Learning Rules for Anomaly Detection of Hostile Network Traffic

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

Abstract

We introduce an algorithm called LERAD that learns rules for finding rare events in nominal time-series data with long range dependencies. We use LERAD to find anomalies in network packets and TCP sessions to detect novel intrusions. We evaluated LERAD on the 1999 DARPA/Lincoln Laboratory intrusion detection evaluation data set and on traffic collected in a university departmental server environment.

1. Introduction and Related Work

An important component of computer security is intrusion detection--knowing whether a system has been compromised or if an attack is occurring. Hostile activity can sometimes be inferred by examining inbound network traffic, operating system events, or changes to the file system, either for patterns signaling known attacks (signature detection), or for unusual events signaling possible novel attacks (anomaly detection). Anomaly detection has the advantage that it can sometimes detect previously unknown attacks, but has the disadvantage that it issues false alarms, because unusual events are not always hostile. Often both approaches are used. For example, a virus detector might scan files for strings signaling known viruses, and might also test for modifications of executable files as indications of possible new viruses.

Network anomaly detection is a particularly difficult problem because higher level (application) protocols are complex and difficult to model, and because data must be processed at high speed. A common approach is to use a firewall with rules programmed by a network administrator to block and/or log packets based on lower level features such as IP addresses and port numbers. This technique can detect or block port scans and unauthorized access to private services (e.g. *ssh*) from untrusted clients. However, detection of attacks on public services such as HTTP (web), SMTP (email), and DNS (host name lookup) currently rely on signature detection systems such as SNORT [10] or Bro [8] to scan for strings signaling known attacks. The rule set is quite large (SNORT has over 1800) and must be updated frequently. This would not be an effective defense against novel attacks or fast spreading worms. Network anomaly detection systems such as ADAM [2], SPADE [3], and eBayes [11], use machine learning approaches to model normal network traffic in order to identify unusual events as suspicious, but they model low-level (firewall-like) features such as addresses and port numbers, rather than application protocols.

We introduce an efficient, randomized algorithm called LERAD (Learning Rules for Anomaly Detection), which can discover relationships among attributes in order to model application protocols. LERAD differs from association mining approaches such as APRIORI [1] in that it finds enough rules with a small set of allowed values in the consequent to describe the data, rather than *all* rules (allowing only one value) above a support/confidence threshold. We believe this form is more appropriate for "bursty" (non-Poisson) time series data with long range dependencies, a characteristic of network traffic [4, 9].

2. Rule Learning Algorithm

LERAD learns conditional rules over nominal attributes in a time series (e.g. a sequence of inbound client packets or TCP sessions), in which the antecedent is a conjunction of equalities, and the consequent is a set of allowed values, e.g. *if port = 80 and word3 = HTTP/1.0 then word1 = GET or POST*. A value is allowed if it is observed in at least one training instance satisfying the antecedent. If in testing a disallowed value is observed, then an anomaly score of tn/r is generated, where t is the time since the last anomaly by this rule, n is the support (number of training instances satisfying the antecedent), and r is the number of allowed values (2 in this example). The idea is to identify rare events: those which have not

Proceedings of the Third IEEE International Conference on Data Mining (ICDM'03)
0-7695-1978-4/03 $ 17.00 © 2003 IEEE

occurred for a long time (large t) and where the average rate of "anomalies" in training is low (small r/n). If the total anomaly score summed over all violated rules exceeds a threshold, then an alarm is generated.

LERAD is a two pass algorithm. In the first pass, a candidate rule set is generated from a random sample S of the training data (attack-free network traffic). In the second pass, the rules are trained by collecting the set of allowed values for each antecedent. After training the rules are validated on a portion of the training data (e.g. the last 10%) to remove "poor" rules where the training data is not representative of the test data (Fig. 1). For example, the set of client IP addresses contacting a web server would be expected to grow steadily over time, so we would not wish to restrict the set to only those clients observed during a training period. On the other hand, a set of local server addresses or ports would not be expected to grow after a short training period, so this would be a "good" rule.

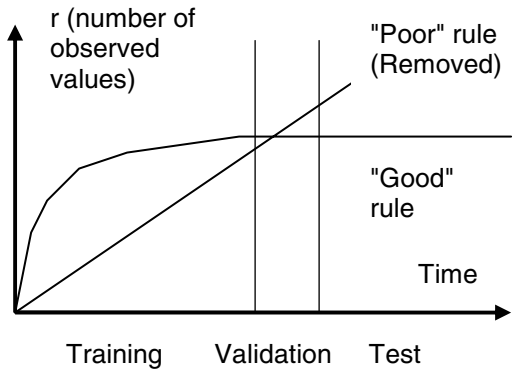

Figure 1. Growth of *r* for "good" and "poor" rules

The LERAD rule algorithm is as follows:

1. Rule generation. Randomly sample L pairs of training instances from a random subset S of the training data, and generate up to M rules per pair that satisfy both instances with $n/r = 2/1$, generating rule set R.
2. Coverage test. Discard rules from R to find a minimal (but not optimal) subset of rules that cover all instance-values in S, favoring rules with higher n/r over S.
3. Training, pass 2. Set the consequent of each rule in R to all values observed at least once in the training data when the antecedent is satisfied.
4. Validation. If a validation instance satisfies the antecedent but not the consequent of a rule (a violation), then remove the rule from R.
5. Test. For each instance, assign an anomaly score of $\Sigma\ tn/r$ summed over the violations.

LERAD requires two passes over the training data, one to sample S uniformly prior to generating rule antecedents, and a second pass to assign the consequents. We cannot simply use the beginning of the training data for S because attribute values are not Poisson distributed, so S would not representative of the rest of the training data.

In the rule generation step, we pick pairs of training samples and suggest rules based on the matching values. The algorithm is as follows:

Repeat L times
  Randomly pick two instances S_1 and S_2 from S
  Set $A = \{a: S_1[a] = S_2[a]\}$ (*matching attributes*)
  For $m = 1$ to M and A not empty do
    Randomly remove a from A
    If $m = 1$ then create rule r_i = "$a = S_1[a]$"
    Else add $S_1[a] = a$ to r_i's antecedent
  Add r_i to rule set R

For example, suppose that we randomly pick the first two instances of Table 1 as S_1 and S_2. Then the set of matching attributes is $A = \{word1, port, word3\}$. Suppose $M = 4$ and we randomly choose the attributes a in the order listed above. Then we generate the following rules:

- R_1: *word1 = GET*
- R_2: *if port = 80 then word1 = GET*
- R_3: *if port = 80 and word3 = HTTP/1.0 then word1 = GET*

Table 1. Example training sample S

Port	Word1	Word2	Word3
80	GET	/	HTTP/1.0
80	GET	/index.html	HTTP/1.0
25	HELO	pascal	

In the coverage test (step 2), we remove "redundant" rules, those which predict values in S already predicted by another rule with higher n/r over S (i.e. a rule which would probably generate higher anomaly scores in testing). The procedure is as follows:

Update the consequents in R over S
Sort R by decreasing n/r
For each rule R_i in R in decreasing order of r/n
  Mark the values predicted by R_i
  If no new values can be marked, remove R_i

For example, consider the rules above. After training over S and sorting by n/r these become:

- R_2: *if port = 80 then word1 = GET* ($n/r = 2/1$)
- R_3: *if port = 80 and word3 = HTTP/1.0 then word1 = GET* ($n/r = 2/1$)
- R_1: *word1 = GET or HELO* ($n/r = 3/2$)

(Note that the ordering of R_2 and R_3 is arbitrary, and R_1 has changed). R_2 marks the two *GET* values in S. R_3 would mark the same two values and no new values, so we remove it. R_1 marks the *HELO* in the third instance, in addition to the previously marked values, so we retain this rule.

3. Experimental Evaluation

Evaluation details are available from [7] and source code from [6]. To summarize, we tested LERAD using two attribute sets, one for IP packets and one for TCP connections. For packets, the attributes were the first 24 byte pairs (as 16-bit nominal values), beginning with the 10 pairs of bytes from the IP header. For TCP connections, the attributes were the source and destination port numbers, the individual bytes of the source and destination addresses, the connection length (in bytes), duration (in seconds), the TCP flags of the first and last two packets, and the first 8 words of the application payload (delimited by white space). We used a sample size of $|S| = 100$ (20 to 500 work well) and drew $L = 1000$ sample pairs, generating up to $M = 4$ candidate rules per pair, which are sufficient to generate 50 to 100 final rules for good results. Using larger L and M do not add significantly more rules.

We tested LERAD using two data sets: the 1999 DARPA/Lincoln Laboratory intrusion detection evaluation (IDEVAL) [5], and 623 hours of traffic collected from a university departmental server over 10 weeks in which we previously identified six attacks. We examine only inbound client (unsolicited) traffic, rate limited to 16 packets per connection per minute, and further truncated TCP connections after 256 bytes of the first payload packet. This filtering removes 98-99% of traffic, greatly speeding up LERAD with minimal effect on detection accuracy.

For IDEVAL, we trained LERAD on 7 days of attack-free traffic from inside sniffer week 3, and tested on 9 days of traffic from weeks 4 and 5, which contains evidence of 146 simulated probes, denial of service, and remote to local (R2L) attacks against four "victim" machines running SunOS, Solaris, Linux, and Windows NT. We evaluated LERAD according to the same criteria used in the 1999 blind evaluation, which requires only that we identify the target address and the attack time within 60 seconds at a threshold allowing 100 false alarms (10 per day). We exclude U2R (user to root) attacks, as allowed by the evaluation criteria, because these attacks exploit operating system weaknesses rather than network protocols, and would be difficult to detect in network traffic.

The second test was on switched Ethernet traffic collected on a Sun Ultra-AX i2 running Solaris 5.9 as a file, web, and mail server. We used SNORT and manual inspection to identify six attacks that eluded the university firewall: an inside port/security scan, three HTTP worms (*Code Red II, Nimda*, and *Scalper*), an HTTP proxy scan and a DNS version probe.

We used a stricter evaluation criteria: LERAD must identify at least one packet or TCP session involved in the attack. We counted multiple instances of a worm probe from different sources as a single attack, since a detection is likely to lead to a rule being added to an accompanying signature detection system. Lacking attack-free training data, we tested LERAD by dividing the traffic into 10 one-week periods and tested each week after training on the previous week. An attack in the training or validation data might mask a similar attack in the test data, but at least the first attack ought to be detected in the previous training/test pair. We allowed 250 false alarms, or 10 per 24 hours. Results averaged over 5 runs with different random number seeds are shown in Table 2.

Table 2. Number and percent of attacks detected at 10 false alarms per day in IDEVAL and university traffic

Data	Packets	TCP
IDEVAL	48.2 (33%)	95.2 (64%)
Univ.	1.4 (23%)	2.4 (40%)

LERAD using TCP attributes detects 64% of 146 attacks in IDEVAL, compared to 40% to 55% detected by the top four (of 18) systems in the original blind 1999 evaluation [5], even though most of those systems combined both signature and anomaly detection using both host and network based attributes. However, the comparison is biased in our favor because we had access to all of the test data during development. In the 1999 evaluation, participants were provided only with the first three weeks of data, containing a subset of the labeled attacks for development.

In IDEVAL we identified five categories of anomalies in the detected attacks.

- User behavior anomalies, e.g. unusual destination ports as part of a port scan, or client IP address anomalies in a password guessing attack.
- Anomalies due to exploitation of bugs in legal but seldom used (and therefore poorly tested) features of the protocol, for example, IP fragmentation in *teardrop* and *land*, which exploit bugs in IP reassembly code in a denial of service attack.
- Anomalies due to the failure to reproduce the idiosyncrasies of normal clients, for example, omitting the opening SMTP *HELO/EHLO* handshake (which is not required) in the *sendmail* buffer overflow root shell exploit.

- Anomalies deliberately introduced in an attempt to hide the attack signature at a higher protocol level, for example, scanning with FIN packets (with a missing ACK flag) to avoid having the probe logged by the server.
- Anomalies from the victim after a successful attack, for example, interrupted TCP connections from a crashed host.

In the university traffic, all of the anomalies are due to idiosyncratic variations, mostly at the application layer, for example, generic values in the HTTP *host* field for *Nimda, Scalper*, and the proxy scan, and an unusual backslash in the port/security scan: *GET / HTTP\1.0*. The one anomaly at the network layer was the unusual TCP segmentation in the Code Red HTTP command *GET default.ida?NNNNN...* in which *GET* appears in its own packet. The actual buffer overflow exploit code was truncated during filtering.

Our implementations process 10,000 packets or 3500 TCP sessions per second (after filtering) on a 750 MHz PC. The 8.9 GB of IDEVAL traffic was filtered in 7 minutes and processed by LERAD in under two minutes.

4. Concluding Remarks

LERAD differs from conventional network anomaly detection in that it models application protocols, allowing it to detect novel attacks on public servers. Application protocols are complex, but LERAD is able to learn important relationships between attributes given only rudimentary syntactic knowledge (e.g. tokens are separated by white space). It detects both simulated and real attacks, although there is a tradeoff between detection accuracy and a low false alarm rate. The false alarm problem is fundamental to anomaly detection because unusual events are not necessarily hostile.

Many of the anomalies detected by LERAD are not due to hostile code, but rather to legal but unusual protocol implementations. Unfortunately, this makes it difficult to understand the nature of the attack from the anomaly alone, or even to decide if an alarm should be dismissed as false. We could identify no consistent differences between true and false alarms.

One may argue that many attacks could be trivially modified to elude detection. Nevertheless, idiosyncratic anomalies are common in attacks in both of the data sets we used. We argue that writing an attack to elude detection is difficult because an attacker would not be able to test it in the target environment prior to launching it.

Future work includes a single-pass version of LERAD, research into better tokenization techniques in order to parse binary protocols such as DNS, and testing on additional data sets.

Acknowledgments

This work is partially funded by DARPA (F30602-00-1-0603).

References

[1] R. Agrawal & R. Srikant, "Fast Algorithms for Mining Association Rules", *Proc. 20th Intl. Conf. Very Large Data Bases*, 1994.

[2] D. Barbara, J. Couto, S. Jajodia, L. Popyack, & N. Wu, "ADAM: Detecting Intrusions by Data Mining", *Proc. IEEE Workshop on Information Assurance and Security*, 2001, pp. 11-16.

[3] J. Hoagland, SPADE, Silicon Defense, http://www.silicondefense.com/software/spice/, 2000.

[4] W. E. Leland, M. S. Taqqu, W. Willinger, & D. W. Wilson, "On the Self-Similar Nature of Ethernet Traffic", *Proc. ACM SIGComm*, 1993.

[5] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, & K. Das (2000), "The 1999 DARPA Off-Line Intrusion Detection Evaluation", *Computer Networks 34(4)*, 2000, pp. 579-595.

[6] M. Mahoney. Source code for PHAD, ALAD, LERAD, NETAD, SAD, EVAL3, EVAL4, EVAL and AFIL.PL is available at http://cs.fit.edu/~mmahoney/dist/

[7] M. Mahoney & P. K. Chan, " Learning Rules for Anomaly Detection of Hostile Network Traffic", Florida Tech. technical report CS-2003-16, 2003.

[8] V. Paxson, "Bro: A System for Detecting Network Intruders in Real-Time", *Proc. 7'th USENIX Security Symposium*, 1998.

[9] V. Paxson, S. Floyd, "The Failure of Poisson Modeling", IEEE/ACM Transactions on Networking (3) , 1995, pp. 226-244.

[10] M. Roesch, "Snort - Lightweight Intrusion Detection for Networks", *Proc. USENIX Lisa*, 1999.

[11] A. Valdes & K. Skinner, "Adaptive, Model-based Monitoring for Cyber Attack Detection", *Proc. RAID*, 2000, pp. 80-92.