

NIC-based intrusion detection: A feasibility study*

M. Otey R. Noronha G. Li S. Parthasarathy[†] D. K. Panda
Department of Computer and Information Science, Ohio State University
Email: {otey,noronha,ligan,srini,panda}@cis.ohio-state.edu

November 27, 2002

Abstract

We present and evaluate a NIC-based network intrusion detection system. Functions such as signature-based and anomaly-based packet classification are performed on the NIC, which has its own processor and memory. This makes the system virtually impossible to bypass or tamper with as can be the case with software-based systems that rely on the host operating system to function. We empirically evaluate such systems from the perspective of quality and performance (bandwidth of acceptable messages) under varying conditions of host load. The preliminary results we obtain are very encouraging and lead us to believe that such NIC-based security schemes could very well be a crucial part of next generation network security systems.

1 Introduction

In today's information age, where nearly every organization is dependent on the Internet to survive, it is imperative to guarantee the privacy and security of the information being exchanged. This issue has been brought further into the foreground by the recent thrust toward cyber-space security and the almost omnipresent deployment of network intrusion detection systems. The goal of an intrusion detection system is to detect inappropriate, incorrect, and unusual activity on a network or on the hosts belonging to a local network by monitoring network activity.

How do we know if an attack has occurred or if one has been attempted? This typically requires sifting through huge volumes of data gathered from the network, host, or file system 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. Most work on signature and anomaly detection has relied on detecting intrusions at the host processor level, even those that detect intrusions occurring only at the network layer and below. A problem with these approaches is that even if anomalous/intrusion activity is detected, one is often unable to prevent the anomalous packets from causing havoc in the form of disrupting the system and over utilizing the system CPU (e.g. via denial-of-service attacks). *In this paper we study the feasibility of using Network Interface Cards (NICs) to detect and potentially prevent such intrusions.*

The primary role of NICs in computer systems is to move data between the system's components and the network. A natural extension to this role would be to actually police the packets forwarded in each direction by examining packet headers and simply not forwarding suspicious packets. The rationale for NIC-based intrusion detection can be stated as follows: First, NIC-based strategies, if successful, can prevent malicious packets from reaching the host operating system. This trapping before the host operating system can improve the performance of the overall intrusion detection system. Second, if the host is loaded (with other programs running simultaneously), an intrusion detection system that relies on host processing capability may be slowed down, thereby adversely affecting the bandwidth available for acceptable network transmissions. A NIC-based strategy will not be affected by the load on the host and therefore will not suffer the same slowdown. Third, there is a potential to naturally handle the scalability problem associated with centralized intrusion detection systems, since each individual NIC can handle the in-bound and out-bound traffic of the particular processor/local area network it is concerned with, thus effectively distributing the work concerned.

*This work was supported by an Ameritech Faculty Fellowship and NSF grants CCR-0204429 and EIA-9986052

[†]Contact Author

The above advantages notwithstanding, the current disadvantage to NIC-based intrusion detection is that processing capability on the NIC is much slower and the memory sub-system is much smaller when compared with host-based methods. Given this fact, the question now becomes how best to use the NIC's processing capabilities for intrusion detection? This is the key question that this paper seeks to answer.

We describe three relatively simple intrusion detection strategies (designed with the memory and processor limitation of current-day NICs in mind) and evaluate the performance of these schemes. We compare the NIC-based schemes with a purely host-based schemes while varying the load on the host processor. We evaluate each scheme based on the quality (how effective are such schemes for detecting intrusions) of detection, and the performance (in terms of bandwidth supported). Our preliminary results show that this direction of research is promising, and given the technology trends such systems are likely to form a crucial part of network security systems of the future.

The rest of this paper is organized as follows. In section 2 we document some of the related work in intrusion detection, NIC-based computing and NIC-based security. Section 3 documents our three intrusion detection strategies. We report on the experimental evaluation of above strategies in Section 4. Finally we conclude with some directions for future research in Section 5.

2 Related Work

In this section we present the related work in intrusion detection and NIC-based computing most relevant to our work.

2.1 Intrusion Detection Systems

There are two general approaches to the problem of intrusion detection: 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 of not being able to detect new attacks. Though anomaly detection can detect novel attacks, it has the drawback of not being able to discern intent. It can only signal that some event is unusual, but not necessarily hostile, thus generating false alarms.

Signature detection methods are better understood and widely applied. They are used in both host based systems, such as virus detectors, and in network based systems such as SNORT [22] and BRO [20]. 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. A limitation of these systems is that as new vulnerabilities or attacks are discovered, the rule set must be manually updated. Another disadvantage is that minor variations in attack methods can often defeat such systems.

Anomaly detection is a harder problem than signature detection because while signatures of attacks can be very precise, 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 [7]. Since what is considered normal could vary across different environments, a distinct model of normalcy can be learned individually. 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. Several approaches to detecting novel attacks have been proposed in the literature including those based on n-grams [7, 6], non-stationary probabilistic models [11], association rules [2, 9] etc. While most research in intrusion detection has focused on either signature detection or anomaly detection, most researchers have realized that the two models must work hand-in-hand to be most effective [2, 1].

2.2 NIC-based Computing and Security

Recently there has been a fair amount of activity in the area of NIC-based computing. The primary focus of such efforts has targeted the use of NIC resources for improving communication and synchronization performance. [17] discusses a NIC based implementation of atomic remote memory operations along with an implementation of locks based on these remote memory operations. An implementation of NIC-based QoS on a Myrinet cluster is discussed in [23]. A NIC-based barrier implementation that is an order of magnitude faster is described in [16]. In other system level work, the use of an active NIC was shown to significantly improve the performance of a Parallel Discrete Event Simulator [14, 13]. The SPINE project focuses on mechanisms to improve application performance by downloading

pieces of code to the NIC which are then executed in a sandbox [10]. There have also been several other investigations into the issues of improving application performance by using active NIC's [3, 5, 21, 8].

More closely related to this work is the use of NICs for firewall security [12]. The idea is to embed firewall-like security at the NIC level. Firewall functionality, such as packet filtering, packet auditing, and support for multi-tiered security levels, has been proposed. While most of these ideas are in their infancy, a couple of simple ideas have been commercialized (e.g. 3Com's embedded firewall).

3 Algorithms

In this section we describe the three algorithms (Port Scan Detector, Anomalous Client Detector, Naive Bayes Packet Classifier) we evaluated. The first and third are examples of signature detection algorithms while the second one is an example of an anomaly detection algorithm. Each of these algorithms were implemented both on the NIC as well as on the host processor. However, they have all been designed keeping the the memory and processing limitations of the NICs in mind.

3.1 Port Scan Detector

```
function DetectPortScan(DstPort)
begin
  Vector[]: An array of N bit vectors, each of length B bits
  f: function to map port numbers to bit numbers
  F: Number packets received between successive checks
  S: The number of bits that must be 1 for a port scan to be reported
  P: The number of packets per vector
  Current: A global counter that keeps track of the current bit vector
  Count: The number of packets represented by current bit vector
  Result: The result of the detector
  Set bit f(DstPort) of Vector[Current mod N] to 1;
  Count++;
  if (Count mod F = 0)
    Temp := OR all N bit vectors together;
    if (Sum of Ones in Temp > S)
      Result := "Port Scan Detected";
    else
      Result := "No Port Scan Detected";
  else
    Result := "No Check Performed";
  if (Count = P)
    Current++;
    Count := 0;
    Clear Vector[Current mod N];
  return Result;
end
```

Figure 1: The Port Scan Algorithm

The port scan detector algorithm (see Figure 1), uses an array of N bit vectors of length B to keep track of the ports of a given destination machine that have been accessed. Since the memory requirements of keeping track of each of the 65536 possible ports would be too expensive, and intractable, given the NIC's resource capabilities, we apply a many-to-one hash function (f) on each port to reduce the storage requirements. This many-to-one function reduces the set of possible ports to B bits. We maintain N of these B -bit vectors to implement a sliding window, so that one can detect potentially hard-to-detect time-delayed port scans. These N vectors are cycled through depending on the frequency with which packets arrive at the destination machine: After P packets have arrived, a new vector is used.

Once a packet has been received, the algorithm sets the proper bit in the current bit vector and updates its count. If less than F packets have been received since the last check for a port scan, the algorithm finishes with the message

that no check was made. If F packets have been received, the algorithm checks for a port scan by proceeding to OR the N vectors. If the number of 1's found in the OR is greater than a given threshold S, it declares the detection of a port scan.

3.2 Anomalous Client Detector

```

function DetectAnomalousClient(SrcIP, DstIP)
begin
  SDTable: A two dimensional hash table with one axis indexed by DstIP and the other by
  SrcIP, which holds the number of packets each DstIP receives from each SrcIP.
  DstTable: A one dimensional table holding the number of packets received by each DstIP
  H: Hash function for the SrcIP axis in the SDTable
  DstTable[DstIP]++;
  SDTable[DstIP][H(SrcIP)]++; /* conflict resolution is handled using standard approaches*/
  if ((SDTable[DstIP][H(SrcIP)] / DstTable[DstIP]) > threshold(DstIP))
    return "Normal Client";
  else
    return "Possible Anomalous Client";
end

```

Figure 2: Anomalous Client Detector

The anomalous client detector algorithm (see Figure 2) is loosely based on one of the models used in the non-stationary application layer anomaly detection (ALAD) algorithm proposed by Chan and Mahoney [11]. The objective of this model is to determine the anomaly score of a given packet based on previous interactions between a particular client and the particular destination host in question. The anomaly score is based on the value of $P(\text{srcIP} \mid \text{destIP})$, the probability that a client machine connects to a given host. The set of normal clients for a host are those for which $P(\text{srcIP} \mid \text{destIP}) > \text{threshold}(\text{DstIP})$. A value of $P(\text{srcIP} \mid \text{destIP})$ that is lower than the threshold may be indicative of suspicious behavior.

The model is capable of detecting two types of anomalous behavior. One behavior is when a new or existing client (someone the system has not seen before) attempts to connect to a host that it has not connected to before. The second behavior is when the amount of a client's interaction with a particular host radically changes over time. Note that the system can only detect anomalies in the quantity of the interactions between the host and the client; it cannot detect anomalies in the character of the interactions.

To model the first scenario we need to model an *anomaly score* or *surprise factor*, for such a scenario unfolding. Basically, a new client accessing a well known world wide web portal is less surprising than a new client accessing a particular internal machine that is typically accessed only by a handful of trusted client machines. To model this surprise factor, we rely on incrementally keeping track of a threshold function that is inversely proportional to the entropy of accesses to a particular destination host. The entropy for the web based example (low threshold value) would be much higher than the entropy for the trusted client example (high threshold value). This threshold function (by the very definition of how entropy is computed) is dependent on the number of different client connections for a given host as well as the frequency of client connections. We incrementally maintain the threshold by adopting ideas from our previous research [18]. Note that a destination host which receives connections from a large number of sources (clients), is more likely to be accessed by a new source (client), than by a host which typically receives connections from just a few sources (clients). To model the second scenario, we monitor the rate of change of $P(\text{srcIP} \mid \text{destIP})$ over temporal windows. This would allow us to detect large fluctuations in the conditional probabilities which would trigger a possible anomaly alarm.

The basic algorithm stores information in a set of hash tables. Again we use hash tables so that one can save on memory utilization, at a (hopefully small) cost to accuracy of the model. Upon receiving a new packet, it simply computes the conditional probability $P(\text{srcIP} \mid \text{destIP})$ and if this conditional probability is less than the threshold determined from the training data, then the algorithm considers the source host to be an anomalous client. In the current implementation of the algorithm the thresholds are not adjusted as a function of time (i.e. the algorithm is not non-stationary), so the second scenario is not modeled, but we plan to address this problem in the near future.

3.3 Naive Bayes Packet Classifier

```
function Classify(Packet)
begin
  FeatureVector: A vector holding the features upon which the classifier computes probabilities
  Op: Operation to pass to FindAverageTTLDeviation function
  FeatureVector.Protocol := Packet.Protocol;
  FeatureVector.Flags := Packet.Flags;
  FeatureVector.TTL := Packet.TTL;
  FeatureVector.SrcPort := Packet.SrcPort;
  FeatureVector.DstPort := Packet.DstPort;
  FeatureVector.PacketSize := Packet.Size;
  if (Packet.Flags contains SYN)
    Op := AddFlow;
  else if (Packet.Flags contains FIN)
    Op := RemoveFlow;
  else
    Op := UpdateFlow;
  FeatureVector.TTLDev := FindAverageTTLDeviation(Packet.SrcIP, Packet.SrcPort, Packet.DstPort, Packet.Protocol, Packet.TTL, Op);
  if (Prob(FeatureVector | GoodClass) > Prob(FeatureVector | BadClass))
    return "Good Packet"
  else
    return "Bad Packet"
end
```

Figure 3: Simplified Naive Bayes Packet Classifier

The Naive Bayes packet classifier (see Figure 3) attempts to distinguish between good and bad packets by using a set of directly extractable features (Protocol Type, TCP and IP Header Flags, Source Port, Destination Port, Packet Size, TTL) and one derived feature (average TTL deviation). The classifier is first trained on training data. On receiving a new packet, the algorithm first extracts the direct features and then computes the derived feature. Then, using the Naive Bayes assumption (conditional independence) and the information derived from the training data, the probabilities that the given packet belongs to the good class and the bad class is computed. If the probability that a packet is good exceeds the probability it is bad, then the packet is allowed to pass through to the destination. Otherwise, it is filtered (dropped) at the NIC. The derived feature average TTL deviation is computed as shown in Figure 4.

4 Experimental Results

In this section we detail the experimental results we obtained. We first start by describing the experimental setup (the machine configurations, data sets used and evaluations performed) for each of the three algorithms. We then examine the qualitative and quantitative aspects of the algorithms. The quantitative study focuses on the comparing the host-based and NIC-based schemes while the quality study focuses on the quality of the algorithms as measured by the accuracy of detection.

4.1 Setup

All host-based experiments, unless otherwise noted, were evaluated on dual-processor 300 MHz Pentium II machines with 128 MB of memory. All NIC-based evaluations were done on Myrinet NICs, each of which had a 66Mhz LANai 4 processor and 1 MB of memory.

To test each algorithm, we used synthetic data sets. The synthetic data sets were generated using a program available at <http://www.cis.ohio-state.edu/~otey/NIC/>. The size of the packets, unless otherwise noted, is 2048 bytes. Each packet is composed of four protocol bits, four flag bits, four source IP bytes, four destination IP bytes, two source port bytes, two destination port bytes, one TTL byte, two packet size bytes and 2032 bytes for the data. We set it up so

```

function FindAverageTTLDeviation(SrcIP, SrcPort, DstPort, Protocol, TTL, Op)
begin
  Table: A table indexed by SrcIP, SrcPort, DstPort, and Protocol that holds the number of
  packets in a flow (Count), the average TTL (AvgTTL) of each packet in the flow, and an
  estimate of the average deviation (AvgDev) of the TTLs of the packets in the flow
  Key: The row of the table corresponding to a given SrcIP, SrcPort, DstPort, and Protocol
  switch(Op)
  case AddFlow:
    Table[Key].Count := 1;
    Table[Key].AvgTTL := TTL;
    Table[Key].AvgDev := 0;
  case UpdateFlow:
    TTLSum := (Table[Key].AvgTTL * Table[Key].Count) + TTL;
    Table[Key].Count++;
    AverageTTL := TTLSum / Table[Key].Count;
    Dev := abs(TTL - AverageTTL);
    Table[Key].AvgTTL := AverageTTL;
    Table[Key].AvgDev := [(Table[Key].AvgDev * (Table[Key].Count - 1)) + Dev] / Table[Key].Count
  case RemoveFlow:
    Remove row Table[Key];
  end switch
end

```

Figure 4: Deriving the average TTL feature for the NB classifier

that the network packet generator sends roughly 1 packet every 0.001 seconds for a net theoretical throughput of 2MB per second. However, the actual measured throughput is 1796 KB per second.

For evaluating port scans the data set generator generates K independent port scans (that scan different parts of a particular destination host's ports). Note that here we are actually evaluating the case where a single NIC can be responsible for a system area network or a small subnet. The parameters to the data set generator include the number of port scans ($K = 5$), the length of each port scan (32768 ports), the average distribution of normal packets to port scan related packets (5:1) when a port-scan is in progress, and the total number of network packets (1 million). As for the packets that are not part of the port scan, their destination port numbers are distributed such that 99% of all port accesses are to port numbers below 1024 (reflecting realistic distributions). The algorithmic parameters we used for the port scan were $F = 16384$ and 32768 , $B = 8192$, $P = 32768$, $N = 4$, and $S = 3072$.

The data set generated for evaluating the Naive Bayes classifier has two labels for each packet (good or bad). The data set identifies a packet as bad if one of the following hold: a) the packet uses an invalid protocol; b) the packet has an invalid combination of flags set; c) uses a protocol that makes no use of flags and yet one or more flags are non-zero; d) the source port of a connection is less than 1024; e) the packet size is too large for a given protocol; f) the TTL value is invalid; g) for a given flow, the average deviation in TTL values is high. In the last case, a flow is defined as a series of packets with the same source IP address, the same source and destination ports, the same protocol and the last packet in the flow has the FIN flag set. The idea is to simulate real network traffic as closely as possible. We use two hundred thousand packets (half good, half bad) to train the classifier and we tested the classifier on the remaining 1 million packets. For the Naive Bayes classifier we had to discretize the continuous attributes. Details of how we handle this are described elsewhere [15].

The data set for the anomalous client detector focuses solely on the source and destination IP addresses. All other fields are filled with random numbers. The data set we used consisted of 256 local hosts, the set of valid clients per host ranged from 0-64, and the total set of clients was the remaining valid IP addresses on the Internet (approximately 4.3 billion hosts). The first hundred thousand packets were messages from valid clients (the training data) and the next 1 million packets are testing packets, about 90% of which come from the set of valid clients.

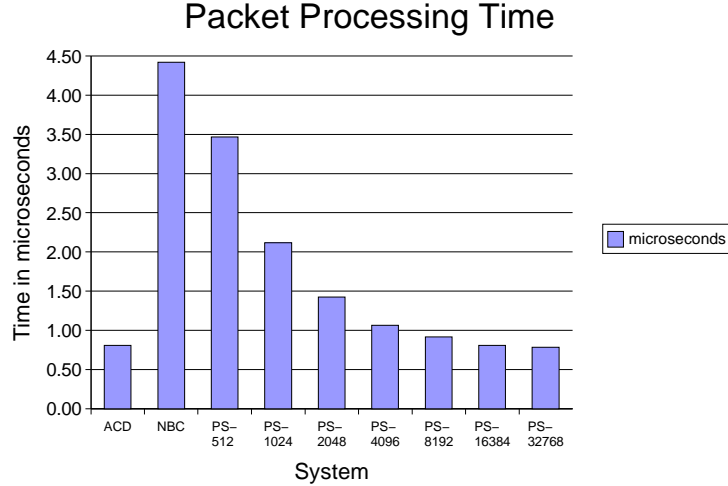


Figure 5: Evaluation of Proposed Method: Similarity plots

4.2 Quantitative Experiments

Before detailing the experimental results on performance, we wanted to quickly get a sense for the individual resource requirements (memory and computational) of the different algorithms.

Figure 5 documents these results. The Y-axis represents wall-clock execution time in seconds on the host processor. As we can see from the figure, the Naive Bayes classifier takes the longest time to execute followed by the port-scan algorithm at various parameter settings. The anomaly detection algorithm takes the least time to execute although it is comparable to some of the port-scan algorithm at larger parametric ($F \geq 16384$) settings.

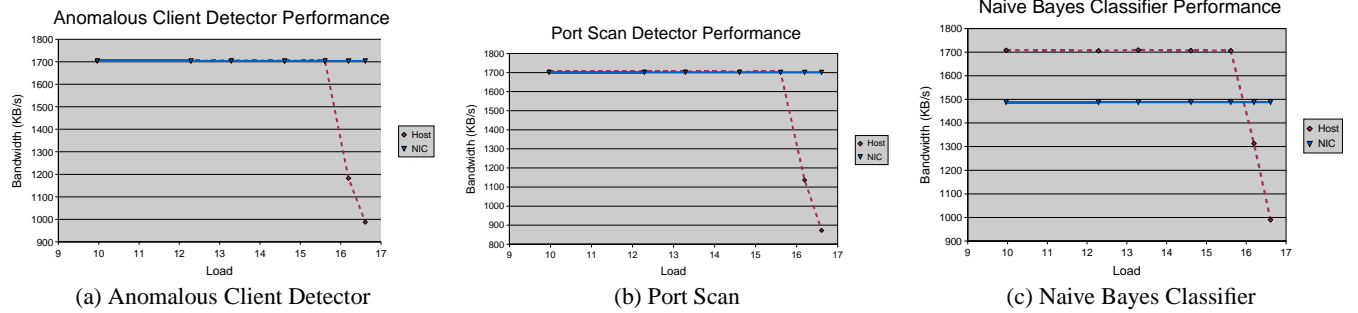


Figure 6: Quantitative Comparison of NIC-based and Host-Based Intrusion Detection

We next compared the performance of the host-based and NIC-based strategies. The results are documented in Figure 6. The X-axis in all cases documents variation in host load. The Y-axis is the net throughput of valid packets (allowed to filter through). Recall that the maximum throughput is limited by the incoming data rate which is 1796 KB per second.

For both the anomalous client detector and port scan algorithms we notice that the NIC-based and Host-based strategies perform comparably until the host becomes overloaded. The NIC-based strategy is unaffected by the load and performs at a relatively constant rate allowing for a 95% throughput efficiency.

For the NBC algorithm we notice that the Host-based strategy performs better when the host is not loaded. Since the computation involved is larger, the NIC-based strategy suffers from the fact that a very slow processor is computing the results. However again we notice that when the host is loaded the performance of the host-based strategy degrades while the NIC-based strategy continues to perform at a constant rate allowing for an 84% throughput efficiency.

4.3 Qualitative Experiments

For the port scan, preliminary results showed that a value of 3072 for S would be suitable given the parameters used to generate the test data. Varying the parameter F in the port scan detection algorithm not only affected the speed of the algorithm but the quality of its results as well. Figure 7 documents the port scan algorithm's behavior when $F = 32768$. The lower function in the figure represents the test data, and is high when a port scan is in progress and low when one is not. The upper function represents the algorithmic behavior, and is high when the algorithm detects a port scan and low when it does not. The figure shows that there is a lag between when a port scan begins and when it is actually detected by the algorithm. This lag is directly proportional to the parameter F , as seen in figure 8. However, decreasing the value of F makes the detector more sensitive to noise, as can be seen in the table in figure 9. Noise can cause the number of ones in the OR of the bit vectors to rapidly fluctuate above and below the threshold value S , causing the detector to find multiple phantom port scans that in reality correspond to a single port scan.

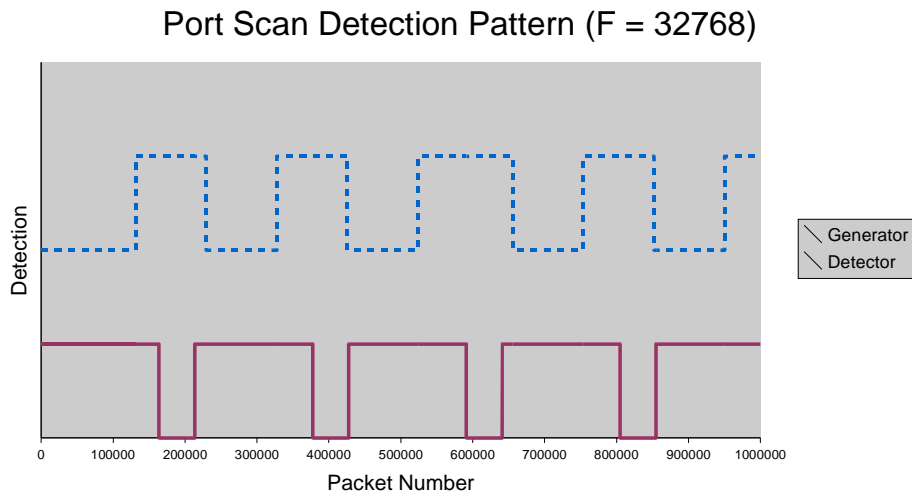


Figure 7: Port Scan Detector Behavior

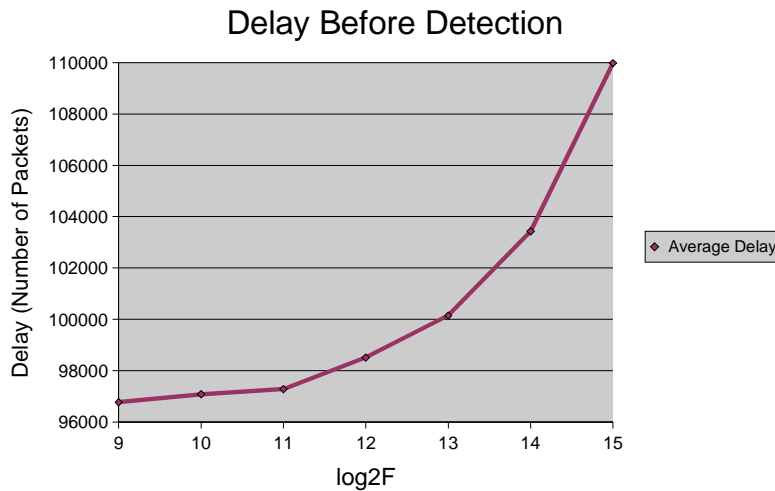


Figure 8: Different Values for F Affect How Quickly Port Scans Are Detected

An accuracy table and a typical confusion matrix are provided in figure 10 to show the quality of the results generated by the Naive Bayes packet classifier. For the given types of intrusions the classifier was trained on, and the features used to do the classification, the results are very good. Figure 11 also contains an accuracy table and a confusion matrix showing the quality of the anomalous client detector algorithm. The dimensions of the SDTable were 256 by 128 (128 slots for each of the 256 local hosts), the hash function H used was the sum of the first three bytes of the source IP address modulo 256, and the threshold used was a constant value of 0.05. Since the last byte of the IP

<i>F</i>	<i>Number of Scans Detected</i>
512	7
1024	7
2048	7
4096	6
8192	5
16384	5
32768	5

Figure 9: Different Values for F Affect How Many Port Scans Are Detected

address (the byte that signifies the actual host on a network) is ignored, we are in fact only calculating that some host from a given network will connect to a local host. This makes it easier to distinguish known clients from unknown clients because all known clients belong to a single class C network. However, by discarding the last byte of the IP address, we are no longer able to calculate the rates at which a given known client connects to a given local host.

<i>Class</i>	<i>Accuracy</i>		<i>Good</i>	<i>Bad</i>
Good	100.00%	<i>Good</i>	105118	0
Bad	92.47%	<i>Bad</i>	67545	827337

(a) Accuracy

(b) Typical Confusion Matrix

Figure 10: Quality of the Naive Bayes Packet Classifier

<i>Class</i>	<i>Accuracy</i>		<i>Good</i>	<i>Bad</i>
Known Client	100.00%	<i>Good</i>	899486	0
Unknown Client	99.21%	<i>Bad</i>	790	99724

(a) Accuracy

(b) Typical Confusion Matrix

Figure 11: Quality of the Anomalous Client Detector

5 Conclusions

We present and evaluate a NIC-based network intrusion detection system. We consider embedding both signature detection algorithms as well anomaly detection algorithms in our evaluation.

The quantitative improvements we achieve with this approach relies on the fact that the operating system of the host does not have to be interrupted with the detection process. Thus on heavily loaded hosts admissible network traffic proceeds at a consistent rate provided the computational and memory resources of the NIC is not stretched. Our preliminary empirical results bear this out. A key element in understanding the tradeoff's involved is the amount of computation and memory utilization involved in the programs. The larger the computation cost, the better the performance of a purely host-based approach.

From the qualitative angle, the downside of using simplified algorithms is that the quality and detection rate are hampered. However, the benefit of having the NIC do the policing is that it can actually prevent network-based intrusions from wrecking havoc on host systems. Since the intrusive packet, if caught, never reaches the host operating system, this approach can detect and prevent, unlike host-based systems. In effect, the NIC acts as a basic shield for the host. If the NIC cannot catch up with the rate the packets are arriving, it can begin dropping the packets (this is the default behavior in the Myrinet NICs), as this may be indicative of a denial-of-service attack. If the NIC were to become overwhelmed by a such an attack, the host would be spared from it. We would prefer to sacrifice only the NIC to the attack rather than the entire host machine.

However, from a technology perspective we are not very far away from 1GHz NIC processors (with appropriately larger memory). With those projected systems we anticipate that NIC-based intrusion detection will do better both from a quantitative standpoint and from a qualitative standpoint (as we will be able to use less-restrictive algorithms). In terms of ongoing and future work, we are looking into taking the algorithms proposed and porting them onto the

newer Myrinet cards featuring faster LANai 9 processors that are now available for use in our group. We are also looking at how to use existing real intrusion data (for example, the KDDCUP and DARPA datasets) to evaluate our algorithms. Finally, we are looking at the problem of implementing incremental techniques[4, 19, 24] on the NIC platform, to update the information used to detect intrusions (as described in Section 3).

References

- [1] D. Barbara and S. Jajodia, editors. *Applications of Data Mining in Computer Security*. Kluwer, 2002.
- [2] D. Barbara, N. Wu, and S. Jajodia. Detecting novel network intrusions using bayes estimators. In *Proc. SIAM Intl. Conf. Data Mining*, 2001.
- [3] J. Anderson D. Gallatin A. Lebbeck A. Chase and Yocum K. Network i/o with trapeze. In *Proceedings of 1999 Hot Interconnects*, 1999.
- [4] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Micha el Wimmer, and Xiaowei Xu. Incremental clustering for mining in a data warehousing environmen t. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 323–333, 24–27 1998.
- [5] M. Martin R. Owa T. Fiuczynski and Bershad B. On using intelligent network interface cards to support multimedia applications. In *Proceedings of NOSSDAV’98*, 1998.
- [6] S. Forrest, S. Hofmeyr, and A. Somayaji. Computer immunology. *Comm. ACM*, 4(10):88–96, 1997.
- [7] S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longstaff. A sense of self for unix processes. In *Proc. of 1996 IEEE Symp. on Computer Security and Privacy*, 1996.
- [8] Yocum K. and Chase J. Payload caching: High speed data forwarding for network intermediaries. In *2001 Usenix Conference*, 2001.
- [9] Wenke Lee and Salvatore J. Stolfo. A framework for constructing features and models for intrusion detection systems. *Information and System Security*, 3(4):227–261, 2000.
- [10] Fiuczynski M. and Bershad B. SPINE: A safe programmable and integrated network environment. In *Proceedings of the Eighth ACM SIGOPS Workshop*, 1998.
- [11] M. Mahoney and P. Chan. Learning nonstationary models of normal network traffic for detecting novel attacks. In *SIGKDD*, 2002.
- [12] D. Nagle and D. Friedman. Building firewalls with intelligent network interface cards. In *CMU SCS Technical Report CMU-CS-00-173*, 2002.
- [13] R. Noronha and N. Abu-Ghazaleh. Early cancellation: An active nic optimization for time warp. In *16th Workshop on Parallel and Distributed Simulation (PADS)*, 2002.
- [14] R. Noronha and N. Abu-Ghazaleh. Using programmable nic’s for timewarp optimization. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2002.
- [15] M. Otey and S. Parthasarathy. Experiences with intrusion detection algorithms. In *OSU-CISRC-8/02-TR21, in preparata*, 2002.
- [16] D. Buntinas D. K. Panda and P. Sadayappan. Fast nic-based barrier over myrinet/gm. In *Int’l Parallel and Distributed Processing Symposium (IPDPS)*, 2001.
- [17] D. Buntinas D.K. Panda and W. Gropp. Nic-based atomic operations on myrinet/gm. In *SAN-1 Workshop, to be held in conjunction with High Performance Computer Architecture (HPCA) Conference*, 2001.
- [18] S. Parthasarathy and A. Ramakrishnan. Parallel incremental 2d-discretization on dynamic datasets. *International Conference on Parallel and Distributed Processing Systems*, 2002.
- [19] S. Parthasarathy, M. Zaki, M. Ogihara, and S. Dwarkadas. Incremental and interactive sequence mining. ACM Conference on Information and Knowledge Management (CIKM), Mar 1999.
- [20] V. Paxson. Bro: A system for detecting network intruders in real-time. In *Proc. 7th USENIX Security Symp.*, 1998.
- [21] Krishnamurthy R. Schwan K. West R. and Rosu M. A network co-processor-based approach to scalable media streaming in servers. In *Proceeding of the International Conference on Parallel Processing (ICPP’00)*, 2000.
- [22] M. Roesch. Snort – lightweight intrusion detection for networks. In *USENIX LISA*, 1999.
- [23] A. Gulati D. K. Panda P. Sadayappan and P. Wyckoff. Nic-based rate control for proportional bandwidth allocation in myrinet clusters. In *Int’l Conference on Parallel Processing*, 2001.
- [24] A. Veloso, W. Meira, M. Carvalho, B. Possas, S. Parthasarathy, and M. Zaki. Mining frequent itemsets in evolving databases. SIAM International Conference on Data Mining, 2002.