Two-Stage Decomposition of SNORT Rules towards Efficient Hardware Implementation

Hao Chen, Douglas H. Summerville, Yu Chen*
Dept. of Electrical and Computer Engineering, SUNY – Binghamton, Binghamton, NY 13902

***Abstract**– The performance gap between the execution speed of security software and the amount of data to be processed is ever widening. A common solution is to close the performance gap through hardware implementation of security functions. However, continuously expanding signature databases have become a major impediment to achieving scalable hardware based pattern matching. Additionally, evolutionary rule databases have necessitated real time online updating for reconfigurable hardware implementations. Based on the observation that signature patterns are constructed from combinations of a limited number of primary patterns, we propose to decompose the Snort signature patterns. These smaller primary pattern sets can be stored along with their associations to allow dynamic signature pattern reconstruction. Not only does the matching operation potentially become more scalable, but the real time online updating task is simplified. The approach is verified with patterns from the latest version of the Snort rule database. The experimental results show that after decomposition, a reduction in size of over 77% can be achieved on Snort signature patterns.**

Key Words: Network Intrusion Detection Systems (NIDS), Security, Finite State Machine, Scalability, Decompose, FPGAs.

1. Introduction

In the past decades, the performance gap between the processing requirements of Network Intrusion Detection Systems (NIDS) and their software-based implementations has been widened due to the escalation of sophisticated attack tools and the performance limitations of sequential execution. A common solution is to close the performance gap through hardware implementation of security functions. The major motivation is to pursue more powerful computing capability for the execution of more sophisticated security functions in real time. In addition, by exploring unique features of hardware execution style, such as multi-threaded parallelism and multi-stage pipeline, further improvements could be achieved.

Packet inspection is one of the important fundamental tasks performed by NIDS. It consists of two parts, packet classification that focuses on the packet header and deep packet inspection that examines the payload. Research efforts have focused more on the later, since performing deep packet inspection is more difficult due to the diverse formats of packet payloads.

Pattern matching or signature detection is a fundamental technique for deep packet inspection. By comparing input data with predefined signature patterns, malicious content can be accurately identified. Actually, more than 80% of the Snort rules contain signature patterns and more than 80% of CPU time is taken by pattern match operations [12].

Snort [17] is a well-known software based security application originally designed for lightweight network intrusion detection. Unlike commercial NIDS, in which core parts are hidden for intellectual property protection, Snort is a free open-source NIDS. Its rule database is used to generate regular expressions for intrusion detection. Users are allowed to strengthen the rule database as well as other parts of Snort, and verified rules are collected for the update of its core rule database. Over a ten-year evaluation time, the Snort rule database has earned a good reputation due to its accuracy, comprehensiveness and efficiency.

With the trend of using hardware based applications for security protection in high speed network environments, continuously growing rule databases have become a significant impediment to hardware implementation. Nowadays, hardware solutions are likely to be integrated on a single chip with tight resource constraints. Along with many advantages, the consequent side-effect is that hardware implementations are more resource-sensitive. Without deliberate design methodologies, hardware resources can be quickly exhausted.

At the same time, it is highly desirable that a NIDS can update its rule database in real time, so that newly emerging attacks can be handled promptly. However, this issue is still an open problem, since hardware design is not as flexible as software programming. Modification in reconfigurable hardware implies replacement and rerouting of circuits. In addition, many realistic conditions that are seldom considered in software implementations must be taken into account in a hardware implementation, such as signal delay, fan-in/fan-out and power consumption.

Therefore, it would be ideal if a NIDS could be implemented in a lightweight manner, so that the signature database neither occupies huge memory space nor increases infinitely. Furthermore, the update operation should be done in real time and online, which implies that adding new signatures should not lead to significant replacement and rerouting efforts.

NIDS signatures can vary from being as simple as checking the value of a header field to as complex as calculating the statistical characteristics of a connection or conducting sophisticated protocol analysis. Essentially signatures represent the activities an intruder has to perform to gain access into a computer system. These activities can include such things as launching programs, running scripts, querying databases and following the steps of a protocol.

* Manuscript submitted on May 1, 2009 to *the 7th International Workshop on Design of Reliable Communication Networks (DRCN 2009)*, October 25 – 28, 2009, Washington, D.C., USA. Corresponding author: Yu Chen, Dept. of Electrical & Computer Eng., SUNY–Binghamton, Binghamton, NY 13902. E-mail: ychen@binghamton.edu, Tel.: (607) 777-6133, Fax: (607) 777-4464.

Activities such as these can be broken down into a set of operations, and signatures describe sequences of these operations.

Since intruders must interface through computer, their operation sequences should be input sequences that computer can accept. In practice, basic input sequences are limited, defining a finite set of allowable activities. Any sophisticated input sequence could be disassembled to basic sequences. If an input sequence could be mapped to a signature, a basic sequence could be mapped to a primary pattern. Hence, no matter how large the signature database grows, the number of primary patterns which represent fundamental operations is limited.

This observation implies that the total number of primary patterns is limited if each primary pattern corresponds to an operation. If we can represent the signatures as permutations of these primary patterns, the memory or hardware resources required to characterize the primary patterns appearing in the signature database can be significantly reduced.

By analyzing signature pattern sets extracted from a Snort database, we verified this rationale as we observed that many primary patterns appear repeatedly within signatures. Even when new signatures are added, many of them can be decomposed into the existing set of primary patterns, with infrequent additions. Practically, this implies that a smaller amount of memory or other hardware resources could be required to implement these primary patterns.

Since all patterns are distilled from network traffic carrying data for common protocols and services, it is reasonable to believe that this observation is not only applicable to the pattern set abstracted from the Snort rule database that we studied, but also applicable to pattern sets from other pattern match based Network Intrusion Detection Systems (NIDSs) as well. Though the appearance of signature pattern sets may be different due to diverse environments, the primary pattern sets should be similar.

The remainder of this paper is organized as follows: Section 2 briefly introduces reported work that is closely related to our effort. Section 3 illustrates our observation and presents a two step approach to decompose the Snort signature/rule database. Section 4 verifies our observation through detailed analysis. Section 5 discusses some basic principles of our current ongoing work based on the decomposed Snort signatures. Section 6 summarizes this paper.

2. Related Work

To the best of our knowledge, it was Franklin et al. who first attempted implementing patterns from the Snort rule database on top of reconfigurable hardware [9]. Since the primary focus was on hardware adaptation, many other important issues were deliberately not considered in this design. From the perspective of pattern matching implementation, their rationale was that hardware circuits can be precompiled and the compiling time is not a big issue since patterns are predefined. However, considering today's rapidly evolving network security requirements, this rationale is no longer feasible. Currently, signature pattern updating is crucial to maintaining the reliance of any signature-detection based NIDS. As to reconfigurable hardware applications, recompilation is one of the integrated procedures of update, so it greatly affects the efficiency of these applications. Time is very limited for recompilation in the case of high performance detection. Although they mentioned the scalability issues of patterns due to the growing size of Snort rule database, they did not propose any meaningful solutions.

Up to now, design innovations based on a variety of techniques have been applied to hardware-based pattern matching approaches. However, they have generally suffered from limited hardware resources, especially the tight budget of memory resources to store patterns or perform pattern comparison.

Cho et al. [6] proposed to relieve the scalability issue of pattern comparison and to improve the performance by using 8-to-1 decoders for each byte comparator and instantiating one decoder output for all the same output. The 8-to-1 decoder was setting the character decoder to the first stage of the pipeline. Before conducting a comparison, input data at the character level is decoded to a single bit level; all comparisons are then performed based on these single bits. By this means, they reduced the overall size of the comparators to one-eighth. In circuit design, reducing the number of fan-in also reduces the side effect to gate. Secondly, all the comparators use the same data input and many decoders are exactly the same. Instantiating one for sharing saves hardware resources. To achieve more efficient operation, patterns are divided into prefix and suffix part, and the prefix part was used to index major portion of patterns contained in suffix part. While the prefix part was kept on chip, the suffix part was kept on off-chip ROM. It was obvious that the size of on-chip RAM and the bandwidth in-between memories greatly impact the system performance.

There are reported efforts that focus on the implementation of regular expression pattern matching engine [4], [14]. Specifically, by using Nondeterministic Finite Automata (NFA), instead of centralizing on character decoder, they employed the SRL 16 module [7], and they also explored certain common prefix sharing techniques. Good experimental results were achieved, containing 500 IDS regular expressions from Snort in 25K logic cells. However, the scalability issue would be prominent with larger numbers of expressions being implemented.

Aldwairi et al. developed a configurable string matching accelerator to speed up the deep packet inspection [1]. They executed software on a general purpose processor for Finite State Machine (FSM) operation with the support of standard RAM. The software generates a FSM from patterns extracted from the Snort rule database, and the FSM is in charge of pattern matching operation. However, both the generation and operation of the FSM for pattern matching would become more complicated as the number of patterns increases. In addition, although they tried to increase throughput by increasing on-chip RAM bandwidth, there is a limit to how much bandwidth can be increased.

Instead of completely focusing on pattern comparison for performance improvement, researchers have also proposed to improve performance by taking advantage of some properties in network traffic [13], [2]. They indicated that malicious packets make up only a small share of total traffic. Consequently, they adopted hybrid architectures in which hardware devices handle pre-filtering and PC-based software implements Snort for final identification. It was reported that more than 90% of the workload of traffic processing could be relieved from Snort software through this approach. In addition, this design could sustain more than 10Gbps throughput compared to 1Gbps Snort throughput. Compared with the other approaches, this approach is more flexible and has good scalability. However, the problem of maintaining 1Gbps throughput of software-based Snort is not trivial.

Another approach [3] that is close to our effort attracts our attention due to its good performance on both feasibility and scalability achieved by partitioning Snort patterns. While many hardware-based pattern matching applications focus more on hardware innovation, their investigation focused more deeply into the original pattern sets. The graph-based "min-cut" technique [10] was applied to help achieve optimal design. The basic idea of min-cut partition is that the number of edges between nodes within the group is maximized, and the number of edges between different groups is minimized. After partitioning, optimized pattern structures are implemented in hardware for pattern matching operation. In fact, this is the only work we know of that focuses on the analysis of relationships concealed in Snort patterns to improve the performance of the corresponding hardware processing and to solve scalability issues.

3. Decomposition of SNORT Patterns

Most efforts to improve the performance of hardware-based pattern matching implementations have focused on the innovation of hardware architectures or matching algorithms. However, the original source of the complexity and scalability problems are the characteristics of the signature pattern set. Therefore, it is more effective to gain a deeper insight into these characteristics before investigating hardware architectures.

Our study has revealed that the most significant factor causing Snort signature pattern set inflation is the existence of internal redundancies. If these concealed redundancies can be successfully reduced the size of signature pattern set can be well controlled, so as to relieve the complexity and scalability issues of subsequent implementations. This section introduces a two-step signature pattern decomposition method developed for this purpose.

3.1 Redundancies in SNORT Patterns Sets

The signature patterns used in this work are extracted from the Snort rule database. In addition to these patterns, other packet information is also contained in the database for detection and administrative support such as source/ destination addresses, source/destination ports, protocols, flags and tags. In general, a complete Snort rule consists of two logic sections, the rule header and the rule options [17].

Figure 1 shows a Snort signature selected from its database. The text up to the first parenthesis is the rule header portion, which includes basic packet header information with specific IP addresses replaced by wildcards. The second portion within the parentheses is the rule options, which includes alert messages, pre-identified signature patterns, and other support information for Snort NIDS.

```
alert tcp $EXTERNAL_NET any -> $SQL_SERVERS
1433 (msg:"MS-SQL shellcode attempt";
flow:to_server,established; content:"9 |D0 00 92 01 C2
00|R|00|U|00|9 |EC 00|"; classtype:shellcode-detect;
sid:691; rev:5;)
```

Figure 1. An example of a Snort rule.

Within the rule, the data string being quoted after the "content" is the signature pattern that is sought. "Content" works as a keyword in the Snort rules database, introducing the specific signature patterns for deep packet inspection [17]. An example of such a signature pattern is shown in Figure 2. In this case, the pattern is composed of both text characters and binary data. The binary data is represented in hex format and enclosed within pipe characters (|).

```
9 |D0 00 92 01 C2 00|R|00|U|00|9 |EC 00|
```

Figure 2. A Sample of Snort signature pattern

For convenience, we take a group of signature patterns extracted from the Snort rule database as an example to study its properties. We call such a group of signature patterns a signature pattern set. There are two important properties that exist in a signature pattern set: repetition and composition.

1. **Repetition:** as shown in Figure 3, signature patterns contained in 3 rules out of total 6 are the same. This redundancy implies that increasing the number of signatures may not necessarily lead to a commensurate increase in the number of primary patterns.
2. **Composition:** as shown in Figure 4, each signature can be considered as a combination of smaller pattern fragments, or primary patterns. Thus, a signature pattern can be decomposed into one or more primary patterns.

Essentially, the existence of both repetition and composition imply that redundancies exist in the pattern set. Instead of passing these redundancies to corresponding hardware operations or applying more sophisticated mechanisms for redundancy mitigation at the hardware design level, as many current approaches do, it is more effective to remove these redundancies up front at the time when the pattern set is generated.

Once we carefully peel off these redundancies, not only is the complexity of the corresponding hardware design reduced, but also the scalability of the system is improved. Then, we can maintain the same functionality as the original design

```
|5C|PIPE|5C 00 05 00 0B|

|98 D0 FF|k|12 A1 10|6|98|3F|C3 F8|~4Z

|05 00 0B|

|98 D0 FF|k|12 A1 10|6|98|3F|C3 F8|~4Z

|04 00|

|98 D0 FF|k|12 A1 10|6|98|3F|C3 F8|~4Z
```

Figure 3 Repetition of patterns

```
dbms_repcat.alter_priority_raw

dbms_repcat.alter_priority

dbms_repcat.alter_priority_varchar2

dbms_repcat.alter_site_priority_site

dbms_repcat.alter_site_priority
```

Figure 4 Composition of primary patterns

with fewer patterns implemented in real circuits. Obviously, this approach relieves the conflict between the growing signature databases and limited hardware storage resources.

3.2 Pattern Decomposition

Based on the above observations, we decompose the abstracted pattern set further to remove redundancies. The decomposition consists of two steps corresponding to the properties of repetition and composition. The first step, dealing with repetition in the pattern set, is straightforward. It checks all individual signature patterns inside the pattern set in turn and simply removes repeated patterns. The second step, handling composition, involves further decomposing signature patterns into primary patterns. A primary pattern is a substring of a signature pattern. This step is more difficult since there are many ways one can parse and decompose individual signature patterns into primary patterns.

The most challenging issue in the second step of deep decomposition is balancing the tradeoffs between system performance and redundancy. Signature patterns can be decomposed into primary patterns of various lengths, ranging from single characters up to entire signature patterns. It is naïve and inefficient to decompose the signature patterns down to the character level. Although the total number of characters used, and hence the number of primary patterns, will never exceed the number of ASCII codes (256, with even fewer used in practice [3]), such fine-grained decomposition leads to tremendous overhead in system performance while performing dynamic pattern matching since it is extremely inefficient to reconstruct signature patterns from individual characters.

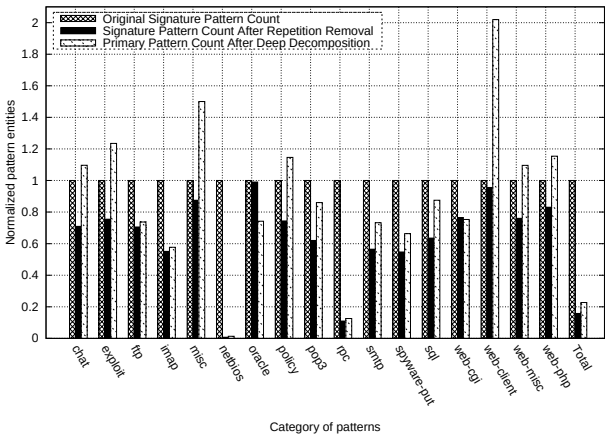

Figure 5.Variation of pattern entities

We conduct pattern decomposition on text-based patterns at the "word" level, which takes advantage of text-based redundancies exhibited in signature sets, such as those shown in Fig. 4. Although pattern decomposition could also be performed on binary data patterns, such as those shown in Fig. 3, methods of delineating primary patterns are not as immediately obvious. Since our primary goal is to verify the correctness and feasibility of the proposed approach, this preliminary analysis of decomposition biased on text dominant patterns is enough.

For decomposition, we have chosen the following heuristic. A string of adjacent characters not containing any "hyphen symbol" between any two characters is considered as a primary pattern, as are the hyphen symbols by themselves. Taking the signature patterns in Fig. 4 for example, "*dbms*", "*repcat*", "*alter*", and "*priority*" are all considered as primary patterns and "_" and "." are considered as "hyphen symbols". In general, any symbol in between two character strings could be considered as "hyphen symbol"; we have chosen underscore ('_'), hyphen ('-'), period ('.') and space (' ') and forward slash ('/') characters to be hyphen symbols. Specifically, in cases when patterns form combinations of both text characters and binary data, any binary data string in between pairs of "|" is considered as an individual primary pattern. Considering the patterns in Fig. 3 for example, "|5C|", "PIPE", "|5C 00 05 00 0B|" are all considered as primary patterns. It worth noting that "|" itself is not a part of patterns in practice. Instead, it is used only as a mark to distinguish between text characters and binary data.

Figure 5 shows the change in the number of pattern entities resulting from the two-step signature pattern decomposition method for various categories of Snort rules, while Fig. 6 shows the change in total storage size. The signature patterns are taken from Snort V.2.3.3 and are divided into 48 categories. To maintain a clear view, pattern categories that contain less than 50 entities are excluded from Fig. 5 and those that contain less than 500 characters total are excluded from Fig. 6. The complete figures are included in our technical report [18]. Within each category, the graphs show the original number of signature entities values (before

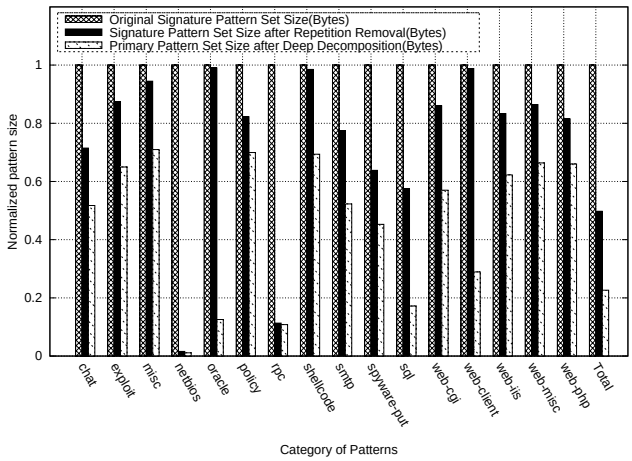

Figure 6. Variation of pattern sizes

two-step processing), the number after redundancy removal and the number of primary patterns after subsequent deep decomposition. To highlight the change within each category, rather than the variations among them, all values have been normalized to the original count (Fig. 5) or size (Fig. 6) of the pattern signatures in the category. Though not evident due to the normalization, there is a wide variation in the number of signature patterns and the total size of the individual categories. The rightmost category in each figure provides the normalized result for all categories combined.

For nearly every signature pattern category in Fig. 5, there is a significant reduction in the number of signature patterns after redundancy removal, which suggest a high rate of reuse of entire signature pattern entities. One of the extreme cases in Fig. 5 is the *netbios* rule category, for which the original number of signature entities (20,087) is reduced to 148 after repetition removal. The *Oracle* and *web-client* rule sets have the lowest redundancy of the pattern categories. The cumulative results for the entire data set show that there are a total of 25,802 signature pattern entities in the database, only 4076 are unique. However, the number decreases to 164 entities and 1355 bytes after deep pattern decomposition, respectively. In Fig. 6 one observes a decrease in size due to repetition removal that is based on the amount of reuse and size of the patterns. The aggregate reduction in size for all pattern categories is from 145,348 bytes down to 72,315 bytes, a reduction of just over 50%.

After subsequent deep decomposition, the pattern entity count represents the number of unique primary patterns present; thus, one would expect the count to increase since each pattern is broken down into substrings and, since each pattern is itself unique, one would expect each to contribute something new to the total. In the case of the *oracle* and *web-cgi* categories, there is a decrease in the number of pattern entities. This suggests the signature patterns are formed from permutations of a smaller set of primary patterns. Even more important than entity count is the size of the resulting primary pattern set (32,898 bytes), which represents a further decrease of 55% after repetition removal, resulting in a total decrease of over 77% for the two step processing method. Normalized pattern sizes

4. Further Analysis and Verification

The fundamental motivation of deep decomposition is that signature patterns are correlated to activities that an intruder must to perform in order to gain access to a computer system. An activity may consist of multiple atomic operations that are mapped to certain primary patterns contained in a signature pattern. Depending on the service being attacked, the activities of an intruder may exhibit a wide range of diverse data formats and/or functionalities; yet for each service, the atomic operations should be limited in practice to the set of operations (inputs) that are recognized by the computer software implementing the service. This set is further limited by the activities that will benefit the attacker. Since the number of services running on a practical machine is expected to be finite, the total number of primary patterns should not exceed a finite upper bound. After reaching some practical size, the number of primary patterns is unlikely to increase significantly with incremental increases in the number of signature patterns in the database.

Assuming that it is feasible to reconstruct signature patterns from a limited number of primary patterns, the end result should be smaller, more efficient hardware implementations. We consider issues related to recomposing signature patterns in section 4.1. To verify the premise that the number of primary patterns is bounded in practice and that this bound is not based on signature database size, we analyze the incremental growth in primary patterns in section 4.2.

4.1 Signature Pattern Reconstruction

To benefit from the reduced size of the primary pattern set, it is critical to find an efficient approach to express and reconstruct the original signature patterns from primary patterns. Such an approach can overcome the conflict between the growing size of signature databases and limited hardware resources, since newly added signature patterns can be decomposed to and reconstructed from existing primary patterns. In another words, the implementation achieves scalability with respect to the number of signature patterns.

Compared to approaches that require more memory space to store new patterns [9], [4], [14], [1], our novel approach has the potential to achieve much better scalability. Compared to approaches that require complex hash functions for pattern condensing [8], [15], our approach is much simpler and more feasible for on-board application.

The problem of efficiently reconstructing signature patterns based on primary patterns resulting from our two-step decomposition procedure is an area of further research. Considering each primary pattern as a single graph node, reconstructing a signature pattern is simply a process of forming a directed graph for each signature. As illustrated in Fig. 7, “*dbms*”, “*repcat*”, “*alter*”, “*site*” and “*priority*” are five primary patterns obtained from examples of Fig. 4.

Figure 7 also illustrates the pattern reconstruction phase,

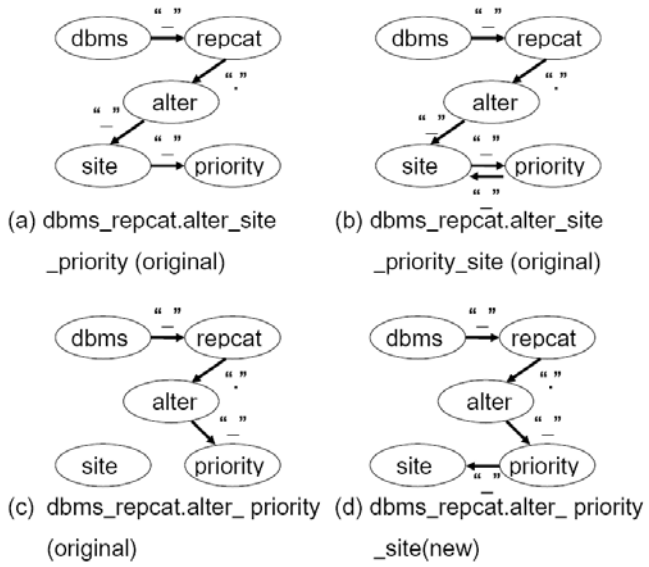

Figure 7. Signature Pattern Reconstruction

in which these primary patterns could be used to recover original signature patterns. Figure 7(a-c) shows how original signature patterns are reconstructed, while Fig. 7(d) shows how a new signature pattern can be reconstructed from the same set of primary patterns. This is the essential feature that we are pursuing: increasing the variation of edges between nodes rather than increasing the number of nodes, allowing different signature patterns to be easily reconstructed. Efficient methods for implementing this reconstruction in hardware are an area of further study.

4.2 Statistical Verification

In order to verify our premise that the number of primary patterns increases at a much slower rate compared to the rapid growth of signature pattern databases, we extended our analysis of pattern decomposition to determine the variations across multiple versions of the Snort rule database. Since these versions represent the evolution of the Snort database over a period of several years, this experiment presents us a clear view on the relationship between growth in the number of signature patterns and growth in primary patterns.

We have analyzed the series of signature pattern sets currently available, representing seven major public versions of Snort rule databases. They include V2.1 to V2.4 and V2.6 to V2.8, which were released over a period from April 2005 to October 2008. Although new types of rules were added to rule databases to accommodate the continuous evolution of network intrusion patterns, major parts of Snort rule databases still follow an accumulating update policy. From V2.1 to V2.2, V2.3 to V2.4 and V2.6 to V2.8, the corresponding types of Snort rules were increased from 48 to 50, and later to 52. However, the sizes of rule database inflated from 935Kb to 8.86Mb.

The number of pattern entities and the size of pattern sets are two key parameters that we are interested in. With these, we are better able to determine the growth relationship between signature patters and primary patterns. By extracting the "content" signature pattern rules from the databases, we obtain the original signature pattern sets, summarized in Table 1. The size of the signature patterns grows from 31.1kB in version 2.1 to 164.2kB in version 2.8. We then further decomposed each set of original signature patterns down to primary pattern sets following the two-step decomposition procedure presented in Section 3.

Table 1. Summary of parameters of signature pattern set and primary pattern set in different versions

Version	Signature Pattern Set		Primary Pattern Set	
	Entities	Size	Entities	Size
2.1	2,739	31.1 kB	2,432	11.7 KB
2.2	3,442	33.0 kB	2,501	11.7 KB
2.3	25,802	145.3 kB	5,856	32.9 KB
2.4	26,936	147.9 kB	5,867	32.9 KB
2.6	31,103	164.1 kB	6,482	37.0 KB
2.7	31,097	163.9 kB	6,525	36.9 KB
2.8	31,156	164.2 kB	6,523	37.0 KB

The plot in Figure 8 illustrates the incremental increase in the size of the decomposed database as the number of signature pattern entities increases. From the most recent set of rules tested (version 2.8), the *netbios* content signature rule set was selected, since it is the largest set of rules.

Because the rules do not contain a timestamp indicating when they were added to the database, the rules were randomized to minimize the effect of grouping within the file. To generate the plot, one additional rule was added to the database at each step, and the size of the database after two-step decomposition was recorded. Thus, the graph shows the incremental growth in the primary pattern set size as the signature set size increases.

Clearly, as the number of signatures increases the growth of the primary pattern set size appears to be decreasing rapidly. The individual points on the plot show where the *netbios* (signature pattern set size, primary pattern set size) pairs fall on this plot.

Note that except for version 2.8, the points do not exactly fall on this curve since the rules were incrementally added in random order and some rules may differ between versions. However, the points do show good agreement, justifying the sampling procedure used to generate the plot.

While Figure 8 supports our premise that after reaching some practical size, the number of primary patterns saturates and is unlikely to increase significantly with incremental increases in the number of signature patterns in the database, we cannot say for certainty that this is indeed the case. Many of the signature pattern sets in the snort database are too small to have reached the point of saturation and the database as a whole, being composed of many types of traffic, also have not yet reached the point of saturation.

Despite this, the signature pattern sets all exhibited sub-linear incremental growth when analyzed. Furthermore,

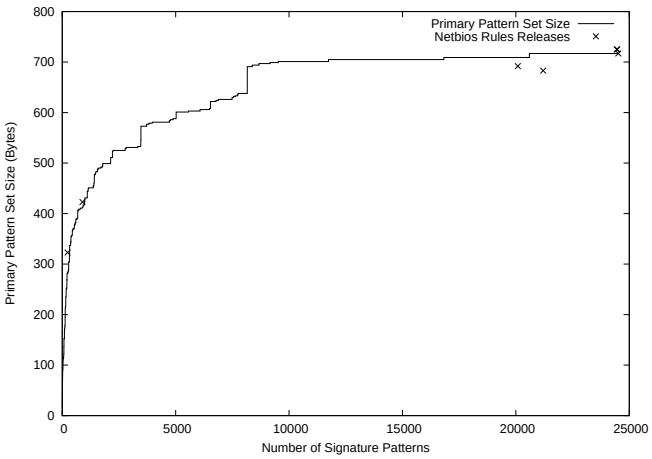

Figure 8. The growing trends of pattern entities between different versions.

even if the database never reaches the saturation point given changes in service types, the overall reduction in size achieved by our two-step decomposition procedure still justifies the use of static signature pattern decomposition and dynamic reconstruction.

5. Discussion

The main advantage of adopting the proposed decomposition and reconstruction based pattern processing technology is able to achieve a good scalability. As we have successfully demonstrated a scalable way to reduce the redundancy in signature pattern sets, the natural extension is to investigate scalable approaches for real-time dynamic updating and high-speed online signature matching. However, as this is ongoing work, only some basic principles are discussed in this section.

5.1 Scalable Pattern Update

Static update refers to the update mode that is pre-scheduled. Due to its easy operation, static update is the most conventional and popular mode used for the update of signature patterns. However, static update is not conducive to dynamic update requirements. Although manual operations could provide certain flexibility, their impact is limited. Hence, the efficiency of static update is questionable due the increasing possibility of inaccurate results. Increasing the frequency of update is not an ideal solution. If the update is conducted too often, it causes unnecessary overhead to the system since there is not new pattern to be added. In contrast, it may be still not fast enough to handle a burst of signature update. A more flexible and intelligent update scheme is expected to overcome these drawbacks. Consequently, events driven dynamic update is desired.

As illustrated in previous sections, we have relieved the pressure on storage space requirements by decomposing the signature into primary patterns. When a new signature has been caught, the update operation is correspondingly decomposed into two sub-tasks: i) adding new entry into the primary pattern set if new primary pattern appears; and ii) storing the relationships among primary patterns into the relation database. Since new signature patterns do not necessarily lead to the increase of primary patterns, particularly when the set of primary patterns become “large”. A new signature more likely implies a new relation among existing primary patterns.

Unfortunately, it is non-trivial to describe the complex association relations concisely and completely. We are considering simplify the associated relations through graph clustering algorithms, which cluster nodes according to specific sharing features [11]. Hierarchical Dirichlet Process with Hidden Markov State [16] is a possible solution for further processing of these relations.

It is also a critical mission to develop a non-disruptive updating strategy, since the normal operation of an intrusion detection system should not be interrupted for signature updating. A delayed write strategy could be an option. While a new primary pattern is detected, the system only updates the record of the bloom filter [5]. The pattern will be stored temporarily in a buffer, and written into the database later when the system is not so busy.

5.2 Dynamic Signature Matching

One advantage of our dynamic matching scheme is that we do not need to conduct the “matching” operation bit-by-bit as the current technologies. Once the incoming pattern has been decomposed, the existence of the primary patterns is detected through the Bloom Filters in parallel. Indexing by the Bloom Filters, a set of “links” will be identified, which is corresponding to association relationships among those primary patterns.

Treating the Snort signature database as a complex graph, we can re-model the problem as a path-finding problem. A signature matching is finding an existing directional path going through numbers of nodes in the graph.

Based on the above principle, we are designing a two-phase fast parallel dynamic signature matching scheme that takes advantage of the decomposition operation. Similar to the dynamic scalable pattern update operation, first phase will decompose the investigating patterns into primary patterns. Then the Bloom Filters [5] checks whether these patterns match the signature primaries in the Snort signature sets. If the result indicates there are multiple attack/intrusion primary patterns, we will try to walk though those nodes in the graph to detect the existence of a path.

6. Conclusion

In this paper, we proposed a novel two-step pattern decomposition scheme to remove hidden redundancies in the Snort signature database. Signature patterns are decomposed into primary patterns that can be stored along with their association relationships. Our approach has been validated through detailed analysis of multiple versions of the Snort signature database. In particular, our results suggest that increases in the number of signature patterns do not necessarily lead to commensurate increases in the number of

primary patterns. This technique relieves the resource demands on hardware implementations. In addition, it is promising towards a self-adaptive network infrastructure.

Currently we are extending our work to a real hardware application – a scalable dynamic pattern update and pattern matching for real time distributed intrusion detection application. Section 5 has presented the rationale of our ongoing efforts. One of the major challenges is how to handle the complex association relationship in a concise but accurate manner. Successful solution of this problem would not only benefit signature pattern based network intrusion detection, it will also benefit general complex pattern matching/updating applications.

Reference

[1] M. Aldwairi, T. Conte, and P. Franzon, "Configurable string matching hardware for speeding up intrusion detection," *SIGARCH Comput. Archit. News 33, 1,* Mar. 2005.

[2] M. Attig and J. W. Lockwood, "SIFT: Snort Intrusion Filter for TCP," *13th Annual Proceedings of Hot Interconnects (HotI-13)*, Stanford, CA, August 17-19, 2005.

[3] Z. Baker and V. Prasanna, "A Methodology for the Synthesis of Efficient Intrusion Detection Systems on FPGAs," *In Proceedings of the Twelfth Annual IEEE Symposium on Field Programmable Custom Computing Machines 2004 (FCCM '04)*, 2004

[4] J. C. Bispo, I. Sourdis, J. M. Cardoso, and S. Vassiliadis, "Regular Expression Matching for Reconfigurable Packet Inspection," *in Proc. IEEE Int'l Conference on Field Programmable Technology (FPT'06)*, Bangkok, Thailand, Dec. 13-15, pp. 119–126, 2006.

[5] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," *Internet Mathematics* **1** (4): 485–501, 2005.

[6] Y. H. Cho and W.H. Mangione-Smith, "Programmable hardware for deep packet filtering on a large signature set," *Workshop on Architectural Support for Security and Anti-Virus*, 2004

[7] C. R. Clark and D. E. Schimmel, "Scalable Parallel Pattern-Matching on High-Speed Networks," *in IEEE Symposium on Field-Programmable Custom Computing Machines*, 2004.

[8] S. Dharmapurikar, P. Krishnamurthy, T.S. Sproull, and J.W. Lockwood, "Deep Packet Inspection using Parallel Bloom Filters," *IEEE Micro*, Vol. 24, No. 1, Jan 2004, pp. 52-61.

[9] R. Franklin , D. Carver, B. L. Hutchings , "Assisting Network Intrusion Detection with Reconfigurable Hardware," *Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, p.111, September 22-24, 2002.

[10] B. Kernighan and S. Lin. *An Efficient Heuristic Procedure for Partitioning Graphs*, 1970. Bell System Tech.

[11] B. Long, Z. Zhang, P. S. Yu and T. B. Xu, "Clustering on Complex Graphs," *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, AAAI 2008, Chicago, Illinois, USA, July 13-17,2008.

[12] E. P. Markatos, S. Antonatos, M. Polychronakis, and K. G. Anagnostakis, "Exclusion-based signature matching for intrusion detection," in *IASTED International Conference on Communication and Computer Network (CCN'02)*, 2002.

[13] H. Song, T. Sproull, M. Attig, and J. Lockwood, "Snort offloader: A reconfigurable hardware NIDS filter," *In Proceedings of 15th International Conference on Field Programmable Logic and Applications (FPL)*, Tampere, Finland, Aug. 2005.

[14] I. Sourdis, J. Bispo, J. M. Cardoso, and S. Vassiliadis, "Regular Expression Matching in Reconfigurable Hardware," *J. Signal Process. Syst. 51, 1 (Apr. 2008)*, 99-121, 2008

[15] D. C. Suresh, Z. Guo, B. Buyukkurt and W.A. Najjar, "Automatic compilation framework for Bloom filter based intrusion detection," *Int. Workshop On Applied Reconfigurable Computing (ARC 2006)* Delft, The Netherlands, March 1-3, 2006.

[16] T. Xu, Z. Zhang, P. Yu, and B. Long, "Evolutionary Clustering by Hierarchical Dirichlet Process with Hidden Markov State," IEEE International Conference on Data Mining (ICMD 2008), Pisa, Itali, Dec. 15 – 19, 2008.

[17] Snort Team, "Snort Users Manual 2.8.3," http://www.snort.org/docs/snort_manual/, 2008

[18] H. Chen, D. H. Summerville, and Y. Chen, "Two-Stage Decomposition of IDS Rules towards Efficient Hardware Implementation," *Technical Report*, Dept. of Electrical & Computer Engineering, SUNY - Binghamton, Jan. 2009.