we examine only incoming connections), four bytes of the source IP address (as four attributes), source and destination port numbers, TCP flags for the first, next to last, and last TCP packets (3 attributes), duration in seconds, total number of bytes transmitted in the application payload, and the first 8 words in the data, delimited by white space (as 8 attributes). We find that LERAD effectively weeds out attributes that have little or no predictive value from the rule set, such as the date and time; thus there is little harm done by adding too many attributes.

In step 1, the rule building phase we select 1000 to 10,000 pairs of training instances and generate up to four rules from each pair. Selecting L = 1000 pairs, we typically generate about 1050 to 1150 rule candidates. With L = 10,000 pairs, we generate about 5600-6200 rule candidates, but this does not lead to any more detections than the smaller set. We used L = 1000 pairs for our results. Also we find no significant difference in the detection rate whether we sample the pairs of training instances from the full training set or from the smaller set S of instances selected for the coverage test. For our results we took pairs from S to simplify the implementation.

In step 2, the coverage test, we sample |S| = 20 to 100 training instances from the full set and use these to estimate *n/r* and remove rules that do not predict any values in this sample not already predicted by a better rule (higher *n/r*). We find that using fewer than 10 samples or more than a few hundred results in fewer detections. The reason that using too many samples is bad may be that too many rules with poor coverage are kept. As evidence, we found that we could partially offset the loss of detections with a large sample size by imposing the more stringent requirement during the coverage test that a rule mark at least two values instead of one. We sorted rules with equal *n/r* by placing rules with fewer antecedent conditions first, although doing this does not have a significant effect on the number of detections. The result of the coverage test is to reduce the number of candidate rules from 1050-1150 to about 80-85, or if we had used 10,000 pairs, from 5600-6200 to about 110-130. We used |S| = 100 samples for the coverage test.

In step 3, we train the rules on the full training set (N = 35,455 instances from week 3), and remove any rule that generates an anomaly in the last E = 0.1N = 10% of the training data. This is an experimental maximum, producing more detections than using the last 5% or 20% to remove rules. This step reduces the rule set by about 30%. Most of the rules have *n* at least 10% of the training set and $r < 10$. Most rules have one or two conditions in the antecedent.

Since LERAD is a randomized algorithm, it produces a different result for each run. In ten runs, we observed 112 to 118 detections (average 114.1, standard deviation 2.3, standard error 0.7) using 56 to 67 rules (average 61.2). We found no significant correlation between the number of rules and number of attacks detected over these runs. Appendix A shows a typical rule set and a list of detected attacks for one run.

5.4. Run Time Performance

PHAD, ALAD, and LERAD are experimental programs which read the inside *tcpdump* files from the DARPA IDS evaluation data set. The data consists of one file per simulated day, totaling 2.9 gigabytes of training data (about 12 million packets) and 4.0 gigabytes of test data (about 20 million packets). We focused our software development on maximizing the number of detections rather than for speed and memory efficiency. Nevertheless, execution time is on the order of a few minutes on modern hardware for all systems, and memory requirements are quite reasonable, a few kilobytes at most to represent the rule set.

PHAD was implemented as a 400 line C++ program. It runs in 6 minutes under Solaris on a Sparc Ultra 60 with a 450 MHz 64-bit processor and 512 MB memory and 4 MB cache. This is about 23 seconds of CPU time per simulated day. The processing rate is 95,000 packets per second for training and 73,000 packets per second for testing.

ALAD and LERAD were implemented in two parts, one to reassemble the packets into TCP streams, and a second to perform the algorithm. The reassembly is done by a 400 line C++ program, which runs (one time) for 17 minutes under Windows Me on a 750 MHz Duron processor with 256 MB of memory. The output of this program is 35,455 TCP training connections stored in a 20 megabyte text file, and 178,101 connections in a 40 megabyte test file. ALAD, a 90 line Perl program, processes this data in 60 seconds. LERAD, a 400 line C++ program, runs in 30 seconds, out of which 6 seconds are used to train and construct a rule set and 24 seconds to process the test data.

6. Analysis of Detected and Missed Attacks

In this section, we examine each attack in detail, and how it was detected or missed by PHAD, ALAD, and LERAD. Because LERAD is randomized, we examine five runs and report the average number of detections. Attack descriptions are due to (Kendall, 1999) and examination of the test data.

The attacks in the 1999 DARPA IDS evaluation are based on known vulnerabilities at the time of the evaluation. Most of the attacks are from published sources. In most cases, patches to the operating system or servers were available to thwart the attacks at the time, but these were deliberately not applied so that the attacks could succeed. By using anomaly detection on this data, we simulate the present day case of detecting attacks that exploit vulnerabilities that have not yet been discovered.

The attacks used in the DARPA evaluation are classified as probes, DOS, R2L, U2R, and Data, based on a more detailed classification scheme by (Kendall, 1999). The evaluation includes a security policy which prohibits probes, which might not otherwise be considered attacks. The policy also defines Data attacks as unauthorized copying of secret data or its unencrypted transmission over the network by authorized users. PHAD, ALAD, and LERAD have no knowledge of the security policy. Table 5 summarizes the attacks detected by each system at 100 false alarms, with the LERAD results averaged over 5 runs.

Attack Type	**Number**	**PHAD**	**ALAD**	**LERAD (5 runs)**
Probe	37	22 (59%)	6 (16%)	23.2 (63%)
DOS	63	24 (38%)	19 (30%)	36.6 (58%)
R2L	53	6 (11%)	25 (47%)	34.8 (66%)
U2R/Data	37	2 (5%)	10 (27%)	20.0 (54%)
Total	**190**	**54 (28%)**	**60 (32%)**	**114.6 (60%)**

Table 5. Number of attacks of each type detected by PHAD, ALAD, and LERAD at 100 false alarms.

In the following sections, we describe each attack in detail and how it was detected. Most attacks have several instances which exploit the same vulnerability, but may have different sources and targets, and may differ in other details, such as what is done to the target once the attack succeeds. In some cases, the attack may be "stealthy", modified in ways intended to make it harder to detect. The number of instances includes the mislabeled *apache2* attack and does not include attacks for which the inside network traffic is missing (week 4 day 2).

6.1. Probes

A probe is any attempt by a potential attacker to gather information in preparation for an attack. Probes are prohibited by the simulation's security policy. Table 6 shows the number of probes detected by each system and the anomalies that led to detection.

Probe	Number	PHAD	ALAD	LERAD	How Detected
illegalsniffer	2	2	1	0	Coincidental false alarms
ipsweep	7	4	0	1.2	Small Ethernet packet
ls_domain	2	0	0	2	Port number, payload
mscan	1	1	1	1	Destination address, port, payload
ntinfoscan	3	0	2	3	Outgoing connection, payload
portsweep	15	10	0	10	FIN without ACK, dest port, source
queso	4	3	0	3	TCP flags
resetscan	1	0	0	1	Source address, duration
satan	2	2	2	2	Destination ports, payload
Total	**37**	**22**	**6**	**23.2**	
Percent		**59%**	**16%**	**63%**	

Table 6. Probes detected by PHAD, ALAD, and LERAD (average) with 100 false alarms.

Illegalsniffer. A compromised host on the local network is run in promiscuous mode to sniff Ethernet traffic. It is detectable only because it makes reverse DNS lookups to resolve sniffed IP addresses (which a more careful attacker could have avoided doing). DNS is normally a UDP protocol, so ALAD and LERAD do not detect it. PHAD detects both attacks by coincidence. Because there are multiple victims (all hosts on the local network) and the attack is prolonged (hours), any alarm for any host during this period would be counted as a detection.

IPsweep. The attacker scans a block of IP addresses using ICMP ECHO REQUEST packets (*ping*). ALAD and LERAD do not monitor ICMP, so they do not detect it. PHAD detects an unusually small Ethernet packet size (52 bytes) in 4 of the 7 attacks.

Ls_domain. The attacker spoofs a backup DNS server to make a DNS zone transfer request, obtaining a list of all local hostnames and IP addresses. PHAD and ALAD miss it. LERAD consistently detects both attacks because the request is made on TCP port 53 rather than the usual UDP port 53. The anomaly varies in each run. In 70% of the detections, the TCP port number is anomalous. In the other 30%, the third word (^@^A) is anomalous in the context of the fourth or fifth word being empty.

Mscan. This is a system administration tool used to test for many well known vulnerabilities. PHAD, ALAD, and LERAD all detect it. LERAD generates hundreds of duplicate alarms (which are removed during postprocessing) due to the various tests. The highest scores result from anomalous destination addresses.

NTinfoscan. This probe gathers information about a Windows NT host similar to *mscan*. LERAD consistently detects every attack with many duplicate alarms, mostly in the TCP flags and application payload. In some cases, LERAD detects outgoing connections on a server port, or an incoming RST-ACK packet (client connection refused). In the application text, LERAD detects a missing carriage return before a linefeed in HTTP requests. There are also anomalies due to destination addresses and ports. ALAD detects two instances by the unusual HTTP keyword “HEAD”.

Portsweep. The attacker tests a range of TCP ports to see which ones have servers listening. LERAD detects 10 of the 15 attacks, usually because they use a stealth technique called FIN scanning. In this probe, the attacker sends an unsolicited FIN packet, a request to close the connection, although none was opened, and waits for a reply, either a FIN-ACK or a RST if no server is listening. A SYN packet (request to open) would accomplish the same thing, but a FIN normally does not result in a log entry in the target. LERAD (and PHAD) detect the probe because there is no ACK flag to acknowledge the previous data packet, as would normally be the case. There are also some anomalies involving destination addresses and ports, which would still be detected in the absence of FIN scanning.

Queso. The attacker identifies the operating system and version of the target by sending a series of unusual or malformed packets. Different operating systems respond in characteristic ways because the exact response is not specified by the protocols. One of the four attacks is against the router and not visible in the inside traffic, but LERAD detects the other three by anomalous TCP flag combinations: FIN without ACK (also detected by PHAD), SYN + FIN, and SYN + the two reserved flags. There is also a SYN packet containing application data, which is allowed by the TCP protocol but rare.

Resetscan. The attacker tests for listening ports by sending RST packets, which are not normally logged. LERAD detects it by an anomalous source address in 4 of 5 runs, and in one case by a probe to the FTP port having a duration of 0 seconds.

Satan. This is a system administration tool that tests for multiple vulnerabilities as with *mscan*. Likewise, it is easily detected by PHAD, ALAD, and LERAD with multiple alarms. Most involve destination port numbers, but ALAD and one run of LERAD detect the anomalous “QUIT” command in an

HTTP connection.

6.2. Denial of Service (DOS) Attacks

A denial of service attack is one which degrades or disables a server, host, or network. The two most common approaches are to flood the target with data, or to send malformed data causing the target to crash due to a bug. Table 7 shows the number of attacks detected by each system.

DOS Attack	Number	PHAD	ALAD	LERAD	How Detected
apache2	4	2	4	4	TCP options, payload, source addr.
arppoison	5	0	0	1.8	Incomplete TCP connections
back	4	0	0	4	Payload, some by duration, length
crashiis	7	0	6	7	Payload, interrupted connections
dosnuke	4	4	0	4	URG flag, interrupted connections
land	1	0	0	0	
mailbomb	3	0	3	3	Payload (lowercase commands)
neptune	4	0	0	3	Dest. port, source, coincidence (1)
pod	4	4	0	0	Fragmented IP packets
processtable	3	1	1	3	Source address
selfping	3	0	0	0	(no network traffic generated)
smurf	5	4	0	0	ICMP checksum, source address
syslogd	4	3	0	0	Short IP packet size, source address
tcpreset	3	0	1	2	Improper TCP open/close, source
teardrop	3	3	0	1.2	IP fragments, interrupted TCP
udpstorm	2	2	0	0	UDP checksum
warez	4	1	4	3.6	Source address
Total	**63**	**24**	**19**	**36.6**	
Percent		**38%**	**30%**	**58%**	

Table 7. Denial of service attacks detected by PHAD, ALAD, and LERAD (average) with 100 false alarms.

Apache2. This attack exploits the inability of some versions of the Apache web server to handle very long HTTP requests. A typical attack contains multiple requests each with thousands of lines and looking something like this:

```
GET / HTTP/1.1
User-Agent: sioux
User-Agent: sioux
User-Agent: sioux
User-Agent: sioux
...
```

PHAD detects anomalies in the TCP options field in two attacks, possibly an idiosyncrasy of the attacking host (real or simulated). ALAD detects anomalous source addresses, as well as one attack where the anomalous command "x" appears in place of "GET / HTTP/1.1". LERAD detects all four attacks (including the mislabeled one) by various anomalies in the request, such as "HTTP/1.1" or "sioux". In the training data, the version is always HTTP/1.0. Other alarms are due to improperly closed TCP connections caused by the server crashing.

ARPpoison. An attacker who has compromised a host on the local network disrupts traffic by listening for "ARP-who-has" packets and sending forged replies. ARP (address resolution protocol) is used to resolve IP addresses to Ethernet addresses. Thus, the attacker disrupts traffic by misdirecting traffic at the data link layer. PHAD detects some anomalous Ethernet addresses, but because the packets do not contain IP addresses, the alarms do not meet the DARPA criteria for detection. LERAD detects an average 1.8 of 5 attacks, mostly due to incomplete TCP connections with long durations or filled with null bytes.

Back. The attacker sends an HTTP request "GET ///////////////..." with 6000-7000 slashes, which causes some versions of the Apache web server to consume excessive CPU time. LERAD detects all four attacks, usually because the third word (normally the HTTP version) is missing after it truncates the input to 1000 characters. A few attacks are detected by excessively long lengths or durations.

CrashIIS. This crashes the IIS web server on Windows NT with a very long, malformed request. "GET ../../../../.." LERAD detects the 7 attacks in a manner similar to *back*.

Dosnuke. The attacker crashes Windows ("blue screen of death") by sending urgent data to the NetBIOS port, exploiting a bug. Normally urgent data in a TCP connection is rare. It is intended to send data to the front of the queue at the receiver. PHAD and LERAD detect all four attacks, usually by the TCP URG flag being set. LERAD also detects incomplete TCP connections or unusually long durations.

Land. This attack crashes SunOS 4.1 by sending a spoofed TCP SYN packet with the source address equal to the destination address. PHAD, ALAD, and LERAD miss this attack.

Mailbomb. This attack floods a user with thousands of junk emails. ALAD and LERAD detect all three attacks because the SMTP "mail" command is lowercase. It is normally uppercase but not required to be.

Neptune. This is also known as a "SYN flood" or "half open" attack. The attacker floods the target with SYN packets with spoofed source addresses, causing it to exhaust memory and refuse connections until the spoofed connections time out. In Solaris 2.6, sending 20 spoofed packets to each port every 10 minutes causes it to refuse connections for one hour after the attack stops. A distributed attack of this type was used to shut down several major websites (CNN, Yahoo, Amazon) in 2000, and was used by the Code Red worm to attack a White House web server. Servers can now defend against this attack by encrypting TCP state information in the acknowledgment field and reconstructing it when the client responds with the first data packet, rather than saving state information.

LERAD detects 3 of 4 attacks. One of these is actually a coincidental *back* attack that occurs at the same time against the same target. The others are due to anomalous destination ports (attacking nonexistent servers) and occasional anomalous source addresses.

POD. This attack, also known as "ping of death", crashes some older operating system (but none of the DARPA hosts) by sending an oversize fragmented IP packet that reassembles to more than 65,535 bytes, the maximum allowed by the IP protocol. It is called "ping of death" because some older versions of Windows 95 could be used to launch the attack using "ping -l 65510". PHAD detects the attack, because the IP packets are fragmented, which is rare in normal traffic. ALAD and LERAD miss the attack because they do not monitor ICMP.

Processtable. This attacks exhausts the UNIX process table by flooding a server with requests. LERAD detects all of the attacks by anomalous source addresses.

Selfping. This attack crashes Solaris with a *ping* command to the local host. It is issued locally and generates no network traffic, so it is not detected.

Smurf. This is a distributed network flooding attack initiated by sending ICMP ECHO REQUEST packets (*ping*) to a broadcast address with the spoofed source address of the target. The target is then flooded with ECHO REPLY packets from every host on the broadcast address. PHAD detects 4 of 5 attacks: one anomalous source address and 3 ICMP checksum errors. The checksum errors could be bugs in the program that was used to simulate what a *smurf* attack should look like in the DARPA simulation. ALAD and LERAD do not monitor ICMP and did not detect the attack.

Syslogd. This attack crashes the *syslogd* remote logging service by sending a UDP packet with a spoofed source IP address that cannot be resolved by DNS. PHAD detects 3 of 4 attacks, two unusually short IP packets and one anomalous source address. ALAD and LERAD do not monitor UDP.

TCPreset. This attack listens for TCP SYN packets on a compromised host on the local network and immediately sends a spoofed RST (connection refused) packet, disrupting traffic. LERAD detects two attacks by an anomalous source address and by lone TCP packets without SYN and FIN/RST to open and close the connections.

Teardrop. This attack reboots the Linux host by sending a fragmented IP packet that cannot be reassembled because of a gap between the fragments. PHAD detects all the IP fragments, which rarely occur in normal traffic. LERAD detects 2 of 3 attacks due to unrelated TCP connections on the target having unusually long durations and not being closed when the target reboots.

UDPstorm. An attacker floods the local network by setting up a loop between an *echo* server and a *chargen* or another *echo* server by sending a UDP packet to one server with the spoofed source address of the other. PHAD detects a UDP checksum error in both initiating packets (but does not detect the actual storm). ALAD and LERAD do not monitor UDP.

Warez. This is a security policy violation in which an FTP server is used to upload (*warezmaster*) or download (*warezclient*) illegal software. ALAD detects 4 attacks and LERAD 3.6 by anomalous source addresses.

6.3. Remote to Local (R2L) Attacks

An R2L attack is where an unauthorized user gains the ability to execute commands on the target. These attacks usually exploit software errors in servers or improperly configured systems to gain access. The attacking code normally occurs in the application payload, so we would expect these attacks to be detected by ALAD and LERAD, but not by PHAD. Table 8 confirms this.

R2L Attack	Number	PHAD	ALAD	LERAD	How Detected
guess passwd	10	0	3	10	Source address, dest. port, payload
framespoofer	1	0	1	0	Payload
ftpwrite	2	0	2	2	FTP upload, dest. port, payload
httptunnel	2	0	0	0	
imap	2	0	0	2	Destination port, source address
named	3	0	0	3	Destination port
ncftp	5	3	5	4	Source/destination address, payload
netbus	3	0	2	3	Source address, unclosed TCP
netcat	4	0	4	3.4	Source address, destination port
phf	3	0	2	3	Source address, payload
ppmacro	3	0	1	1	Source address
sendmail	2	2	2	2	Payload, source address
snmpget	4	0	0	0	(no inside network traffic)
sshtrojan	3	0	3	1.2	Source address
xlock	3	1	0	0	Source address
xsnoop	3	0	0	0.2	Coincidental false alarm
Total	**53**	**6**	**25**	**34.8**	
Percent		**11%**	**47%**	**66%**	

Table 8. R2L attacks detected by PHAD, ALAD, and LERAD (average) at 100 false alarms.

Password Guessing. In the DARPA set, these attacks are variously known as *guesstelnet*, *guessftp*, or *guesspop* depending on the target. An attack may try a few common passwords, such as *guest* or repeat the user's name, or may try every word in a dictionary (*dict*). ALAD detects 3 of 10 attacks. LERAD consistently detects all ten, mostly by anomalous source addresses and destination ports, but occasionally by the use of lowercase "user" and "pass" FTP commands.

Framespoofer. This attack delivers a malformed HTML message by email with a hidden frame to exploit a bug in the email client. Only ALAD detects it with the anomalous keyword "Content-Transfer-Encoding:" in the message.

FTPwrite. This attack exploits an improperly configured FTP server in which the home directory is not write protected. The attacker uploads a file *.rhosts* with contents "+ +", and is then able to *rlogin* as user *ftp* without a password. ALAD and LERAD detect both attacks. ALAD detects an anomalous source address and an upload on port 20 (FTP), which is used only for downloads in training. LERAD detects the upload and also an anomalous port 513 (*rlogin*). In one run, it detects the anomalous string "/root/.rhosts" (on the FTP control port), and in another, the string "+" (on the FTP data port).

HTTPtunnel. This is a backdoor which evades a firewall (if one were to be used) by appearing to be a web browser to communicate with the attacker. It is not detected.

IMAP. This is a buffer overflow attack on the mailbox server in Linux. Buffer overflow vulnerabilities often exist in *C* programs that use functions like *strcpy*() or *gets*() which do not check the length of the input string. If the string overflows the array (buffer) into which it is written, it can overwrite the return address on the stack. Then when the currently executing function returns, it jumps to the overwritten address instead. In a typical attack, this would be the address of a short string of machine code, supplied as part of the input, which opens a shell to the attacker. Since *imap* runs as root on the target, the attacker would be able to execute arbitrary commands as root.

LERAD detects 2 of the 3 attacks by anomalous source addresses (30% of the time) and anomalous destination port 143 (*imap*) for the other 70%. A destination port anomaly is probably not realistic, but source addresses would be because an *imap* server is not public; it requires a password to retrieve mail.

Named. This is a buffer overflow in the DNS server. Instead of opening a root shell, the published version of the exploit opens an X client running as root on the target. LERAD detects all 3 attacks, either because destination port 53 is anomalous (the exploit uses TCP rather than UDP) or it detects a long string of null bytes in the application payload.

Ncftp. This attack uses an FTP server's data port to gain local network access to other servers, such as *auth* and SMTP. ALAD and LERAD detects 4 of 5 attacks by anomalous source and destination addresses. ALAD also detects keyword anomalies on ports 21 (FTP) and 113 (*auth*).

Netbus. This is a backdoor on the STMP (mail) port. ALAD detects 2 of 3 attacks by anomalous source addresses. LERAD detects all 3, 2 by source address and one by an unclosed TCP connection.

Netcat. This is a backdoor disguised as a DNS client. Two of the attacks are actually the break-in and setup phases. ALAD and LERAD detect the attacks mostly by anomalous source addresses and anomalous use of TCP port 53 (normally UDP).

Phf. This attack exploits a badly written CGI script distributed by default with older Apache web servers. A vulnerable server is attacked by sending it the URL "http://*target*/cgi-bin/phf?Qalias=x%0a*command*", which causes *command* to be executed by *target*. (Newer versions of Apache have fixed the bug and log the attacker's IP address). ALAD detects 2 attacks, both by anomalous source addresses and one by an extra null byte which mysteriously appears as a keyword. LERAD detects all 3 attacks because the third word is missing from the HTTP request, a characteristic of HTTP/0.9 which is used only by very old clients but still accepted by servers.

PPmacro. This is a trojan PowerPoint macro which is delivered as an email attachment. ALAD and LERAD detect 1 of 3 attacks by an anomalous source address.

Sendmail. This is an SMTP buffer overflow which gives root access. It is one of the few attacks that had to be written specially for the DARPA simulation because no exploit code had been published. PHAD detects both attacks by anomalous source addresses. ALAD detects both by source address and by the anomalous keyword "Sender:" LERAD detects both attacks because the first word is MAIL instead of HELO or EHLO, although this is legal SMTP protocol.

SNMPget. This is an outside attack on the Cisco router which does not generate any traffic visible on the inside network. It is not detected.

SSHtrojan. This is a fake *ssh* (secure shell) client which captures the password of a user who unknowingly tries to log in with it. ALAD detects the 3 attacks by anomalous source addresses on port 22 (*ssh*). LERAD detects one attack by an anomalous source address.

Xlock. This is a fake *xlock* screensaver that captures passwords. It can be started remotely on any UNIX host with an open X server (set by typing *xhost +*). PHAD detects one attack by an anomalous source address. ALAD and LERAD do not detect it because they only monitor well known ports (0-1023), and X normally runs on port 6000.

Xsnoop. This attack monitors keystrokes on any host with an open X server. It is not detected because ALAD and LERAD do not monitor port 6000. (One of 5 runs of LERAD produces a coincidental detection).

6.4. User to Root (U2R) and Data Attacks

A U2R attack is one in which an attacker who already is able to execute nonprivileged commands (legitimately or not) exploits a flaw in the operating system to execute commands as another user, usually root or administrator. In UNIX, this is usually done by exploiting a vulnerability in a *suid root* program to open a shell running as root. Examples are *eject*, *fdformat*, *ffbconfig*, *loadmodule*, *perl*, *ps*, and *xterm*. All but *perl* and *loadmodule* are buffer overflows. *Anypw*, *casesen*, *sechole*, and *yaga* are Windows NT exploits. SQL*attack* exploits a bug in a database application running as a restricted shell to escape to a user level. *NTFSdos* requires physical access to the target to bypass the operating system's file protections by booting from a floppy disk and copying or modifying the hard disk. The only data attack, *secret*, is where an authorized user copies or transmits secret data in violation of a security policy.

A network intrusion detection system is not designed to detect U2R or data attacks. These are best detected by host based systems, by monitoring the system calls of the programs under attack, or monitoring file systems. The DARPA IDS evaluation set provides this data, which was used by many of the original participants, but we did not use it. In theory, it is possible to monitor *telnet* sessions for signs of a U2R attack, but it is very difficult to model normal sessions, and impossible if the session is encrypted using *ssh*.

Nevertheless ALAD and LERAD detect many U2R attacks. Usually they detect anomalous source addresses during the *telnet* session or the FTP session used to upload the exploit code, or simply that the FTP server is being used to upload files when it was only used for downloads in training. (This is detected because the first TCP flags on the FTP data port is SYN-ACK instead of SYN). Occasionally LERAD detects anomalies in the exploit code itself as it is being uploaded on the FTP data port. *NTFSdos* does not directly generate any traffic, but is sometimes detected because of interrupted TCP connections when the target is rebooted. Either the duration is unusually long, or there is no FIN or RST flag to close the connection. Table 9 shows the U2R and Data attacks detected.

U2R Attack	Number	PHAD	ALAD	LERAD	How Detected
anypw	1	0	0	1	Source address
casesen	3	0	3	2.8	FTP upload (3), source address (1)
eject	2	0	1	1	FTP upload, source address
fdformat	3	1	2	1.8	FTP upload, source address
ffbconfig	2	0	1	1	Source address
loadmodule	2	0	0	0	
ntfsdos	2	0	0	1.8	Interrupted TCP connection
perl	4	0	0	2	Source address
ps	3	0	0	2	Source address
sechole	2	0	1	1.8	FTP upload (0.2 payload), source
sqlattack	2	0	0	1	Source address
xterm	3	0	1	1.4	FTP upload (0.2 payload), source
yaga	4	1	1	2.4	FTP upload, source address
secret (data)	4	0	0	0	
Total	**37**	**2**	**10**	**20**	
Percent		**5%**	**27%**	**54%**	

Table 9. U2R and Data attacks detected by PHAD, ALAD, and LERAD (average) at 100 false alarms.

6.5. Poorly Detected Attacks

We have demonstrated that it is possible to merge the outputs of two intrusion detection systems (such as PHAD and ALAD) and detect more attacks than either one by itself at the same combined false alarm rate. In general, an anomaly detection system would not be used by itself, but in combination with signature detection for known attacks. In addition, a network based system might be used in combination with a host based system. Merging systems does not always work, however. If one system performs poorly, then it may drag down the other. Also, they must detect different types of attacks. For example, ALAD + LERAD does not perform as well as LERAD alone because the attacks they detect are similar and their detection rates are quite different.

We wish to test whether our systems could be combined with the systems from the original DARPA evaluation (some of which use host based or signature techniques) to increase the total number of detections. For this to happen, the distributions of detected attacks must be different. In particular, we wish to test whether our systems detect attacks that were poorly detected in the original evaluation.

In the original 1999 evaluation there were 21 attack types (77 instances) which were poorly detected by all 18 of the original participants. Lippmann et. al. define an attack as “poorly detected” if none of the participants detect more than half of the instances with 100 false alarms. We find that PHAD and LERAD are about equally likely to detect a poorly detected attack as any other attack. The poorly detected attacks make up 38% of the total set, and about 37% of the attacks detected by PHAD or LERAD. Only ALAD does worse on the poorly detected attacks than the others, and then only slightly. Out of the 60 attacks it detects, 20 (33%) of them are poorly detected attacks.

Table 10 summarizes the results for the poorly detected attacks. The column “best orig.” is the maximum official number of detections (blank if zero) for any of the 18 original systems at 100 false alarms as reported by Lippmann et. al. (2000, Table 4). For example, no system detected more than one of two instances of *ls_domain* or any of the three stealthy *ipsweep* attacks. The columns PHAD, ALAD, and LERAD are the unofficial number of attacks detected at 100 false alarms according to our own measurements. The values for LERAD are averaged over 5 runs using different random number seeds.

We caution against comparing our systems directly to the original participants. The original evaluation was blind (no access to test data), and our evaluation is unofficial. In addition, participants may have excluded some attacks by design, so that they would not count even if detected. The number of attacks listed in the table includes the missing data (week 4 day 2) which was available to systems that monitored outside network traffic or used host based techniques.

Poorly Detected Attack	**Number**	**Best orig.**	**PHAD**	**ALAD**	**LERAD**
stealthy ipsweep **(Probe)**	3		1		0.2
ls_domain	2	1			2
stealthy portsweep	11	3	9		7
queso	4		3		3
resetscan	1				1
arppoison **(DOS)**	5	1			1.8
dosnuke	4	2	4		4
selfping	3				
tcpreset	3	1		1	2
warezclient	3			3	2.6
ncftp **(R2L)**	5		3	5	4
netbus	3	1		2	3
netcat	4	2		4	3.4
snmpget	4				
sshtrojan	3			3	1.2
loadmodule **(U2R)**	3	1			
ntfsdos	3	1			1.8
perl	4				2
sechole	3	1		1	1.8
sqlattack	3				1
xterm	3	1		1	1.4
Total poorly detected	**77**	**15**	**20**	**20**	**43.2**
Total of all attacks	**201**		**54**	**60**	**114.6**
Percent poorly detected	**38.3%**		**37.0%**	**33.3%**	**37.6%**

Table 10. Poorly detected attacks detected by the best original system in the 1999 evaluation (Lippmann et. al., 2000) and by PHAD, ALAD, and LERAD (average of 5 runs) at 100 false alarms (unofficial). The last row shows the percentage of detected attacks that were in the original set of poorly detected attacks.

6.6. Categories of Anomalies

We might ask why anomalies should signal attacks. In a host based IDS, such as Forrest's, anomalies are indicated by unusual sequences of system calls. We might expect this for some attacks, such as buffer overflows, where the program is executing code not written by the original programmer.

However, many of the attacks that we detect are not of this type, in particular, probes and DOS. Also, many attacks, including most R2L and U2R attacks, are detected by anomalies in the input to the target, which detect the attack even before the target enters an anomalous state. Why should such anomalies indicate hostility rather than just unusual but legitimate behavior?

To answer this question, we categorize the anomalies into five groups according to what is being modeled. Two of these are familiar from traditional network and host based anomaly detection systems. Traditional network systems model user behavior in the form of IP addresses and ports, looking for unfamiliar clients accessing a service. Host based systems look for signs that the program is in an unusual state after an attack. However, instead of monitoring system calls, we monitor output over the network.

The three new categories are related to modeling inadequately tested software. In one case, the target program has a vulnerability (a bug) which the attacker exploits with unusual input on which the target was never tested. In the second case, the attacking traffic differs in some arbitrary way from normal traffic because the attack was not tested in the target's environment. In the third case, the attacker deliberately inserts anomalies in an attempt to attack the IDS with traffic on which it was probably not tested.

Table 11 summarizes the five categories of anomalies and the attacks they detect (at least some of the time) based on the analysis we did in Sections 6.1 to 6.4.

Anomaly Type	Attacks Detected or Partially Detected (out of 56 types)	Total
User behavior (source address, FTP upload)	**Probe**: portsweep; **DOS**: apache2, neptune, processtable, syslogd, warez; **R2L**: guess, imap, ncftp, netbus, netcat, phf, ppmacro, sendmail, sshtrojan, xlock; **U2R**: anypw, casesen, eject, fdformat, ffbconfig, perl, ps, sechole, sqlattack, xterm, yaga	27
Induced by successful attack	**DOS**: apache2, arppoison, back, crashiis, dosnuke, tcpreset, teardrop; **U2R**: ntfsdos	8
Pattern related to attack (bug in target)	**Probe**: ls_domain, mscan, ntinfoscan, queso, satan; **DOS**: apache2, back, crashiis, dosnuke, pod, smurf, teardrop; **R2L**: ftpwrite, named, ncftp, netcat; **U2R**: sechole, xterm	18
Pattern unrelated to attack (bug in attacking program)	**Probe**: ipsweep; **DOS**: apache2, mailbomb, syslogd, udpstorm; **R2L**: guess, framespoofer, netbus, phf, sendmail	10
Attempt to evade IDS	**Probe**: portsweep	1

Table 11. Anomaly categories and the attacks that they detect or partially detect .

The following are examples of each anomaly category.

- **User behavior.** As with conventional network anomaly detection, we model the range of source IP addresses that are typically used to access a host or service. We are suspicious of anyone new, especially if the service is private or password protected. We include in this category FTP uploads on a server that is normally only used for downloads, which is how most of the U2R exploits are detected. (The actual anomaly is a server initiated connection on port 20, the FTP data port, as indicated by the TCP flags). While this may not be realistic outside the DARPA simulation, a network IDS is not designed to detect U2R attacks anyway, so we could consider these to be bonus detections.
- **Induced by a successful attack.** These are anomalies output by the target as symptoms of a successful attack, analogous to the anomalous system calls detected by a host based system. ALAD and LERAD do not monitor outgoing traffic, but they do detect interrupted TCP connections when the target crashes due to a DOS attack.
- **Patterns related to the attack.** These anomalies exploit vulnerabilities in the target due to software errors. The reason these errors exist is because they were not caught in field testing because the inputs

required to invoke the error rarely occur normally. For example, it is rare (but legal) to fragment IP packets, so if there is a bug in the TCP/IP stack related to reassembly, it goes undetected. *Teardrop* exploits such a bug, but it must use a rarely seen pattern to do so.

- **Patterns unrelated to the attack.** Some attacks have anomalies because the attacker did not go to the effort of making the attack resemble normal traffic, which varies with each environment and is difficult for the attacker to know. For example, FTP and SMTP allow either uppercase or lowercase commands, but most client programs use uppercase. The *sendmail* and one of the FTP password guessing attacks are discovered because they use lowercase commands. These anomalies could be considered bugs in the attacking program, because the exploits could be easily modified to hide them if the attacker knew the environment.
- **Evasion.** The DARPA simulation did not make much use of the techniques outlined in Ptacek and Newsham (1998) to evade or attack a network IDS, such as IP fragmentation, short TTL expirations, overlapping TCP segments, bad checksums, and so on. Such techniques probably would have been caught by PHAD if they had tried to hide an R2L or U2R attack that might have otherwise escaped notice. The only example of a backfired attempt at evasion is *portsweep*, where the attacker used FIN packets (without an accompanying SYN packet or ACK) to prevent servers from logging the scan.

To summarize, the five types of anomalies are:

- User behavior - the client is unfamiliar.
- Induced - the target is in an unusual state after an attack.
- Related pattern - the target has a bug because it was tested only under normal conditions.
- Unrelated pattern - the attack has a bug because it was not tested in the target's environment.
- Evasion - the attacker guesses that the IDS has bugs that show up under abnormal conditions.

7. Conclusions and Future Work

We described a network anomaly detection system that is unique in three respects. First, it uses a large number of attributes in order to model program behavior in addition to user behavior. Second, it uses a nonstationary model in which the time since an event is significant and the average frequency is not.

Third, it efficiently finds a small number of good rules from the huge set of possibilities. The system performs well on the DARPA IDS evaluation data set, detecting a broad range of attacks, including those that were poorly detected in the original evaluation. We identified three new categories of anomalies related to inadequate software testing and attempts to exploit it.

There are some obvious minor improvements we could make to LERAD. We could extend the model by adding IP, UDP, and ICMP attributes, as well as attributes appropriate for binary application protocols like DNS. Our analysis of the application payload is limited as well (first 8 words), and could be extended. LERAD should also be appropriate for host based systems by analyzing audit logs or BSM (system calls) to detect R2L and U2R attacks. The design of LERAD as a flexible machine learning algorithm should make such systems just a matter of extracting attributes.

Our system is limited in that it has not been tested in a live environment. This is an obvious drawback, but not one which is easy to resolve. In order to conduct a reproducible evaluation, the traffic must be recorded and published, which raises privacy and ethical issues. The DARPA evaluation was simulated, but this raises questions about artifacts due to simulation errors (such as the TTL field) or overly clean background traffic that might make attacks easier to detect. It is very difficult to simulate the "crud" found in real traffic (Floyd and Paxson, 2001). One possibility may be to add real traffic (with sensitive data removed) to the DARPA data set to add background noise.

Furthermore, we have assumed that attack free traffic is available for training. This would not be true in a real environment. We expect that having attacks in the training data would mask their detection in the test data. We have done some preliminary work in this area and found that when LERAD is trained on data containing attacks (week 2), that there is a 29% decrease in the detection of novel attacks and a 33% decrease in the detection of repeat attacks (at 100 false alarms).

A common strategy for anomaly detection (e.g. NIDES and SPADE) is to compare short term behavior (the current event) with long term behavior (hours, days, or weeks), under the assumption that the number of attacks in the training data will be reasonably small. This type of adaptive model does not have explicit training and test periods, and should keep up with changes to the system as software, hardware, and users are added to the network. Although LERAD uses an explicit training period (with two passes), we believe that an online anomaly detection system that uses the principles we have learned from LERAD is feasible in a real environment. We plan to pursue work in this direction.

Acknowledgments

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

References

ANDERSON et al. (1995), Detecting unusual program behavior using the statistical component of the Next-generation Intrusion Detection Expert System (NIDES), Computer Science Laboratory SRI-CSL 95-06, http://www.sdl.sri.com/papers/5/s/5sri/5sri.pdf

BELL, TIMOTHY, IAN H. WITTEN, JOHN G. CLEARY (1989), Modeling for Text Compression, ACM Computing Surveys (21) 557-591.

BARBARÁ, D., N. WU, S. JAJODIA (2001), Detecting Novel Network Intrusions using Bayes Estimators, Proc. First SIAM International Conference on Data Mining.

FLOYD, S. AND V. PAXSON (2001), Difficulties in Simulating the Internet, IEEE/ACM Transactions on Networking (9) 392-403.

FORREST, S., S. A. HOFMEYR, A. SOMAYAJI, T. A. LONGSTAFF (1996), A Sense of Self for Unix Processes, Proc. IEEE Symposium on Computer Security and Privacy.

GHOSH, A.K., A. SCHWARTZBARD, M. SCHATZ (1999), Learning Program Behavior Profiles for Intrusion Detection, Proc. 1st USENIX Workshop on Intrusion Detection and Network Monitoring.

KENDALL, KRISTOPHER (1999), A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems, Masters Thesis, Massachusetts Institute of Technology.

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

MAHONEY, M., AND P. K. CHAN (2001), PHAD: Packet Header Anomaly Detection for Identifying Hostile Network Traffic, Florida Tech. technical report CS-2001-04.

MAHONEY, M., AND P. K. CHAN (2002), Learning Nonstationary Models of Normal Network Traffic for Detecting Novel Attacks, Florida Tech. technical report CS-2001-06.

NEUMANN, P., AND P. PORRAS (1999), Experience with EMERALD to DATE, Proc. 1st USENIX Workshop on Intrusion Detection and Network Monitoring, 73-80.

PAXSON, VERN AND SALLY FLOYD (1995), The Failure of Poisson Modeling, IEEE/ACM Transactions on Networking (3) 226-244.

PAXSON, VERN (1998), Bro: A System for Detecting Network Intruders in Real-Time, Proc. 7'th USENIX Security Symposium.

PTACEK, T. H. AND T. N. NEWSHAM (1998), Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection, http://www.robertgraham.com/mirror/Ptacek-Newsham-Evasion-98.html

ROESCH, MARTIN (1999), Snort - Lightweight Intrusion Detection for Networks, Proc. USENIX Lisa '99.

SEKAR, R., M. BENDRE, D. DHURJATI, P. BOLLINENI (2001), A Fast Automaton-based Method for Detecting Anomalous Program Behaviors. Proc. IEEE Symposium on Security and Privacy.

SPADE (2001), Silicon Defense, http://www.silicondefense.com/software/spice/

Appendix A. Typical LERAD Run

Listed below is a typical rule set generated by LERAD after training on 7 days of attack-free network traffic from week 3 of the DARPA IDS evaluation data. This set has 56 rules and detects 117 out of 190 attacks with 100 false alarms (10 per day). Rules are ordered by decreasing *n/r*, which is shown for each rule. The attributes are:

- DATE as month/day/year
- TIME as hour: minutes: seconds
- SA3, SA2, SA1, SA0: 4 bytes of the source IP address
- DA1, DA0: last two bytes of the destination IP address
- SP, DP: source and destination port numbers
- F1, F2, F3, TCP flags for the first, next to last, and last incoming packet. Each letter shows that the flag is set as follows: 1, 0 (reserved flags), U (urgent data), A (acknowledgment), P (push), S (sync, open connection), F (finish, close connection), R (reset, connection refused). A dot (.) by itself

indicates that no flags are set.

- DUR: duration in seconds from first to last packet.
- LEN: length of application data transmitted in bytes.
- W1-W8: first 8 words of application data, shown with a preceding dot (.) so that empty words can be seen. Words are delimited on white space (linefeeds, spaces, tabs) and truncated to 8 characters. The symbol ^M^ indicates a carriage return and linefeed. The symbol ^@ indicates a null byte (ASCII 0).

Table A1. Typical LERAD rule set.

1 28882/2 if F2=.AP then F1 = .S .AS
2 28718/2 if F1=.S F3=.AF then F2 = .S .AP
3 14236/1 if DA0=100 then DA1 = 112
4 12867/1 if W1=.^@GET then DP = 80
5 35455/3 if then DA1 = 113 112 114
6 10857/1 if SA2=016 then SA3 = 172
7 10643/1 if W1=.^@EHLO then DP = 25
8 9914/1 if W5=.MAIL then W3 = .HELO
9 9914/1 if W3=.HELO then W7 = .RCPT
10 9898/1 if DP=25 F3=.AF W3=.HELO then W5 = .MAIL
11 28882/3 if F2=.AP then F3 = .AP .AF .R
12 35455/4 if then F1 = .S .AF .AS .R
13 34602/4 if F3=.AF then F2 = .S .AP . .AS
14 7645/1 if W5=. then W8 = .
15 7596/1 if W3=. then W7 = .
16 7596/1 if W2=. then W6 = .
17 7596/1 if W2=. then W5 = .
18 29549/4 if F1=.S then F2 = .S .AP . .A
19 7365/1 if DUR=0 W2=. then W4 = .
20 35455/5 if then F3 = .S .AP .AF .AS .R
21 6823/1 if F2=.AP W7=.Mozilla/ then W6 = .User-Age
22 12885/2 if DP=80 then W1 = .^@GET .
23 12867/2 if W1=.^@GET then W3 = .HTTP/1.0^M^ .align=
24 5827/1 if DP=20 then LEN = 0
25 10642/2 if F2=.AP W1=.^@EHLO then W3 = .HELO .MAIL
26 10105/2 if W7=.RCPT then W1 = .^@EHLO .^@HELO
27 10105/2 if W7=.RCPT then W5 = .MAIL .RCPT
28 4814/1 if SA3=172 SA0=050 F1=.S then SA2 = 016
29 35455/8 if then SA3 = 196 172 197 194 195 135 192 152
30 12838/3 if DP=25 then W1 = .^@EHLO . .^@HELO
31 7279/2 if SA0=050 then SA2 = 016 073
32 3521/1 if F3=.AF W5=.http://h then DA0 = 100
33 6852/2 if W4=.Referer: then W7 = .Keep-Ali .Mozilla/
34 6824/2 if W7=.Mozilla/ then W4 = .Connection: .Referer:
35 19139/6 if DUR=0 F1=.S then DP = 113 25 80 79 22 515
36 18807/6 if DA1=112 then DA0 = 050 100 194 207 149 020
37 29549/10 if F1=.S then DP = 113 25 23 80 135 21 79 22 515 139
38 35455/12 if then DA0 = 105 050 204 084 168 148 169 100 194 207 149 020
39 2802/1 if SA1=091 then SA0 = 233

40 35455/13 if then SA2 = 037 016 182 168 169 115 218 027 008 227 073 007 013
41 5223/2 if SA3=194 then SA0 = 021 153
42 24974/10 if DUR=0 then DP = 113 25 23 80 20 79 22 1022 515 1023
43 6813/3 if F3=.AF W7=.Mozilla/ then W5 = .Keep-Ali .http://m .http://h
44 35455/16 if then SA1 = 113 075 112 091 114 115 218 251 060 033 151 248 001 177 216 215
45 2138/1 if DA0=100 SA1=113 then SA2 = 016
46 7911/4 if SA2=016 F2=.AP F3=.AF then SA1 = 113 112 114 115
47 12885/7 if DP=80 then W4 = .HTTP/1.0^M^ .Connection: .Referer: . .Host: .User-Agent: .If-Modified-Sinc
48 35455/24 if then SA0 = 105 158 050 204 084 182 233 168 148 169 100 194 108 207 021 149 189 153 020 069 191 234 010 104
49 7634/6 if DA1=112 SA2=016 then DA0 = 050 100 194 207 149 020
50 12875/14 if DP=80 F1=.S then W6 = .User-Age .[en] .Connecti . .Accept: .(X11; .http://m .http://h .03 .06 .08 .16-Mar-9 .23 .11
51 10857/12 if SA2=016 then DA0 = 105 050 204 084 168 148 169 100 194 207 149 020
52 648/1 if F2=.AP W6=.PORT then W8 = .LIST^M^
53 1849/3 if W2=.^C then W6 = ." .' .^X^@DUMB
54 7656/32 if W7=. then DUR = 0 23 1 12 108 4 30 6 9 21 24 7 14 22 2 3 11 15 27 29 18 5 42 44 36 17 64 77 78 123 103 3545
55 12838/92 if DP=25 then DUR = 113 25 0 23 114 1 100 80 12 4 30 6 899 9 21 24 20 7 79 8 26 14 22 2 3 10 11 15 27 29 13 904 902 901 28 69 16 18 5 900 3599 41 33 94 31 44 38 60 96 86 36 90 52 39 19 3600 65 32 61 83 17 34 903 56 53 95 57 59 54 35 37 76 89 70 107 3601 77 106 110 109 87 3602 104 43 93 99 67 62 103 71 81 98
56 10643/84 if W1=.^@EHLO then DUR = 113 25 0 23 114 1 80 12 4 30 6 9 21 24 20 7 79 8 26 14 22 2 3 10 11 15 27 29 13 28 69 16 18 5 3599 41 33 94 31 44 38 60 96 86 36 90 52 39 19 3600 32 61 83 17 34 56 53 95 57 59 54 35 37 76 89 70 107 3601 77 106 110 109 87 3602 104 43 93 99 67 62 103 71 81 98

Below are the attacks detected by the rules above, sorted by the number of the rule that contributed the greatest proportion of the anomaly score. Attack names are those used in the DARPA evaluation. The "Pct." column shows the percentage contribution of the listed rule. The "anomaly" column shows all attributes involved in the rule and their values. A "?" indicates the anomalous value, corresponding to the consequent. For example, the first line shows that rule 1 contributed to 44.70% of the anomaly score for the *apache2* detection. Rule 1 is "if F2=.AP then F1 = .S .AS" The actual value of F1 in the attack is ".AP" (indicating the TCP connection was not opened properly). Table A2 shows 111 detections rather than 117 because the alarms were not postprocessed to remove duplicate alarms (within 60 seconds).

Table A2. Attacks detected by rules in table A1.

```
Attack         Rule Pct.    Anomaly
------         ---  ----    -------
apache2        001 (44.70) F1?=.AP F2=.AP
eject          001 (61.92) F1?=.AP F2=.AP
```

```
apache2       001 (62.37) F1?=.AP F2=.AP
ntinfoscan    001 (98.69) F1?=.AP F2=.AP
tcpreset      001 (99.98) F1?=.AP F2=.AP
neptune       002 (62.67) F1=.S F2?=.A F3=.AF
mscan         003 (99.97) DA1?=118 DA0=100
portsweep     003 (99.99) DA1?=118 DA0=100
dosnuke       005 (41.14) DA1?=115
ncftp         005 (66.75) DA1?=118
ncftp         005 (75.38) DA1?=118
guesstelnet   005 (79.9) DA1?=118
netbus        011 (99.77) F2=.AP F3?=.S
portsweep     012 (27.42) F1?=.F
portsweep     012 (53.83) F1?=.F
queso         012 (55.42) F1?=.F
dosnuke       018 (84.48) F1=.S F2?=.UAP
dosnuke       018 (88.64) F1=.S F2?=.UAP
dosnuke       018 (96.21) F1=.S F2?=.UAP
queso         020 (52.17) F3?=.F
ntinfoscan    020 (85.86) F3?=.AR
portsweep     020 (88.65) F3?=.F
portsweep     020 (99.32) F3?=.F
satan         022 (100) DP=80 W1?=.^@QUIT^M^
apache2       022 (99.99) DP=80 W1?=.^@^@^@^@^@^@^@^@
crashiis      023 (100) W1=.^@GET W3?=.
back          023 (100) W1=.^@GET W3?=.
crashiis      023 (37.22) W1=.^@GET W3?=.
back          023 (45.09) W1=.^@GET W3?=.
phf           023 (74.41) W1=.^@GET W3?=.
back          023 (74.42) W1=.^@GET W3?=.
phf           023 (83.08) W1=.^@GET W3?=.
phf           023 (85.74) W1=.^@GET W3?=.
crashiis      023 (95.98) W1=.^@GET W3?=.
crashiis      023 (99.69) W1=.^@GET W3?=.
crashiis      023 (99.73) W1=.^@GET W3?=.
casesen       024 (100) DP=20 LEN?=27649
sechole       024 (100) DP=20 LEN?=32771
ftpwrite      024 (100) DP=20 LEN?=6
xterm         024 (99.1) DP=20 LEN?=6075
warez         024 (99.65) DP=20 LEN?=283619
casesen       024 (99.72) DP=20 LEN?=27649
satan         024 (99.96) DP=20 LEN?=375
fdformat      024 (99.98) DP=20 LEN?=156430
portsweep     029 (26.15) SA3?=202
ntinfoscan    029 (26.16) SA3?=206
portsweep     029 (26.78) SA3?=209
sqlattack     029 (33.55) SA3?=206
netbus        029 (38) SA3?=209
perl          029 (39.17) SA3?=209
sshtrojan     029 (39.4) SA3?=202
guessftp      029 (39.73) SA3?=208
anypw         029 (40.84) SA3?=204
ffbconfig     029 (40.84) SA3?=206
netcat_setup  029 (40.84) SA3?=207
guest         029 (40.84) SA3?=209
guesstelnet   029 (40.84) SA3?=209
yaga          029 (40.84) SA3?=209
perl          029 (47.27) SA3?=207
ps            029 (47.27) SA3?=209
guest         029 (52.65) SA3?=153
ps            029 (54.55) SA3?=199
netbus        029 (60.38) SA3?=209
```

```
portsweep     029 (74.81) SA3?=153
netcat_breaki 029 (41.54) SA3?=206
mailbomb      030 (100) DP=25 W1?=.^@mail
mailbomb      030 (100) DP=25 W1?=.^@mail
mailbomb      030 (75.69) DP=25 W1?=.^@mail
sendmail      030 (98.16) DP=25 W1?=.^@MAIL
sendmail      030 (99.8) DP=25 W1?=.^@MAIL
crashiis      031 (52.89) SA2?=048 SA0=050
arppoison     035 (100) DP?=23 DUR=0 F1=.S
processtable  035 (100) DP?=23 DUR=0 F1=.S
neptune       035 (29.06) DP?=1 DUR=0 F1=.S
portsweep     035 (30.3) DP?=19 DUR=0 F1=.S
yaga          035 (31.25) DP?=21 DUR=0 F1=.S
tcpreset      035 (34.62) DP?=23 DUR=0 F1=.S
named         035 (36.98) DP?=53 DUR=0 F1=.S
named         035 (40.57) DP?=53 DUR=0 F1=.S
neptune       035 (54.5) DP?=21 DUR=0 F1=.S
portsweep     035 (59.62) DP?=143 DUR=0 F1=.S
netcat        037 (100) DP?=53 F1=.S
portsweep     037 (34.96) DP?=514 F1=.S
imap          037 (80.04) DP?=143 F1=.S
guesspop      037 (81.52) DP?=110 F1=.S
ls_domain     037 (84.1) DP?=53 F1=.S
guesstelnet   040 (42.48) SA2?=005
ls_domain     042 (38.5) DP?=53 DUR=0
named         042 (53.03) DP?=53 DUR=0
resetscan     042 (74.74) DP?=21 DUR=0
dict          044 (50.59) SA1?=118
imap          046 (42.56) SA2=016 SA1?=117 F2=.AP F3=.AF
guest         046 (55.36) SA2=016 SA1?=118 F2=.AP F3=.AF
guessftp      046 (96.16) SA2=016 SA1?=118 F2=.AP F3=.AF
ncftp         046 (97.71) SA2=016 SA1?=118 F2=.AP F3=.AF
crashiis      046 (97.95) SA2=016 SA1?=117 F2=.AP F3=.AF
ncftp         046 (99.78) SA2=016 SA1?=118 F2=.AP F3=.AF
sshprocesstab 046 (60.09) SA2=016 SA1?=118 F2=.AP F3=.AF
sechole       048 (79.82) SA0?=083
ppmacro       048 (95.75) SA0?=016
queso         050 (100) DP=80 F1=.S W6?=.11-Feb-9
ftpwrite      052 (100) F2=.AP W6=.PORT W8?=.STOR
teardrop      054 (100) DUR?=188 W7=.
processtable  054 (34.86) DUR?=1636 W7=.
back          054 (36.66) DUR?=28 W7=.
arppoison     055 (100) DP=25 DUR?=42
casesen       055 (100) DP=25 DUR?=6499
ntfsdos       055 (56) DP=25 DUR?=188
teardrop      055 (99.86) DP=25 DUR?=188
arppoison     056 (98.68) DUR?=55 W1=.^@EHLO
ntfsdos       056 (99.98) DUR?=97 W1=.^@EHLO
```