MRSI: A Fast Pattern Matching Algorithm for Anti-virus Applications

Xin Zhou[1, 2], Bo Xu[1, 2], Yaxuan Qi[1, 3] and Jun Li[1, 3]
[1]Research Institute of Information Technology, Tsinghua University, Beijing, China
[2]Department of Automation, Tsinghua University, Beijing, China
[3]Tsinghua National Lab for Information Science and Technology, Beijing, China
x-zhou05@mails.tsinghua.edu.cn

ABSTRACT
Anti-virus applications play an important role in today's Internet communication security. Virus scanning is usually performed on email, web and file transfer traffic flows at intranet security gateways. The performance of popular anti-virus applications relies on the pattern matching algorithms implemented in these security devices. The growth of network bandwidth and the increase of virus signatures call for high speed and scalable pattern matching algorithms. Motivated by several observations of a real-life virus signature database from Clam-AV, a popular anti-virus application, a fast pattern matching algorithm named MRSI is proposed in this paper. Compared to the current algorithm implemented in Clam-AV, MRSI achieved an 80%~100% faster virus scanning speed without excessive memory usages.

General Terms
Algorithms, Anti-Virus, Performance

Keywords
Pattern Matching, Virus Signatures

1. INTRODUCTION
Virus has become a major threat to today's Internet communications. Virus detection has become an essential part in today's network communication security. Most Intranet security gateway solutions have integrated an anti-virus module, which performs virus scanning on email, web and file transfer traffic flows. As the network bandwidth and the number of virus signatures increases, there is a great demand for high speed and scalable pattern matching algorithms.

The importance of fast anti-virus solution includes:

- First, in terms of network traffic scanning, speed is essential for it not to become bottleneck. Hence, most integrated intranet security gateways also play roles as firewall and NIDS (network intrusion detection system). The performance of anti-virus module could have great influence on the overall performance, given limited CPU resource and cache size.
- Second, the number of virus signatures keeps growing everyday, thus it requires the solution to be scalable in order to deal with more and more signatures.
- Third, the growth of network bandwidth also requires high speed pattern matching algorithm.

Clam-AV [4] is a well-known and widely used anti-virus solution on UNIX platforms. Clam-AV has implemented an extended version of BM (BMEXT) as a core pattern matching algorithm for scanning basic signatures. Although this algorithm adopts a series of heuristics to increase the scanning speed of the original BM [2] algorithm, BMEXT still performs poorly when handling tens of thousands signatures in some of the Clam-AV data sets.

Thus in this paper, we present a high performance pattern matching algorithm to further accelerate the scanning speed of anti-virus applications. The contribution of this paper can be described in three aspects.

- First, we analyzed the virus signatures extracted from Clam-AV and summarized several characteristics. We also analyzed the algorithms implemented in Clam-AV to match these signatures and pointed out there is room for improvement.
- Second, a fast pattern matching algorithm called MRSI was designed, according to observations of real virus signatures and anti-virus applications.
- Third, by implementing MRSI in the anti-virus system Clam-AV, an improvement on performance by a factor of 1.8 to 2 was obtained.

The following of this paper is organized as follows: Section 2 briefly reviews related works. Section 3 gives a deep analysis of the virus signatures and the current solution to the problem in Clam-AV. Section 4 shows the core ideas that motivated us to design MRSI and how it works. Section 5 holds all the performance evaluation results. Section 6 gives a summary of the whole paper and discuss about some future work.

978-0-7695-3106-9/08 $25.00 © 2008 IEEE
DOI 10.1109/ICN.2008.119

2. RELATED WORK

The related work of this paper includes various pattern matching algorithms and other approaches used to accelerate the scanning stage of anti-virus applications. The word "pattern" usually refers to the hexadecimal string in a virus signature.

Many pattern matching algorithms [3], [10], [11], [12] have been proposed to solve the problem of intrusion detection system (IDS). Most of them are shift based algorithms, which originates from a classic single pattern matching algorithm BM [2]. The core idea of BM is to utilize information from the pattern itself to quickly shift the text during searching to reduce number of compares as many as possible. BM introduces a bad character heuristic to effectively capture such information [2].

♦ **Bad character heuristic:** A mismatch occurs during character by character checking when the character 'α' in the text does match the character 'β' in the pattern. The text can be then shifted at least to align the mismatch character 'α' with the last occurrence of 'α' in the pattern or align the character 'α' to the end of the pattern if there is no 'α' in the pattern.

Hash-AV [6] introduced an approach to use cache-resident bloom filters to quickly determine most no-match cases. However bloom filters have false positives. In order to further check those positive cases, Hash-AV has to be used together with the original Clam-AV.

3. UNDERSTANDING ANTI-VIRUS

Clam-AV is a widely used open source (GPL) anti-virus application. It has been integrated into Unified Threat Management Systems (UTM), Secure Web Gateways and Secure Mail Gateways [5]. It provides an anti-virus engine in the form of a shared library. The Clam-AV team keeps maintaining the software and regularly updates the virus signature database.

3.1 Analyzing Virus Signatures

Currently, the total number of signatures in Clam-AV is about 150,000, and the number keeps increasing constantly. In this paper, we take the virus database downloaded from *http://www.clamav.net* on Aug 20th 2007 as a sample to analyze the general characteristics of a typical anti-virus signature database.

3.1.1 Three Major Types of Signatures in Clam-AV

Most of the signatures defined in Clam-AV can be categorized into three major types.

♦ **Basic Signatures**: Basic signatures are all hexadecimal strings. Clam-AV matches these signatures to the whole content of a file according to a sub category of files.

♦ **MD5 Signatures**: MD5 checksum for an executable virus file, MD5 for a white list executable file and MD5 checksum of a specific section in a Portable Executable file. Clam-AV matches signatures to a target file's MD5 checksum or the MD5 checksum of a specific section.

♦ **Regular Expression Signatures**: Actually an extended version of the basic signatures, which support several kinds of wildcards. Clam-AV matches these signatures to the whole content of a file. Although Clam-AV does not fully support standard regular expression, in this paper we refer to this type as regular expression signatures.

Except from these three major types, there are only a few other signatures for certain extended functions. There are 66 signatures for archive metadata and also 167 signatures for anti-phishing.

3.1.2 Identifying Performance Bottleneck

Table 1 shows the number of signatures for three major types and the others.

Table 1. Different Types of Signatures

	Basic	MD5	Regex	Other	Total
Num	78501	64758	4844	233	148270
%	52.9%	43.7%	3.3%	0.16%	100.00%

In order to observe the system's scanning overhead on different types of signatures, we injected several lines of codes to Clam-AV to calculate the time spent on matching each type of signatures in a typical scan. The experiment was performed on a 10 Mbytes randomly generated file and the result is shown in the following table.

Table 2. Scanning Overhead on Different Types of Signatures

	Basic	MD5	Regex	Total
Time (s)	6.200	0.054	2.190	8.444
%	73.4%	0.64%	25.9%	100%

According to the above experiment result, obviously matching the basic signatures consumes a large part of the scanning time. As a result, we focus our work of this paper on matching basic signatures to improve the virus scanning speed on Clam-AV.

3.1.3 Further Analyzing Basic Signatures

The basic signatures are further divided into 7 subsets according to file types, indexed from 0 to 6 for the convenience of analysis. The signatures in each subset apply to certain types of files. For example subset #0 applies to any file and subset #1 applies to portable executable files [4].

Table 3. Statistics of Basic Signatures

Idx	Total Number	Average Length	Min Length	Len<9 Num
0	29611	67.5	10	0
1	46954	123.7	4	8
2	164	106.8	28	0
3	1402	110.7	14	0
4	355	46.6	17	0
5	0	n/a	n/a	0
6	15	105.1	17	0

Table 3 shows several important statistical characteristics of the 7 subsets of basic signatures. We believe that it is crucial and possible to designing a better pattern matching algorithm according to the three observations.

From these analyses, we have three observations that motivated us to design novel algorithms:

♦ **Large scale signature set**: Even the scale of a virus signature subset is much bigger than a traditional IDS system. For example, the subset #1 has 46,954 signatures while Snort [8] only has about 3,000 signatures in total.

♦ **Longer average length**: The average signature length of these 7 sub categories are from 46 to 124, which are much longer compared to IDS signatures.

♦ **Very few short signatures**: Generally, signatures with length less than 9 characters can be viewed as short signatures. Thus, there are only 8 short signatures in subset #1 while there are no short signatures in other subsets.

3.2 Clam-AV's Solution and Limitations

The current version of Clam-AV used two pattern matching algorithms. The basic signatures are handled by an extended version of BM [2] (BMEXT) and the regular expression signatures are handled by a modified AC [1] algorithm.

Firstly, when implemented with DFA [9] data structure, AC consumes a large amount of memory for such large scale signature database. If implemented with NFA [9] data structure, there are several memory compressing techniques available to reduce memory consumption, however such techniques usually come with more memory access, which would hurt the performance badly.

Secondly, the BMEXT uses the last 3 characters of a signature to generate shifts. Given that the average length and shortest length of virus signatures are comparably long, we believe larger shifts could be produced by utilizing more characters of existing signature.

4. Multi-block Recursive Shift Indexing

This section will first give a brief introduction to the core idea of RSI which is the base of MRSI and then discuss more about the motivations behind MRSI.

4.1 Brief Introduction of RSI

RSI [3] is a high performance multi pattern matching algorithm designed specifically for IDS applications. The core idea of RSI is to use two levels of block heuristic to produce shifts.

♦ **Block Heuristic**: The original BM bad character heuristic uses 1 character to calculate shift value [2], while RSI uses 2 characters as a block heuristic.

♦ **Stage 1**: The first level uses two BLTs (Block Leap Table), with each BLT using a 2 characters block heuristic. In the first stage of scanning, each BLT returns a shift value. If they are not both zeros, the larger one can be used as a shift value.

♦ **Stage 2**: The second level introduces a 4 characters block heuristic. When both BLTs return zero in the first stage of scanning, then a Further Leap Table (FLT) is introduced in the second stage of scanning. The FLT is an intersection of zero entries of both BLTs in the first level, thus it is actually a 4 characters block heuristic and the size of the FLT is limited to the square of number of patterns.

♦ **Stage 3**: If the FLT in the second stage of scanning still returns zero, a Potential Match Table (PMT) is used and each entry of PMT links all the potential match patterns with the same first 4 characters. Thus the size of PMT is limited to the number of patterns.

4.2 The Problem of Adapting RSI

There are two major problems if we directly implement RSI directly in Clam-AV.

- First, the size of FLT is n^2 in the worst case, where n is the number of patterns. Given that the largest virus signature subset has about 47,000 patterns, the size of FLT could be 47000^2*2 bytes, about 4 Gbytes in the worst case. It takes too much memory and the preprocessing of shift values for such a large FLT could be very time consuming.
- Second, the possibility of getting both zeros from the two BLTs in the first stage will increase a lot when the number of patterns grows from 1,000 to 47,000.

Thus, we need to seek novel ideas and heuristics to resolve these two problems.

4.3 MRSI Motivations and Solution

In section 3, we gave a thorough analysis of the virus signatures and discussed the problem of the current solutions in Clam-AV. Some of these observations lead to the motivations behind MRSI in this section.

4.3.1 Using 3 BLTs

Before we explain how MRSI works, we will first present another important observation in a deep analysis of the basic signatures, which motivated us to use 3 BLTs in MRSI.

Although directly adapting RSI is not practical, the idea of using a block heuristic would be helpful in our design of a new shift based algorithm. In order to understand the possibilities of getting a non-zero shift by using block heuristic, we did an analysis of the first 6 characters of all signatures by dividing them into three 2 characters blocks.

♦ **Assumption 1**: every character in the content of the target file is randomly generated by a uniform distribution function. ($P(c)=1/256$, where $c=0\sim255$)

A 2 characters block has 65536 different values, which should be also of uniform distribution given assumption 1. However not all of them appear in the corresponding

position of signatures. For example, if a value 'x' ($0 \le x \le 65535$) does not appear in the first 2 characters of all signatures, there should be a non-zero shift for this value 'x' when 'x' appears in the target file. Given assumption 1, we define the number of such values divided by 65536 to be the possibility of non-zero shifts for this block.

Then we computed the possibility of non-zero shifts for the first 3 blocks of all signatures (excluding signatures that are shorter than 6) with 2 characters in each block. Then we use P_1 to stand for the possibility of non-zero shifts at block 1, P_2 for block 2 and P_3 for Block 3.

Thus, if we only use block 1, the possibility of non-zero shifts equals P_1. If we use two blocks, either block can return a non-zero shift, so the possibility of non-zero shifts for using two blocks should be:

$$P_1 + (1 - P_1)P_2$$

Analogically, the possibility of non-zero shifts using 3 blocks should be:

$$P_1 + (1 - P_1)P_2 + (1 - P_1)(1 - P_2)P_3$$

By applying these calculations to the largest two subsets of basic signatures, we got the results shown in Table 4.

Table 4. Possibility of Non-zero Shifts

Idx	Total Num	1 BLT	2 BLTs	3 BLTs
0	29611	81.9%	91.6%	99.4%
1	46954	54.8%	79.6%	90.8%

Obviously, by using 3 blocks instead of 2 blocks or 1 block, there could be much more potential non-zero shifts. This motivates us to use 3 BLTs in MRSI.

4.3.2 Matching Short Signatures Separately

From our previous analysis of Clam-AV's basic signatures in section 1, we know that there are only 8 signatures that are shorter than 9 characters.

By further exploring these short signatures, we found that they are all defined with an offset. This means they should only be matched to a specific position of a file. For example, the shortest signature in sub category #1 is defined as "W32.Deadc0de:1:64:dec0adde", which means only when the pattern "dec0adde" is matched at offset 64. It indicates a virus called "W32.Deadc0de".

Thus these signatures can be quickly checked by a direct brute-force check. In our implementation of MRSI, we do this for all patterns that are shorter than 9 characters.

4.3.3 Indexing a Potential Match Table

According to our observation 3.3.1, using 6 characters (3 BLTs) instead of 4 characters (2 BLTs) could generate more shifts and bigger shifts, assuming randomly generated file. The problem is how to index the potential match signatures when all 3 BLT returns zero shifts.

The original RSI (2 Block Leap Tables) uses the zero entries of the FLT to index a PMT, which link to all the potential match signatures for each entry. The size of PMT is the product of the number of zero entries in BLT1 and the number of zero entries in BLT2. As a result, for the subset 1 of about 50,000 signatures, the size of PMT is estimated to be more than 1 Gbytes.

MRSI's solution to this problem is not using a FLT but to use zero entries in BLT1 to index PMT. In this way, the PMT's size can be limited to the number of zero entries in BLT1. The trade-off is that the number of potential match signatures in one PMT entry becomes bigger than using a FLT. However, in a typical application environment, most of the files do not contain any virus. The possibility of all 3 BLTs returning zero at the same time should be quite low in such cases, consequently the possibility of actually matching a PMT entry should be low.

4.3.4 MRSI Description

We incorporated all these three motivations into one algorithm, Multi-block Recursive Shift Indexing (MRSI). The MRSI algorithm can be described in two parts:

- **Preprocessing Stage**

In the preprocessing stage, the algorithm creates and initializes all the necessary data structures for the later scanning stage. First it creates 3 BLTs for the first 6 characters. Then a PMT is created and indexed sequentially by the zero entries in BLT1, and each entry of the PMT is linked to the signatures that could be a potential match at this position.

For Example, if BLT1[α] has a zero value, all the signatures with the same first two characters of value α should be linked to the corresponding entry of the PMT.

- **Scanning Stage**

The scanning stage is comparably quite simply once all the necessary data structures are created. There are two phases in the scanning stage.

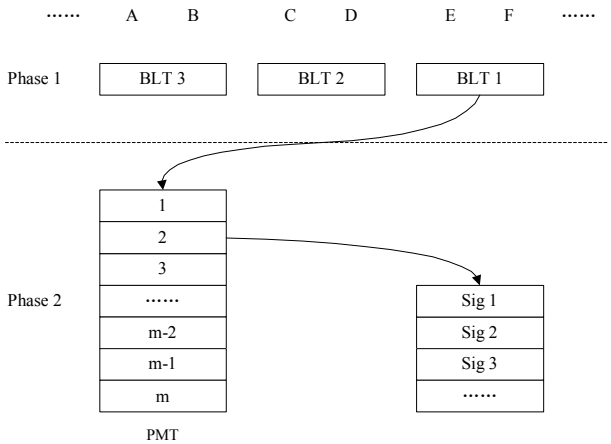

Figure 1. Data Structures and Two Phases of Scanning

In phase 1, 6 characters will be read from the target data, and then they are used as three keys to lookup the three

BLTs respectively. If three lookup results are not all zeros, then we get the largest one of the three shift values. The target data can be shifted by this value, and then phase 1 will be repeated at the new position of the target data.

If all three shift values are zeros, then the PMT table will be checked to match a list of potential match signatures. For each potential match signature, a brute-force check is used to validate if there is really a match.

The data structures and the two phases of scanning stage are shown in Figure 1. The string "ABCDEF" is used as a sample fraction in target data to demonstrate our algorithm in Figure 1. The PMT have *m* entries, where *m* is the number of zero entries in BLT1.

5. PERFORMANCE EVALUATION

In this section, several experiments were designed to give a full performance evaluation of our new algorithm. First a pure algorithm performance evaluation was performed to help understand the performance of MRSI under different circumstances compared to the current solution called BMEXT in Clam-AV. We then implemented MRSI in Clam-AV to see the actual performance improvement.

5.1 Algorithm Performance Evaluation

The experiments were all performed on an x86 Pentium 4 computer. The MRSI algorithm was written in C programming language so that it well supports cross platform applications.

All the Signatures were extracted from Clam-AV's signature database of Aug 20 2007. The largest two subsets of the basic signatures, subset 0 and subset 1 were selected for most of our experiments because the other subsets are comparably very small.

The target data was a 100 Mbytes randomly generated file using a uniform distribution function.

5.1.1 Performance on real Clam-AV Subsets

The experiments were performed with subset #0 of 29611 signatures and subset #1 of 46954 signatures. The memory consumption of all the data structures was calculated for both algorithms. The time cost in the scanning stage was recorded for both algorithms and then translated to the scanning speed in Mbps. The results are shown in Figure 1.

Figure 2 clearly shows that MRSI scanned the target data much faster than BMEXT on both subset #0 and subset #1. At the same time, Figure 3 shows that MRSI did not bring in excessive memory usages. This observation can be explained by the data structures of both algorithms.

In BMEXT, the BM shift table uses about 50~100 Kbytes while in MRSI 3 BLTs and 1 PMT uses 64*4 = 256 Kbytes at most. However both algorithms store a copy of all the

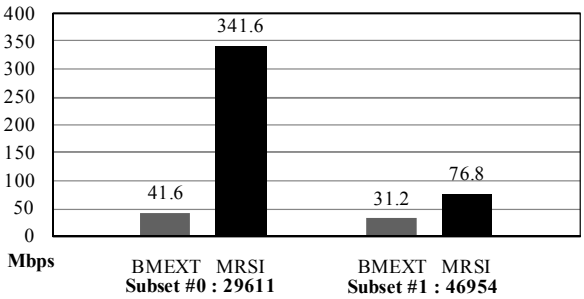

Figure 2. MRSI vs. BMEXT: Scanning Speed

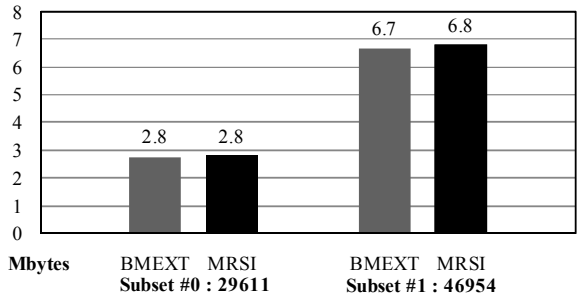

Figure 3. MRSI vs. BMEXT: Memory Usage

patterns in the memory and they cost the major part of memory in the above results.

5.1.2 Scalability Evaluation

In this experiment, we evaluated the scanning performance of both algorithms on different sizes of subsets to demonstrate the scalability of MRSI. We randomly selected signatures from all the basic signatures to form subsets with 1K, 5K, 10K, 15K, 20K, 25K, 30K, 40K, 50K, 60K and 70K signatures.

Figure 4 shows that both algorithms' performance dropped as the size of the pattern set grew. However MRSI constantly outperformed BMEXT with the size of a subset growing from 1,000 to 70,000 signatures.

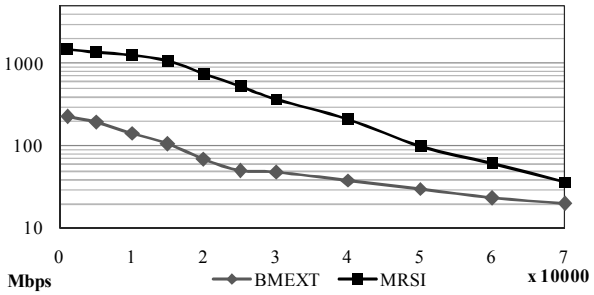

Figure 4. MRSI vs. BMEXT: Scalability

5.1.3 Performance under Attacks

In this test, a certain amount of patterns were injected to the random generated target data. This kind of target data could stand for potential attacking flows to the security gateways. Here we define a percentage of matches to measure how much of the target data actually matches a pattern. The

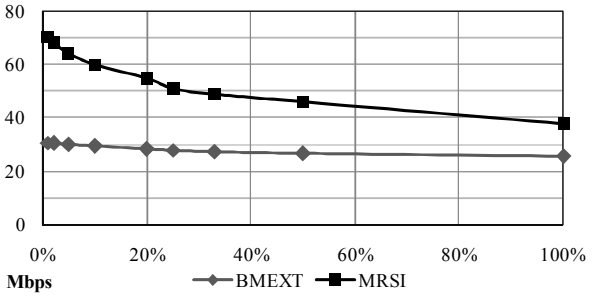

Figure 5. MRSI vs. BMEXT: Performance under Attacks

percentage of matches equals the sum of the lengths of all matched patterns divided by the length of the target data.

As the percentage of matches in the target data increased, MRSI's performance dropped, yet still outperformed BMEXT with 100% matches target data.

5.2 Clam-AV's Performance

The last test was designed to evaluate the performance of a real anti-virus system with MRSI implemented. First MRSI was implemented in Clam-AV to replace BMEXT. Two sets of data were used in the tests. One was a randomly generated file of 100 Mbytes and the other was a sample set of several executable files of about 80 Mbytes in total.

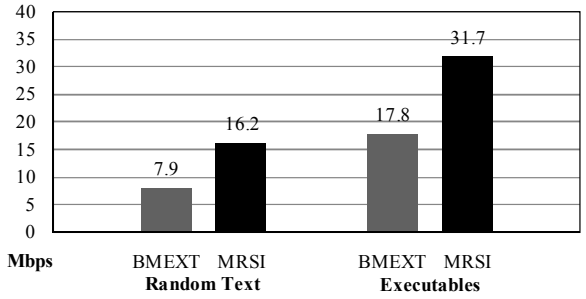

Figure 6. Real System Performance on Clam-AV

Figure 6 shows that for both sets of data, the performance of Clam-AV had been improved by a factor of 1.8-2 using MRSI.

6. CONCLUSION

Pattern matching algorithms have been a constantly explored field by many researchers in various studies due to its importance to network communication security. We have distinguished our work in this paper by focusing on anti-virus applications. Through a deep analysis of a typical virus signature database, we managed to summarize three keys observations, which enabled us to design a new fast pattern matching algorithm MRSI particularly for anti-virus applications. Various performance evaluation tests show that MRSI outperformed the current algorithm implemented in Clam-AV in many aspects.

Our future work contains two major parts. On one hand, we will go on seeking new observations and novel ideas to further improve the algorithm's performance. On the other hand, we plan to implement MRSI on new-generation multi-core network processors in order to obtain multi-Gbps virus scanning speed.

7. REFERENCES

[1] A. V. Aho and M. J. Corasick, Efficient string matching: An aid to bibliographic search, Communications of the ACM, 18(6):333–340, 1975.

[2] R. S. Boyer and J. S. Moore. A fast string searching algorithm, *C*ommunications of the ACM, 20(10), 1977.

[3] B. Xu, X. Zhou and J. Li, Recursive shift indexing: a fast multi-pattern string matching Algorithm, Proc. of the 4th International Conference on Applied Cryptography and Network Security (ACNS), 2006.

[4] T. Kojm, Clam-AV, In *http://www.clamav.net*, 2004.

[5] Sourcefire, Introduction to Clam-AV. http://www.sourcefire.com/products/clamav, 2007

[6] O. Erdogan and P. Cao, Hash-AV: fast virus signature scanning by cache-resident filters, Proc. Of the International Conference on Systems and Networks Communications (ICSNC), 2007

[7] F. Skulason, The evolution of polymorphic viruses, In *http://vx.netlux.org/lib/static/vdat/polyevol.htm*, 2004.

[8] M. Roesch, Snort: Network intrusion detection system, In http://www.snort.org, 2004.

[9] M. Norton, Optimizing Pattern Matching for Intrusion Detection, In http://docs.idsresearch.org/OptimizingPatternMatchingForIDS.pdf, 2004

[10] M. Fisk and G. Varghese, An analysis of fast string matching applied to content-based forwarding and intrusion detection, Technical Report CS2001-0670, University of California – San Diego, 2002.

[11] K. G. Anagnostakis, S. Antonatos, E. P. Markatos, and M. Polychronakis, E2xB: A domain-specific string matching algorithm for intrusion detection, Proc. of IFIP International Information Security Conference (SEC'03), 2003.

[12] S. Wu and U. Manber, A fast algorithm for multi-pattern searching, Technical Report TR-94-17, Department of Computer Science, University of Arizona, 1994.

[13] C. J. Coit, S. Staniford, and J. McAlerney, Towards faster pattern matching for intrusion detection, or exceeding the speed of snort, Proc. of the 2nd DARPA Information Survivability Conference and Exposition (DISCEX II), 2002.

[14] C. P. W. Y. Miretskiy, A. Das and E. Zadok. Avfs: An on-access anti-virus file system. Proc. of the 13th USENIX Security Symposium, 2004.