Network Traffic Anomaly Detection Based on Packet Bytes

Matthew V. Mahoney
Florida Institute of Technology Technical Report CS-2002-13
mmahoney@cs.fit.edu

ABSTRACT

Hostile network traffic is often "different" from benign traffic in ways that can be distinguished without knowing the nature of the attack. We describe a two stage anomaly detection system for identifying suspicious traffic. First, we filter traffic to pass only the packets of most interest, e.g. the first few packets of incoming server requests. Second, we model the most common protocols (IP, TCP, telnet, FTP, SMTP, HTTP) at the packet byte level to flag events (byte values) that have not been observed for a long time. This simple system detects 132 of 185 attacks in the 1999 DARPA IDS evaluation data set [5] with 100 false alarms, after training on one week of attack-free traffic.

1. INTRODUCTION

Network intrusion detection systems are classified as signature based or anomaly based. A *signature* detector, such as SNORT [13] or Bro [10] examines traffic for known attacks using rules written by security experts. When a new type of attack is discovered, new rules must be written and distributed. An *anomaly* detection system such as SPADE [15], ADAM [14], or NIDES [1] models normal traffic, usually the distribution of IP addresses and ports. Hostile traffic often falls outside this distribution. Anomaly detection has the advantage that no rules need to be written, and that it can detect novel attacks. But it has the disadvantages that it cannot say anything about the nature of the attack (since it is novel), and because normal traffic may also deviate from the model, generating false alarms. An anomaly detector can only bring the suspicious traffic to the attention of a network security expert, who must then figure out what, if anything, needs to be done.

Mahoney and Chan [6, 7, 8] identify five types of anomalies in hostile traffic.

- **User Behavior.** Hostile traffic may have a novel source address because it comes from an unauthorized user of a restricted (password protected) service. Also, probes such as *ipsweep* and *portsweep* [4] may attempt to access nonexistent hosts and services, generating anomalies in the destination addresses and port numbers.
- **Bug Exploits.** Attacks often exploit errors in the target software, for example, a buffer overflow vulnerability. Such errors are most likely to be found in the least-used features of the program, because otherwise the error is likely to have been discovered during normal use and fixed in a later version. Thus, any remaining errors are invoked only with unusual inputs (e.g. a very long argument to a seldom used command), which are not likely to occur during normal use.
- **Response Anomalies.** Sometimes a target will generate anomalous outgoing traffic in response to a successful attack, for example, a victimized mail server passing root shell command responses back to an attacker. This is analogous to Forrest's host based detection method [2], where a server compromise can be detected because it makes unusual sequences of system calls.
- **Bugs in the attack.** Attackers typically must implement client protocols themselves, and will fail to duplicate the target environment either out of carelessness or because it is not necessary. For example, many text based protocols such as FTP, SMTP and HTTP allow either uppercase or lowercase commands. An attacker may use lowercase for convenience, even though normal clients might always use uppercase.
- **Evasion.** Attackers may deliberately manipulate network protocols to hide an attack from an improperly coded intrusion detection system (IDS) monitoring the application layer [3, 12]. Such methods include IP fragmentation, overlapping TCP segments that do not match, deliberate use of bad checksums, short TTL values, and so on. Such events must be rare in normal traffic, or else the IDS would have been written to handle them properly.

Given the variety of anomalies, it makes sense to examine as many attributes as possible, not just the packet ports and addresses, as many systems now do. Also, because network traffic has nonstationary or self-similar behavior [11], it makes sense to use a nonstationary model, in which the probability of an event depends on the time since it last occurred, instead of on its average rate. The idea is that if an attribute takes on a novel value, or at least one not seen recently, then the data is suspicious.

The rest of this paper is organized as follows. In Section 2, we describe related work in network anomaly detection. In Section 3, we describe our Network Traffic Anomaly Detector (NETAD), which flags suspicious packets based on unusual byte values in network packets. In Section 4, we evaluate NETAD on the 1999 DARPA evaluation data set [5]. In Section 5, we summarize.

2. RELATED WORK

Network intrusion detection systems such as SNORT [13] and Bro [10] use hand written rules to detect signatures of known attacks, such as a specific string in the application payload, or suspicious behavior, such as server requests to unused ports. Anomaly detection systems such as SPADE [15], ADAM [14], and NIDES [1] learn a statistical model of normal network traffic, and flag deviations from this model. Models are usually based on the distribution of source and destination addresses and ports per transaction (TCP connections, and sometimes UDP and ICMP packets). For example, SPADE offers four models of incoming TCP connections:

- P(destination-address, destination-port)
- P(source-address, destination-address, destination-port)
- P(source-address, source-port, destination-address, destination-port)
- Bayes network approximation of the above.

Lower probabilities result in higher anomaly scores, since these are presumably more likely to be hostile.

ADAM is a classifier which can be trained on both known attacks and on (presumably) attack-free traffic. Patterns which do not match any learned category are flagged as anomalous. ADAM also models address subnets (prefixes) in addition to ports and individual addresses. NIDES is a component of EMERALD [9], a system that integrates host and network based techniques using both signature and anomaly detection. The NIDES component, like SPADE and ADAM, models ports and addresses, flagging differences between short and long term behavior.

SPADE, ADAM, and NIDES use stationary models, in which the probability of an event is estimated by its average frequency during training. PHAD [6], ALAD [8], and LERAD [7] use nonstationary models, in which the probability of an event depends instead on the time since it last occurred. For each attribute, they collect a set of allowed values (anything observed at least once in training), and flag novel values as anomalous. Specifically, they assign a score of tn/r to a novel valued attribute, where t is the time since the attribute was last anomalous (during either training or testing), n is the number of training observations, and r is the size of the set of allowed values. Note that r/n is the average rate of anomalies in training; thus attributes with high n/r are not likely to generate anomalies in testing and ought to score high. The factor t makes the model nonstationary, yielding higher scores for attributes for which there have been no anomalies for a long time.

PHAD, ALAD, and LERAD differ in the attributes that they monitor. PHAD (Packet Header Anomaly Detector) has 34 attributes, corresponding to the Ethernet, IP, TCP, UDP, and ICMP packet header fields. It builds a single model of all network traffic, incoming or outgoing. ALAD (Application Layer Anomaly Detector) models incoming server TCP requests: source and destination addresses and ports, opening and closing TCP flags, and the list of commands (the first word on each line) in the application payload. Depending on the attribute, it builds separate models for each target host, port number (service), or host/port combination. LERAD (LEarning Rules for Anomaly Detection) also models TCP connections, but samples the training data to suggest large subsets to model. For example, if it samples two HTTP port requests to the same host, then it might suggest a rule that all requests to this host must be HTTP, and it builds a port model for this host.

PHAD, ALAD, and LERAD were tested on the 1999 DARPA off-line intrusion detection evaluation data set [5], by training on one week of attack free traffic (inside sniffer, week 3) and testing on two weeks of traffic (weeks 4 and 5) containing 185 detectable instances of 58 attacks. The DARPA data set uses variations of exploits taken from public sources, which are used to attack systems running SunOS, Solaris, Linux, Windows NT, and a Cisco router on an Ethernet network with a simulated Internet connection and background traffic. At a threshold allowing 100 false alarms, PHAD detects 54 attack instances [6], ALAD detects 60, or 70 when the results were merged with PHAD [8], and LERAD detects 114, or 62% [7].

Table 1 shows the results for the top 4 systems (including versions of EMERALD and ADAM) of the 18 (submitted by 8 organizations) that participated in the original 1999 evaluation. These systems used a variety of techniques, both host and network based, and both signature and anomaly detection. The number of detections is shown out of the total number of instances that the system was designed to detect. This counts only attacks visible in the examined data, and only those of the types specified by the developer: probe, denial of service (DOS), remote to local (R2L), or user to root (U2R). PHAD, ALAD, and LERAD were evaluated later under similar criteria by the authors, but the evaluations were not blind or independent. The developers were able to tune their systems on the test data.

Table 1. Top four percentage of attacks detected in the 1999 DARPA IDS evaluation at 100 false alarms, out of the total number of attacks they were designed to detect [5, Table 6].

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

3. THE NETAD MODEL

NETAD (Network Traffic Anomaly Detector), like PHAD, detects anomalies in network packets. However, it differs as follows:

1. The traffic is filtered, so only the start of incoming server requests are examined.
2. Starting with the IP header, we treat each of the first 48 bytes as an attribute for our models--we do not parse the packet into fields.
3. There are 9 separate models corresponding to the most common protocols (IP, TCP, HTTP, etc.).
4. The anomaly score tn/r is modified to (among other things) score rare, but not necessarily novel, events.

3.1. Traffic Filtering

The first stage of NETAD is to filter out uninteresting traffic. Most attacks are initiated against a target server or operating system, so it is usually sufficient to examine only the first few packets of incoming server requests. This not only filters out traffic likely to generate false alarms, but also speeds up processing. NETAD removes the following data.

- Non IP packets (ARP, hub test, etc.).
- All outgoing traffic (which means that response anomalies cannot be detected).
- All TCP connections starting with a SYN-ACK packet, indicating the connection was initiated by a local client. Normally, attacks are initiated remotely against a server.
- UDP to high numbered ports (>1023). Normally this is to a client (e.g. a DNS resolver).
- TCP starting after the first 100 bytes (as determined by the sequence number). A 4K hash table is used to store the starting TCP SYN sequence number for each source/destination address/port combination. There is a small amount of packet loss due to hash collisions.
- Packets to any address/port/protocol combination (TCP, UDP, or ICMP) if more than 16 have been received in the last 60 seconds. Again, a 4K hash table (without collision detection) is used to index a queue of the last 16 packet times. This limits packet floods.
- Packets are truncated to 250 bytes (although NETAD uses only the first 48 bytes).

3.2. NETAD Attributes

NETAD models 48 attributes, consisting of the first 48 bytes of the packet starting with the IP header. Each byte is treated as a nominal attribute with 256 possible values. For a TCP packet, the attributes are usually the 20 bytes of the IP header, 20 bytes of the TCP header, and the first 8 bytes of application payload. A TCP SYN (request to open) packet usually contains TCP options in the first 4 bytes where the application data would go. If the packet is less than 48 bytes long, then the extra attributes are set to 0.

3.3. NETAD Models

NETAD separately models 9 subsets of the filtered traffic corresponding to 9 common packet types, as follows.

- All IP packets (including TCP, UDP, and ICMP).
- All TCP packets.
- All TCP SYN packets (with no other flags set, normally the first packet, usually containing TCP options and no payload).
- All TCP ACK packets (normally the second and subsequent packets, which contain a payload).
- TCP ACK packets to ports 0-255.
- TCP ACK to port 21 (FTP).
- TCP ACK to port 23 (telnet).
- TCP ACK to port 25 (SMTP mail).
- TCP ACK to port 80 (HTTP).

A packet may belong to more than one subset. For instance, an HTTP data packet is also TCP ports 0-255, TCP ACK, TCP, and IP. For each model, an anomaly score is computed, and the sum is assigned to the packet.

The reason for modeling ports 0-255 is for ease of implementation. Each packet type can be distinguished from some other by examining only one attribute (byte). For instance, ports 0-255 can be distinguished from other TCP ACK packets by examining only the upper byte of the destination port number.

3.4. NETAD Anomaly Score

Recall that PHAD, ALAD, and LERAD use the anomaly score $\Sigma\ tn/r$ (summed over the attributes) where t is the time since the attribute was last anomalous (in training or testing), n is the number of training instances, and r is the number of allowed values (up to 256 for NETAD).

We make three improvements to the tn/r anomaly score. First, we reset n (the number of training examples) back to 0 when an anomaly occurs during training. Because the training data contains no attacks, we know that any such anomaly must be a false alarm. The effect is to reduce the weight of this attribute. We call this new score tn_a/r, where n_a is the number of training packets from the last anomaly to the end of the training period. Note that this is different from t, which continues to change during the test period. (Like ALAD and LERAD, NETAD uses the packet count rather than the real time to compute t).

The second improvement is to decrease the weight of rules when r (the number of allowed values) is near the maximum of 256. A large r suggests a nearly uniform distribution, so anomalies are of little value. Thus, we use the anomaly score $tn_a(1\text{-}r/256)/r$. For small r, this is approximately tn_a/r as before.

Third, a criticism of PHAD, ALAD, and LERAD is that they ignore the frequency of events. If a value occurs even once in training, its anomaly score is 0. To correct this, we add a second model, $t_i/(f_i + r/256)$, where t_i is the time (packet count in the modeled subset) since the value i (0-255) was last observed (in either training or testing), and f_i is the frequency in training, the number of times i was observed among training packets. Thus, the score is highest for values not seen for a long time (large t_i), and that occur rarely (small f_i). The term $r/256$ prevents division by 0 for novel values. It is preferred over a simple count offset (e.g. $t_i/(f_i + 1)$) because for novel events it reduces to a value that is large for small r.

Thus, the NETAD anomaly score for a packet is

$$\Sigma\ tn_a(1 - r/256)/r + t_i/(f_i + r/256) \quad (1)$$

where the summation is over the $9 \times 48 = 432$ subset/attribute combinations.

4. EXPERIMENTAL RESULTS

NETAD was tested and evaluated under conditions identical to PHAD, ALAD, and LERAD. It was trained on inside week 3 of the 1999 DARPA IDS evaluation data set, containing no attacks, and tested on weeks 4 and 5, containing 185 detectable attacks. Although there are 201 labeled attacks, the inside traffic is missing one day (week 4, day 2) containing 12 attacks, leaving 189. There is also one unlabeled attack (*apache2*) which we found by examining the test data, and there are five external attacks (one *queso* and four *snmpget*) against the router which are not visible from inside the local network. This leaves 185 detectable attacks.

An attack is counted as detected if the IDS identifies both the target address (or at least one target address if there are multiple targets), and the time within 60 seconds of any portion of the attack. This criteria is the same as the one used by the original DARPA evaluation. If there is more than one alarm identifying the same target within a 60 second period, then only the highest scoring alarm is evaluated and the others are discarded. This technique can be used to reduce the false alarm rate of any IDS, because multiple detections of the same attack are counted only once, but each false alarm would be counted separately.

Experiments with PHAD on the DARPA data set found a simulation artifact in the TTL field of the IP header that made attacks easy to detect, so (as with PHAD), this field was set to 0 for our evaluations [6].

4.1. Effects of Filtering the Traffic

Filtering the traffic as described in Section 3.1 reduces the DARPA training data (inside week 3) from 2.9 GB to 37 MB and the test data (inside weeks 4 and 5) from 6.0 GB to 72 MB. The total number of packets is reduced from 34.9 million to 1.1 million. On the reduced data, PHAD detects 52 attacks instead of 54, but runs about 100 times faster (7 seconds on a 750 MHz PC).

Table 2 shows the number of attack instances detected by NETAD for various models from Section 3.3 as the threshold is varied to allow 20, 50, 100, 500, or 5000 false alarms. For each model, the number of packets (training plus test) is shown, and the run time after filtering. Filtering is I/O bound, so this step dominates the total run time and is similar to the run time on unfiltered data, 835 seconds. Run times (in seconds) are for our implementation (a ~300 line C++ program) running on a 750 MHz Duron under Windows Me. (The filtering program is another ~250 lines of C++).

The first four rows show the number of attacks detected when Equation 1 is evaluated on the entire set of packets or just one of the nine subsets described in Section 3.3. As the set becomes

increasingly restricted, the number of false alarms drops, increasing the number of detections at a fixed false alarm rate. The last line shows the results for NETAD when the scores for all nine subsets are added. At 100 false alarms, NETAD detects 132 attacks.

Table 2. Packets, run time, and number of attacks detected (out of 185) as the threshold is varied to produce false alarm (FA) rates of 20 to 5000 for various models.

Model	**Packets**	**Sec.**	**20 FA**	**50 FA**	**100 FA**	**500 FA**	**5K FA**
unfiltered	34909810	835	4	14	23	42	71
IP	1101653	13	21	34	38	98	137
TCP	606923	9	38	51	64	96	121
TCPSYN	154057	7	74	91	97	109	113
9 models	1101653	20	66	97	132	148	152

4.2. Effects of the Scoring Function

Table 3 shows the effects of each of the changes to the anomaly score, progressing from tn/r to Equation 1. It shows that modeling either novel events (tn/r) and low frequency events ($t_i/(f_i + 1)$) are effective by themselves, and that each of the improvements described in Section 3.4 improves the score. Although the low frequency model is more effective by itself at 20-50 false alarms on the DARPA data, combining both models is more effective at 100-500 false alarms.

Table 3. Number of attacks detected at 20 to 5000 false alarms using various anomaly scoring functions.

Scoring Function	**20 FA**	**50 FA**	**100 FA**	**500 FA**	**5000 FA**
tn/r	56	78	104	141	157
tn_a/r	56	89	118	148	152
$tn_a(1 - r/256)/r$	60	92	120	149	152
$t_i/(f_i + 1)$	33	52	81	130	158
$t_i/(f_i + r/256)$	78	115	127	142	156
NETAD: $tn_a/r+t_i/(f_i+r/256)$	66	97	132	148	152

4.3. Effects of Attacks in Training

In a real setting, one would not have explicit training and test data available. Instead, one just has network traffic, which could contain unknown attacks at any time. One approach to this problem is to assume that the rate of attacks is low (relative to the volume of normal traffic) and to leave NETAD in training mode at all times. (Equation 1 is applicable in either mode). Unfortunately, if we fail to identify anomalous data as hostile, then those anomalies will be added to the model and future instances of the same or similar attacks might be missed.

Table 4 shows the effects of leaving NETAD in training mode during the attack period (weeks 4-5). The results are not as bad as we might expect. If the 58 types of attacks in the DARPA data set had identical signatures, then we should not expect to detect more than one instance of each. However, it seems there are enough differences between instances that we can detect 111 instances (at 100 false alarms) when NETAD is left in training mode for all three weeks, and 70 instances for the more realistic case where attacks might begin immediately after the start of training.

Table 4. Number of attacks detected at 20 to 5000 false alarms when NETAD is left in training mode.

Week 3	**Weeks 4-5**	**20 FA**	**50 FA**	**100 FA**	**500 FA**	**5000 FA**
Train	Test	66	97	132	148	152
Train	Train	47	80	111	120	140
Not used	Train	27	53	70	116	152

4.4. Analysis of Detected Attacks

Table 5 lists the number of detections (at 100 false alarms) for each category of attack. Each value is shown as a fraction of all detectable attacks. NETAD, like most network intrusion detection systems, performs poorly on U2R attacks, where an attacker with shell access gains access to another user (usually *root* or *admin*). Detecting such attacks requires the IDS to interpret user commands, which might be entered locally or hidden by using a secure shell. NETAD detects most of these attacks by anomalous source addresses when the attacker logs in, or uploads the exploit code to an FTP server normally used only for downloads. A *data* attack is where an authorized user copies or transmits secret data in violation of a security policy.

The category *poorly detected* includes the 74 (68 detectable) instances of attack types for which none of the original 18 evaluated systems in 1999 were able to detect more than half of the instances [5]. NETAD detects these at the same rate as other attacks, indicating that there is not a lot of overlap between the attacks detected by NETAD and by other techniques (signature, host based, etc.). This suggests that integrating NETAD with existing systems might improve the overall detection rate.

Table 5. Attacks detected by category.

Attack Category	**Detected at 100 False Alarms**
Probe	32/36 (89%)
Denial of Service (DOS)	43/63 (68%)
Remote to Local (R2L)	38/49 (78%)
User to Root (U2R)	18/33 (55%)
Data	1/4 (25%)
Total	132/185 (71%)
Poorly Detected in 1999	48/68 (71%)

Table 6 shows for each type of anomaly, the number of attack types for which at least one instance is detected by contributing at least 10% of the anomaly score for at least one packet. For example, source IP detects 35 types. For each type, the number detected (by any means) is shown, out of the total number of detectable attacks. Some attacks are detected by more than one attribute, in which case the numbers are shown only once. For example, NETAD detects 14 of 15 instances of portsweep and we record the detections under unusal packet sizes, though some of the detections were also detected by unusual TCP flags.

Source address is by far the most frequent contributor for detection. This is followed by the TCP window size field for a number of unrelated attacks. Because this looked to us suspiciously like a simulation artifact (like TTL), we reran NETAD with this field zeroed out However, this only decreased the number of detections at 100 false alarms from 132 to 129.

Table 6. Number of attack types at least partially detected by each attribute (at 100 false alarms).

Attribute	Attack types detected (total number of types -- category: detected/detectable)
Source IP address	35 -- Probe: 4/7 ipsweep, 2/2 ls, 2/2 satan; DOS: 2/4 apache2, 2/5 arppoison, 3/7 crashiis, 1/3 mailbomb, 1/3 processtable, 5/5 smurf, 4/4 syslogd, 1/3 tcpreset, 3/3 warezclient, 1/1 warezmaster; R2L: 6/7 dict, 2/3 guest, 2/2 imap, 5/5 ncftp, 3/3 netbus, 3/4 netcat, 1/3 ppmacro, 1/3 sshtrojan, 3/3 xlock, 3/3 xsnoop; U2R: 1/1 anypw, 2/3 casesen, 2/2 eject, 1/2 fdformat, 1/2 ffbconfig, 2/4 perl, 2/3 ps, 1/2 sechole, 1/2 sqlattack, 3/3 xterm, 2/4 yaga: Data: 1/4 secret
TCP window	9 -- Probe: ls, 3/3 ntinfoscan, 1/1 resetscan; DOS: apache2, 3/4 neptune; R2L: netbus, netcat, 2/3 phf; U2R: casesen
Packet size	8 -- Probe: 14/15 portsweep, 3/3 queso; DOS: back, 1/1 land, neptune, 4/4 pod, smurf; R2L: named
Payload	7 -- DOS: 4/4 back, neptune, 1/2 udpstorm; R2L: 3/3 named, 2/2 sendmail; U2R: sechole, yaga
Destination	4 -- R2L: dict, ncftp; U2R: perl, xterm
Fragmented	2 -- DOS: pod, 3/3 teardrop
TCP flags	2 -- Probe: portsweep, queso
Urgent data	1 -- DOS: 4/4 dosnuke
TOS	1 -- R2L: 2/2 ftpwrite
Coincidental	1 -- Probe: 2/2 illegalsniffer
Not detected	6 -- DOS: 0/3 selfping; R2L: 0/1 framespoofer, 0/2 httptunnel, 0/0 snmpget; U2R: 0/2 loadmodule, 0/2 ntfsdos

Six of 58 attack types are not detected. *Framespoofer* delivers an exploit by email (as HTML). *Httptunnel* is a backdoor which disguises its communication with the attacker as web client requests. NETAD misses both of these because it does not monitor outgoing traffic or incoming client responses. *Selfping* and *ntfsdos* generate no traffic directly, but could theoretically be detected because they reboot the target, interrupting TCP connections. *Snmpget* is an external router attack, not visible on the inside sniffer. *Loadmodule* is U2R, thus hard to detect.

5. CONCLUDING REMARKS

NETAD, like any network anomaly detector, does not describe the nature of an attack, or even indicate if an event is hostile or not. Instead, it just finds unusual or interesting events in a vast amount of data, and brings them to the attention of a network security expert for further analysis. It is meant to supplement, rather than replace, existing security tools and methods, such as code reviews, encryption, firewalls, virus detectors, and so on. Like all security tools, it requires some investment of time and effort to use it. The challenge is to use good filtering to minimize this effort.

NETAD uses a prefiltering stage to select rate-limited incoming server requests, and then models nine subsets of this data representing the most common protocols. It tests packets independently, flagging packets in which the first 48 bytes are novel or rare within a model. This crude method performs well on the 1999 DARPA intrusion detection evaluation data set, either with or without attack-free training data. Future work will include adding more models (possibly using machine learning to select traffic types), and expanding the attribute set to monitor reassembled TCP connections, and possibly host based data such as system calls or audit logs.

Acknowledgments

This research is partially supported by DARPA (F30602-00-1-0603). I wish to thank Philip K. Chan for collaboration on this research and helpful comments on this paper.

References

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

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

[3] Handley, M., C. Kreibich and V. Paxson, "Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics", Proc. USENIX Security Symposium, 2001.

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

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

[6] Mahoney, M., P. K. Chan, "PHAD: Packet Header Anomaly Detection for Identifying Hostile Network Traffic", Florida Tech. technical report 2001-04, http://cs.fit.edu/~tr/

[7] Mahoney, M., P. K. Chan, "Learning Models of Network Traffic for Detecting Novel Attacks", Florida Tech. technical report 2002-08, http://cs.fit.edu/~tr/

[8] Mahoney, M., P. K. Chan, "Learning Nonstationary Models of Normal Network Traffic for Detecting Novel Attacks ", Edmonton, Alberta: Proc. SIGKDD, 2002, 376-385.

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

[10] Paxson, Vern, "Bro: A System for Detecting Network Intruders in Real-Time", Lawrence Berkeley National Laboratory Proceedings, 7'th USENIX Security Symposium, Jan. 26-29, 1998, San Antonio TX,

[11] Paxson, Vern, and Sally Floyd, "The Failure of Poisson Modeling", IEEE/ACM Transactions on Networking (3) 226-244, 1995.

[12] Ptacek, Thomas H., and Timothy N. Newsham, "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection", January, 1998, http://www.robertgraham.com/mirror/Ptacek-Newsham-Evasion-98.html

[13] Roesch, Martin, "Snort - Lightweight Intrusion Detection for Networks", Proc. USENIX Lisa '99, Seattle: Nov. 7-12, 1999.

[14] Sekar, R., M. Bendre, D. Dhurjati, P. Bollineni, "A Fast Automaton-based Method for Detecting Anomalous Program Behaviors". Proceedings of the 2001 IEEE Symposium on Security and Privacy.

[15] SPADE, Silicon Defense, http://www.silicondefense.com/software/spice/