DISTRIBUTED DATA PARALLEL TECHNIQUES FOR CONTENT-MATCHING INTRUSION DETECTION SYSTEMS

G. Chapman J. Cleese E. Idle

ABSTRACT

Content matching is a necessary component of any signature-based network Intrusion Detection System (IDS). These packet inspections typically require considerable delay often consuming more than 70% of the IDS processing time. Unfortunately, this delay becomes more significant as security policies and network speeds continue to increase. This paper introduces a new parallel IDS content matching technique that provides initial packet inspections with less delay. The technique distributes portions of a packet payload across an array of n processors, each responsible for scanning a smaller amount of original payload. Given this design, each processor has less data to inspect thus reducing the overall delay. Unlike similar parallel approaches, our technique ensures that security is maintained (no false negatives). Furthermore, the proposed parallel technique is shown to result in an initial match speed-up of approximately $1.25n$ *using Snort (an open source IDS), actual IDS policies, and traffic traces – a significant improvement over current parallel techniques.*

KEYWORDS

Aho-Corasick, Data Parallel, Intrusion Detection, Packet, Parallel, Signature Matching, Snort, Wu-Manber

INTRODUCTION

Intrusion Detection Systems (IDS) inspect arriving packets for malicious content (signatures) as defined by a security policy. Unfortunately, comparing packet headers and payloads against a policy can be complex and time-consuming. For example, it has been found that content matching (scanning for signatures) accounts for more than 70% of the packet processing time [2], [7]. Therefore, given the rising number [5] and sophistication of network threats, these systems will be forced to operate at much faster speeds in the near future.

One approach for reducing the content matching time is the use of better searching algorithms. Three string searching algorithms are commonly used for IDS content matching, Aho-Corasick [1], Boyer-Moore [4], and Wu-Manber [13]. The Aho-Corasick algorithm uses a finite state machine for string matching and provides the best worst case performance, but requires a significant amount of memory for the state machine. In contrast, Boyer-Moore provides the best performance when searching for a single signature [6], but scales poorly. The Wu-Manber algorithm can be viewed as a multi-pattern version of Boyer-Moore, which requires less memory than Aho-Corasick and provides a better average case performance. Although the inclusion of these multi-pattern search algorithms and the development of new IDS specific search algorithms have greatly improved performance, it may not be sufficient for the next generation of high speed networks.

Parallel content matching offers a scalable method for inspecting packets in a high speed environment [3], [8], [14]. These systems typically consist of an array of processors that are used to process packets in parallel. For example, a simple parallel approach would distribute the arriving packets evenly across the array of processors, each having a copy of the complete policy [8]. Using terminology borrowed from parallel computing, this is considered a data parallel approach. While this technique reduces the amount of packets per processor, load balancing and maintaining state (sending packets of one connection to the same processor) is difficult [8]. In contrast, another parallel approach divides the packet payload across the array such that each processor inspects a smaller piece of the original packet payload [3], [14]. Although potentially faster, if a signature spans multiple processors it will not be found [3]. This security issue can be resolved if the data assigned to processors *overlaps* such that each processor can observe the entire signature [14]. However, as the overlapped data must be scanned more than once this greatly increases the content search time.

This paper describes a new parallel content matching approach, called *Divided Data Parallel* (DDP), that divides the payload of a packet across an array of processors as described in the preceding paragraph [3],

1-4244-1513-06/07/$25.00 ©2007 IEEE

[14]. However, the proposed method incorporates a lightweight synchronization system that mitigates the impact of data overlaps. The match-bit allows one processor to quickly indicate to other processors that a match has been found for a given packet. Once the notification has been received, the remaining processors can start inspecting another packet. This modification causes DDP to perform significantly better than other data parallel methods. Experimental results were taken using Snort (an open source IDS). Snort policies for web-traffic, and actual traffic traces indicate a traditional data parallel approach using n processors, for content matching, reduces the content matching time by approximately $0.75n$. In contrast, the DDP algorithm reduces the content matching phase by $1.25n$.

The remainder of this paper is organized as follows. The second section gives a thorough overview of Network Intrusion Detection Systems (NIDSs), Snort rule syntax, and Snort detection. The third section covers previous parallel techniques, while the fourth section introduces the new DDP approach.The fifth section shows results of the DDP algorithm compared to other algorithms, and the last two sections summarize the contributions of this paper, and introduce the areas of future research.

NETWORK INTRUSION DETECTION USING SNORT

As described in the introduction, the purpose of a signature-based IDS is to detect malicious behavior on a computer system or network. This is accomplished by scanning packet payloads for known patterns or *signatures*. While this type of IDS requires signatures be know a priori, it remains an important component for securing computer systems. Snort, one of the most popular IDSs, is an open-source project developed and maintained by Sourcefire[9]. It is commonly used by both research projects and commercial products because of its ease of use and versatility. Snort can perform real-time traffic analysis, content matching, and it can detect multiple types of attacks.

Figure 1 diagrams the five primary processing stages in Snort. During the first stage Snort receives packets via `libpcap` from either the network or a user defined trace file. Once the packets are captured, they are sent through an immediate decoding process, which fills a structure based off of the packets protocol. After packets are decoded they are sent through a preprocessing stage, which performs packet analysis and reassembly. For example, if a TCP packet is captured but has a malformed header, the preprocessor can drop the packet from the system [10]. After preprocessing, content normalization occurs. For example, telnet and HTTP are two types of traffic that are normalized.

Once normalization is complete, the data moves to the most time-consuming stage of Snort, the content matching stage. In this stage Snort's detection engine uses a robust string searching algorithm (described in the introduction) to compare each packet's payload with rules from a signature file. The signature file contains a list of known malicious signatures, and upon scanning, if a signature is matched the alert engine is notified. A Snort rule expresses the action to perform on matching packets/streams. For example, consider the following rule in Figure 3.

Each Snort rule consists of three components. The first identifies the action that must be taken if there is a match. The second denotes the primary match criterion. In Figure 3, the match criterion identifies any TCP packet destined for the `10.1.1.0/24` address space and port 222. The third component contains rule options and describes any additional match criteria (for example, patterns in the payload) and parameters for executing the action. In Figure 3, Snort would search for the hexadecimal pattern `"00 11 22 33 aa"` in the payload. If it is also a match, the message `"rpcd request"` is generated. Snort allows the specification of packet header and payload match criteria. It is important to note that the rule options may contain multiple patterns and/or specifications. Snort will first search for the longest pattern for each rule, called the *initial match*. If there is an initial match it is then *verified* by searching for any additional content and/or verifying specifications described by the rule option.

If malicious data is found in the payload the alert engine is notified. The multi-stage approach of Snort is effective in detecting malicious packets and with parallel techniques Snort can be used on networks with high line speed demands.

PARALLEL SIGNATURE MATCHING TECHNIQUES

The signature matching stage of Snort accounts for more than 70% of the processing time [2], [7]. The use of faster searching algorithms has reduced the signature matching delay; however these solutions are not sufficient for the increasing number of policy rules and network speeds. One scalable solution for reducing the signature matching delay is the use of parallelization. A parallel IDS consists of an array of n processors (this

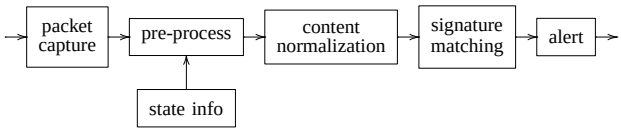

Figure 1. High-level processing stages of Snort IDS.

```
alert  udp any any -> 10.1.1.0/24 222  (content:"|00 11 22 33 aa|"; msg:"rpcd request";)
```

action | primary match | rule options

Figure 2. An example Snort rule that consists of three parts: action, primary match, and rule options.

may be an array of computers, processors, or processor cores). Using concepts developed for parallel computing, the array can be configured two ways, function parallel or data parallel.

In a function-parallel system the policy rules are distributed across the array of processors, therefore each processor has a smaller *local policy*. The data (packet payload) is then duplicated across the array of processors and every processor searches the data for a smaller number of signatures (defined by the local policy). This approach typically reduces the delay since all the processors are used to process the data. However as described in [12], this form of parallelism does not reduce the signature matching delay. This is primarily due to the use of the multi-pattern search algorithms, such as Wu-Manber and Aho-Corasick. The performance of these algorithms is sub-linear with respect to the number of patterns. The search delay for 100 patterns is not substantially more than the delay for 10 patterns [12]. Therefore, distributing the rules across each processor only minimally reduces processing delay.

In a data parallel configuration, each processor in the array has the same policy (same signatures). The data is then sent to one processor, such that each processor has $\frac{1}{n}$ of the original load (load balancing is the objective) [11]. A speed-up of 1.94 for 2 processors and 3.48 for 4 processors was observed using a simple data parallel signature matching technique [11], [12]. However, the previous experiments distributed packets in a round robin fashion, which is difficult to perform in real-time at high speeds because the state information required per packet flow. Although the actual speed-up may be smaller than observed experimentally, these results indicate that a simple data parallel approach can improve system speed in a scalable fashion.

Divided Data Parallel Signature Matching

Another form of data parallelism divides the payload of each packet across the array of processors. Each processor inspects a different portion, or fragment, of the same packet, but collectively the entire packet payload is inspected [3], [14]. There are several advantages to this approach. First, many of the searching algorithms, such as Aho-Corasick, are bounded by the amount of data to be inspected. Therefore, reducing the amount of data per processor should reduce the inspection time. Another important advantage of the divided data parallel method is the ability to maintain state. Since a packet is inspected by every processor, state information can reside on any processor. Using these techniques, the performance is potentially better than the previous data parallel design.

Unfortunately, signatures can be found only if they completely exist within a packet fragment. If a signature spans multiple fragments then it will not be found, resulting in a false negative [3]. As seen in Figure 3, this can be avoided by duplicating data across the processors, called *overlap*, such that a processor can observe the signature [14]. Let T be the packet payload consisting of an array of m bytes (or characters). Furthermore, let p be the number of bytes in the largest pattern, or signature, to search for in T. Each processor is assigned s bytes to search in T, where $s \geq p$ since each processor must be able to observe the largest signature.

How the fragments overlap is critical to the performance and has not been directly addressed by previous work [3], [14]. Given the largest pattern size is p, then consecutive fragments must overlap by, or have in common, $(p-1)$ bytes. Given the values of s and p, the first fragment consists of bytes T_0 through T_{s-1}, while the second fragment would consist of bytes $T_{s-(p-1)}$ through T_{2s+p}. The second fragment starts at byte $T_{s-(p-1)}$ since the last byte of the pattern may be located at $s+1$; therefore, this presents the need of a $(p-1)$ byte overlap with the previous fragment. This is depicted in Figure 3, which consists of an 18 byte packet, a fragment size of 6 bytes, and maximum pattern of 3 bytes. If the signature ($p = 3$) starts at byte 3 then it will be

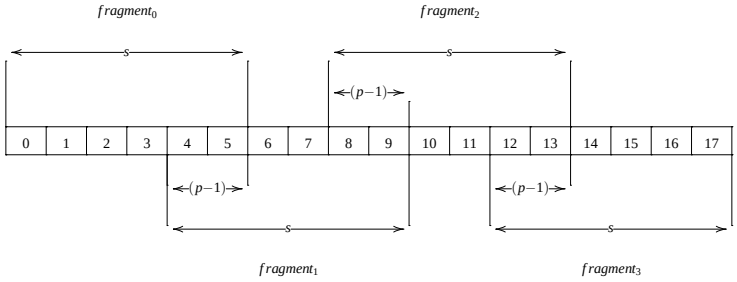

Figure 3. Packet payload consisting of 18 bytes ($m = 18$), fragment size of 6 bytes ($s = 6$), and maximum pattern length of 3 bytes ($p = 3$). Using Equation 1, the minimum number of processors required to process the packet in s time units is 4.

found in fragment 0; however, if the signature starts at byte 4 it will be found in fragment 1. Therefore the $(p-1)$ overlap ensures the signature can be detected anywhere between consecutive fragments. In general, the i^{th} fragment consists of bytes $T_{i(s-(p-1))}$ through $T_{i(s-(p-1))+s-1}$. It is possible that the last fragment is shorter than the others. If it is shorter than p, then these remaining bytes are added to the previous fragment.

A simple analysis of the amount of time required to process a packet payload can be done with the following assumptions. Assume that every fragment contains s bytes and each byte requires one time unit to process. Furthermore, assume all processors must inspect every byte of their fragment and all processors must start at the same time. Given these assumptions the total amount of time required to process the entire packet payload is s. Given m bytes in the packet payload then the minimum number of processors required to process the packet in s units of time is

$$n_s^* = 1 + \left\lfloor \frac{m-s}{s-(p-1)} \right\rfloor \quad (1)$$

As seen in Figure 3, the last fragment is not overlapped. The remaining $m-s$ bytes of the packet are divided across other processors such that each processor has $s-(p-1)$ unique (non-overlapped) bytes as compared to its leftmost neighboring fragment. This equation can be solved for s to determine the appropriate fragment size given a fixed number of processors which typically the case.

As seen in Equation 1, the amount of time required to process the packet payload decreases as s decreases which also requires more processors. The shortest amount of time required to process the packet occurs when $s = p$. Therefore the minimum number of processors required to process the packet payload in the shortest amount of time is

$$n_p^* = 1 + (m-p) \quad (2)$$

Note when $s = p$ every byte, except for the first and last, is inspected by more than processor since it is contained in more than one fragment. Having more processors than defined by equations 1 or 2 will not decrease the processing time. As explained in the next section, if there are more processors than required it is possible to use them to inspect other packets.

Although overlapping does eliminate false negatives, it also increases the search time. For example an 8 processor system only decreased the search time by 60% as compared to a single processor machine [14]. Overlap portions of the payload are inspected multiple times, while a single processor would only inspect each byte of the payload once. As a result the simple data parallel approach described in [11], [12] provides better gains than the current divided data parallel method.

A New Divided Data Parallel Signature Matching Approach

As previously described, the divided data parallel technique has several advantages but the search time is increased when overlap is used. This section introduces a new divided data parallel approach that significantly reduces search time while still eliminating false negatives.

As seen in Figure 4, the new proposed divided data parallel system consists of an array of n processors, each implementing the same policy. As described in the previous section, the payload of an arriving packet is divided across the processors such that each processor inspects only a smaller portion, or fragment, of the original payload. Note, the fragments do contain overlap data to ensure a signature can be detected at any location within the packet.

Fragments are queued at the processors where they are *independently* inspected for signatures. Thus this new design allows processors to inspect different packet fragments simultaneously, which improves performance since the processors need not be synchronized. For

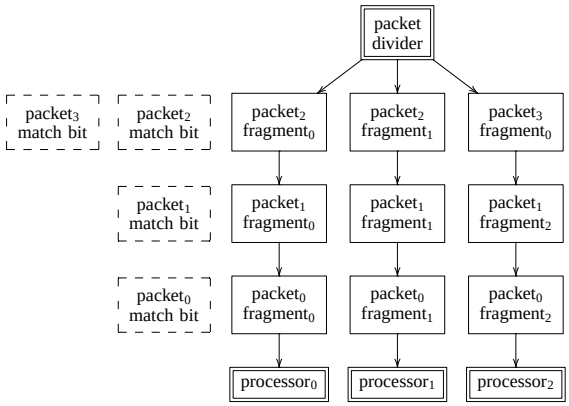

Figure 4. A new proposed divided data parallel system where a packet is divided into fragments then forwarded to an array of processors. A match bit is associated with each fragment to indicate weather a match has been found and allow the processors operate independently.

example, in Figure 4 processors 0 and 1 may inspect fragments from packet 2, while processor 3 is inspecting a fragment from packet 3. This form of *pipelining* significantly improves performance.

If an initial match is only considered, as done in previous divided data parallel techniques, then the performance can be further improved by allowing fragments to be ignored if an initial match has already been found. To provide this functionality, a *match bit* is associated with **each** packet and is tested by the processor before its fragment is inspected. Initially set to false, a match-bit for a packet is set to true if a processor finds a pattern match with an associated fragment. If the match-bit associated with a packet is true, then the processor can ignore any fragments associated with that packet. This also helps the processors to operate more asynchronously since they can quickly ignore certain fragments.

Fragment distribution also impacts the system performance. Consider a simple round robin approach, where each i^{th} portion of a packet is assigned to the i^{th} processor. Since packets have different lengths, the number of fragments may be less than the number of processors. A simple round robin distribution ensures the first processor will always have a fragment while the last processor may not; therefore some form of load balancing is needed. Assigning a fragment to the next unused processor provides load balancing, but experimental results indicate a random distribution provides similar performance with minimal overhead. The proposed improvements, asynchronous search and first match notification, provide significantly faster inspection times while eliminating false negatives.

EXPERIMENTS AND RESULTS

The performance of the parallel content-matching approaches were evaluated experimentally using an eight core, shared memory, Linux-based computer. The packet signature match component of Snort 2.6.0 was changed to perform either a data parallel or a divided data parallel search, and was changed to measure the search time. The number of processors used for the divided data parallel methods were determined using Equation 1. As done in the previous divided data parallel research, all experiments (data parallel and divided data parallel) only performed an *initial match*, which searches for the longest pattern in each rule.

Each experiment used the web and HTTP content rules supplied at the Snort web-site, which consisted of 344 rules total. The maximum pattern length, p, was 80 bytes. The packets used for inspections consisted of 3 days of actual web-traffic sent to a web-server located at a major research university. Experiments measured the speed-up as compared to a single processor machine. In addition, the results of the inspections were compared to a single processor to ensure no false positives or negatives occurred. The performance of different match algorithms for the parallel approaches was measured as well as the impact of different packet sizes.

The first experiment compared the performance of the divided data parallel approach using: no overlap (which results in false negatives) [3], overlap [14], and overlap with the match-bit (the new approach proposed in this paper). The Aho-Corasick algorithm was used for content matching in each experiment. Figure 5 shows the speed-up as the number of processors increased for the three different divided data parallel approaches. When neither the match-bit or overlap were used the results are slightly better than a normal data parallel

approach. Unfortunately as previously discussed, this approach results in false negatives (number of matches were fewer than when using a single processor). As seen in the figure, the performance drops when overlap is used to eliminate false negatives. At 4 processors the speed-up is only 3.1, while at 8 the speed-up is 4.9. Therefore the inclusion of overlap had a negative impact on performance. The use of overlap and the match-bit provided the best performance. This divided data parallel approach resulted in a 2.66 speed-up with 2 processors and 10.65 with 8 processors. On average this corresponds to a $1.25n$ speed-up, where n is the number of processors. The speed-up is greater than n because of *pipelining*, since processors can inspect fragments from different packets in parallel. This additional form of parallelism provides a significant increase in performance.

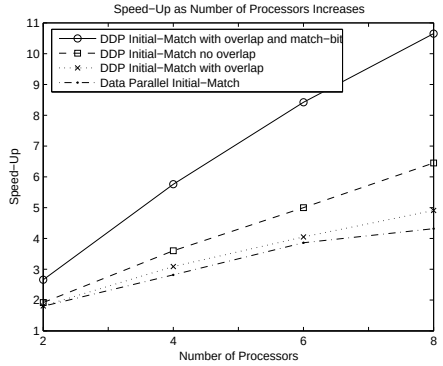

Figure 5. Speed-up of the different parallel approaches, each using the Aho-Corasick content matching algorithm.

The next experiment compared the performance of different search algorithms (Wu-Manber and Aho-Corasick) using the proposed divided data parallel approach (overlap and match bit) and standard data parallel. In general, Wu-Manber provides reasonably fast content matching with a small memory requirement, while Aho-Corasick provides faster search times but requires more memory to store necessary data structures. Both are available in the current version of Snort. As shown Figure 6, data parallel using the Wu-Manber algorithm has a speed gain of 4.48 times when 8 processors are used while DDP using Wu-Manber has a speed gain of 9.05 when 8 processors are used. Data parallel using Aho-Corasick has a speed gain of 3.4 with 8 processors while DDP using Aho-Corasick has a speed gain of 10.65. These results indicate the new proposed divided data parallel method outperforms data parallel using any signature matching algorithm. In addition, like the traditional data parallel method, the performance of the proposed divided data parallel method is independent of the signature matching algorithm. This is expected since the data parallel paradigm divides the data stream to be processed, not the processing method for the data stream.

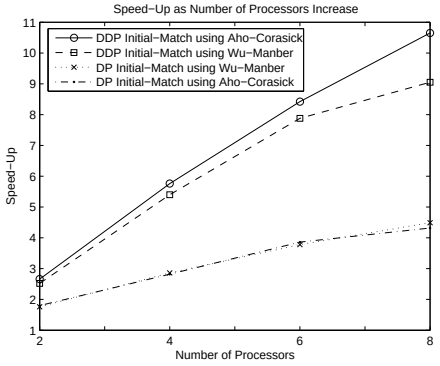

Figure 6. Graph showing the speed-up of the parallel algorithms

The last experiment measured the effect of different packet size on the divided data parallel approach. The largest pattern length was 20 bytes and the packet payloads were 1360, 680, 340, and 170 bytes. These payload amounts give equal length fragments. Note that for each doubling in packet size the number of packets used is halved. This is to maintain a consistent amount of data throughout the experiments.

Assuming every byte of data must be processed, the total amount of bytes inspected per group of processors is computed by using the following equation, where m is the packet size in bytes, n is the number of processors, p is the size of the largest pattern in bytes, and k is the total number of packets.

$$(m+(n-1)*(p-1))*k \tag{3}$$

Figure 7 is a graph of the speed-up as the packet size changes; note that these speed-up rates mirror the values computed using Equation 3. The larger packets perform the best, because a smaller portion of the packet is overlap, whereas small packets perform worst because they contain a larger portion of overlap.

CONCLUSIONS

This paper introduced a new divided data parallel method that builds upon the work in [3], [14] and shows speed gains greater than data parallel [11], [12]. The new method still divides the packet payload into fragments and uses overlapping to prevent false negatives. However, unlike other divided data parallel approaches these fragments can be processed independently. The

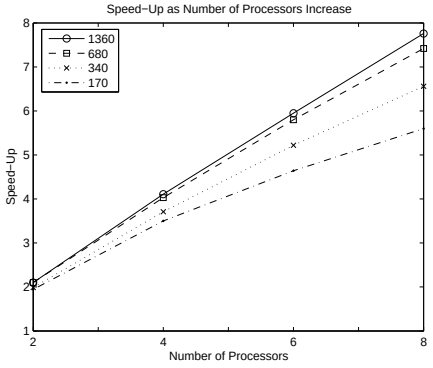

Figure 7. Time to process packet as packet size changes

new method associates a *match bit* with each packet. Initially set to false, the match bit indicates if a match has been found in the packet. If a processor finds a match, then the associated match bit is set to true. Before a processor inspects a fragment the match bit is checked, if it is already true then the fragment is not inspected and the processor moves to the next fragment in its queue. This allows processors to operate independently since certain fragments can be ignored. As a result the system permits pipelining since processors may process fragments from different packets.

Experimental results using Snort (an open source IDS) and actual traffic traces indicate the new divided data parallel method results in a speed-up of approximately $1.25n$ where n is the number of processors; whereas previous work was only able to achieve a speed-up of $0.75n$. Furthermore, the new approach is independent of the content matching algorithm. The new divided data parallel method is a scalable technique that performs better than current methods.

FUTURE WORK

Although the new divided data parallel method has shown great promise, there are several areas for future work. One area important for all divided parallel techniques is the support of match verification. To provide this functionality a two tiered system could be used, where the divided data parallel system sends all initial matches to a secondary IDS for verification. Further, a dynamic metric for determining the packet split at runtime can be developed. The packet can be split in varying ways based upon the packet size and the maximum pattern length. This research focused on the content matching phase, future research can focus on the entire system as a whole. The cost of fragmentation and distribution will affect the speed of the system, research needs to determine how detrimental this cost is.

REFERENCES

[1] Alfred V. Aho and Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Communications of the ACM*, 18(6), June 1975.

[2] S. Antonatos K.G. Anagnostakis and E. P. Markatos. Generating realistic workloads for network intrusion detection systems. In *Proceedings ACM Workshop on Software and Performance.*, 2004.

[3] Herbert Boss and Kaiming Huang. A network intrusion detection system on ixp1200 network processors with support for large rule sets. `http://citeseer.csail.mit.edu/703003.html`.

[4] Robert S. Boyer and J. Strother Moore. A fast string searching algorithm. *Commun. ACM*, 20(10):762–772, 1977.

[5] Cert. Cert/cc statistics 1988-2006. `http://www.cert.org/stats`.

[6] Richard Cole. Tight bounds on the complexity of the boyer-moore string matching algorithm. *Symposium on Discrete Algorithms*, pages 224–233, 1991.

[7] Mike Fisk and George Varghese. Fast content-based packet handling for intrusion detection. Technical report, University of California at San Diego, 2001.

[8] C. Kruegel, F. Valeur, G. Vigna, and R. Kemmerer. Stateful intrusion detection for high-speed networks, 2002.

[9] Sourcefire. Sourcefire network security. `http://www.sourcefire.com/`.

[10] Sourcefire. *Snort Users Manual*, December 2006.

[11] Patrick Wheeler and Errin W. Fulp. A taxonomy of parallel techniques for intrusion detection. *ACMSE 2007*, 2007.

[12] Patrick Stuart Wheeler. Techniques for improving the performance of signature based intrusion detection systems. Master's thesis, University of California Davis, 2006.

[13] Sun Wu and Udi Manber. A fast algorithm for multi-pattern searching. Technical report, University of Arizona, May 1994.

[14] Jianming Yu and Jun Li. A parallel nids pattern matching engine and its implementation on network processor. In *2005 International Conference on Security and Management (SAM 2005)*, 2005.