

國立臺灣大學電機資訊學院電機工程學系

碩士論文

Department of Electrical Engineering


College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

在網路入侵偵測上的可變步伐的樣式比對

Variable-Stride Pattern Matching
for Network Intrusion Detection



徐光民

Kuang-Min Hsu

指導教授：雷欽隆 博士

Advisor: Chin-Laung Lei, Ph.D.

中華民國 101 年 7 月

July 2012

致謝

論文的完成有賴於許多人的幫忙才能完成的，在不同的階段遇到了各種問題由不同人的幫助才得以克服各種難關，特別感謝他們的支援。

老師雷欽隆教授願意聽取在論文未完成前不成熟的想法，並從中指出問題以及給予建議。

張浩祥在提供了論文過程中的經驗，與浩祥每日的一起運動是維持寫論文期間規律生活最重要的因素，而在整個寫論文的期間不斷主動的關心而且全面的幫助，只有讓我由衷的感謝。

鄭君毅學長討論了如何使用 Libpcap，解決 detector 部分的的基本設計流程。

大學同學洪銘彥，有賴於平時常聊如何寫個品質較好的程式碼，在一開始時寫程式時就比較注意程式碼的寫作，才使的最後的程式部分得以正常運作。

Frank 學長在碩一時教導如何使用 latex，使論文寫作時可以很快的進入狀況。

Alan 學長的介紹下，才認識 Snort、Darpa98 data set，於是注意到了入侵偵測的領域。

吳蕙心學妹一起寫資料結構的作業時，精進了使用 lex 和 yacc 的技巧。

大學同學 BUN 多次解答了 Visual studio 的使用問題。

未曾見面的 Michela Becchi 教授，但是他發表的許多 pattern matching 論文中無論在論文內容、解決方法、和實驗的設計都是整個研究過程中重要的參考，而在論文中提出的內容也具有高度的可重製性，實作方面也幫助很大。

姐夫 Stuart 多次的審閱論文內容，修正許多英文上的句子，讓句子表達得更通順。

除了對於論文直接的幫助外，周圍的人的關心、鼓勵與督促也是這篇論文得以完成的必要條件，要特別感謝這方面父母、以及朋友們張浩祥、林男禹、洪銘彥、調酒社的朋友廖春宇、林月亮、吳蕙心陪伴過完寫論文的這段時光。

摘要

樣式比對是研究如何從文章中找出特定樣式的字串。在網路中，電腦之間的溝通可以視為雙方互相傳遞一些字串，所以樣式比對便可以用來偵測網路之間的溝通內容，而其中一種應用便是網路入侵偵測和預防。網路入侵偵測和預防是試著找出由外面網路進來的惡意封包，為了找到這些封包定義了一些關於惡意封包的規則，在偵測封包時會將這些規則轉為自動狀態機，而自動狀態機的效能通常決定了整個系統的效能。

可變步伐是一個基於 Winnowing 演算法的方法，這個方法被應用在字串辨識上能在保持和多步伐策略一樣的偵測速度下使用較少的記憶體。使用可變步伐策略下的自動狀態機每一次狀態轉換時都會一次處理不定數量的符號，而不是只有一個符號，如此減少了狀態轉換的次數而增加偵測速度，然而這個方法只被使用在字串辨識，這篇論文擴展可變步伐的策略到樣式比對，同時保持其原來的優點。

abstract

Pattern matching is a research topic that focuses on how to efficiently find strings of expected form in some text. In the network, the communication between the computers can be view as sending string to each other, so the knowledge of pattern matching is used to detect the content of communication in network. The network intrusion detection and prevention, one of application used pattern matching in the network, is try to find the malicious data from the incoming data stream which come from outside network. To find malicious data, the rules that present how malicious data look like are converted into automata. The performance of the automata always determines the performance of detecting system.

Variable-stride is base on Winnowing algorithm, and this scheme has more memory efficiency than multi-stride method when it has the same throughput improvement. Every transition in the automata applied variable stride may deal with a variable number of symbols, and reduce number of state transtion when detecting, so make detecting process faster. However, this scheme is only applied in string matching. Thus the goal of this dissertation is to extend variable-stride to NFA, and keeps its advantage at the same time.

Keywords: pattern matching, regular expression, automata, intrusion detection, multi-stride

Contents

| | | |
|----------|-----------------------------------|-----------|
| 1 | Introduction | 1 |
| 1.1 | Motivation | 2 |
| 1.2 | Contribution | 3 |
| 1.3 | Organization | 3 |
| 2 | Background | 4 |
| 2.1 | NFA and DFA | 4 |
| 2.1.1 | String Matching and NFA | 5 |
| 2.2 | Snort | 6 |
| 2.3 | PCRE | 7 |
| 3 | Related work | 10 |
| 3.1 | Hardware-Based Method | 11 |
| 3.1.1 | Software-Based Method | 11 |
| 3.1.2 | ASIC-Based Method | 11 |
| 3.1.3 | FPGA-Based Method | 12 |
| 3.1.4 | TCAM-Based Method | 12 |
| 3.2 | General Method | 12 |
| 3.2.1 | History FA | 12 |
| 3.2.2 | XFA | 13 |

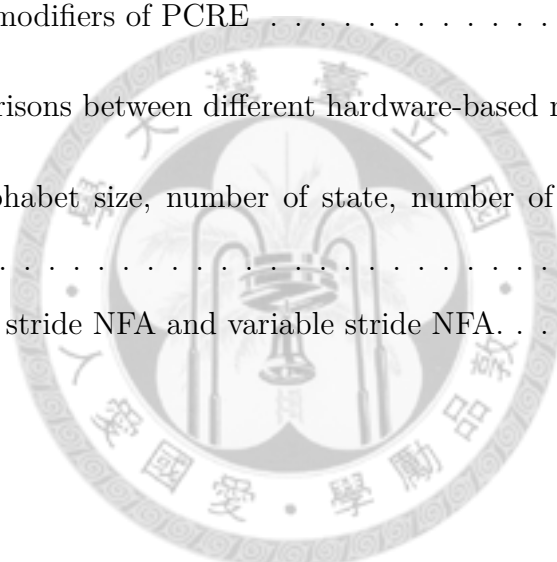
| | | |
|----------|---|-----------|
| 3.2.3 | Hybrid-FA | 13 |
| 3.3 | Compression Algorithm | 13 |
| 3.3.1 | Default Transition | 13 |
| 3.4 | Multi Stride | 14 |
| 3.4.1 | Winnowing Algorithm | 15 |
| 3.5 | Variable Stride DFA | 16 |
| 4 | The proposed scheme | 18 |
| 4.1 | Converting NFA to Variable-Stride | 18 |
| 4.2 | Considered Range | 19 |
| 4.3 | Basic Idea | 20 |
| 4.4 | Limit of a Pivot State | 21 |
| 4.5 | Choose a Processed State | 23 |
| 4.6 | Accepting State | 24 |
| 4.7 | Alphabet Reduction | 24 |
| 4.8 | Combine Single Symbol Blocks | 25 |
| 5 | Implementation and Result | 27 |
| 5.1 | Rule Set and Traffic | 27 |
| 5.2 | Implementation of NFA | 28 |
| 5.3 | Result | 29 |
| 6 | Conclusion | 32 |
| | Bibliography | 33 |

List of Figures

| | | |
|-----|--|----|
| 1.1 | network instruction detection architecture | 1 |
| 2.1 | A sample snort rule | 6 |
| 2.2 | The Format of PCRE in Snort | 7 |
| 3.1 | An normal NFA and its double stride NFA | 14 |
| 3.2 | Winnowing with $k=2$ and $=3$ | 16 |
| 4.1 | show the range to be considered to generate one block. | 19 |
| 4.2 | show the range to be considered for a pivot state | 20 |
| 4.3 | divide one of sequences of symbols into one block | 21 |
| 4.4 | The limit can verify the existence of the sequence and divide the sequence. | 22 |
| 4.5 | Alphabet reduction | 25 |
| 4.6 | Combine single symbol block into one block | 26 |
| 5.1 | An example shows that a regular expression is extracted from a rule. | 28 |
| 5.2 | shows the distribution of the number of outgoing transitions per state | 31 |

List of Tables

| | | |
|-----|---|----|
| 2.1 | Important operators of PCRE | 8 |
| 2.2 | Important modifiers of PCRE | 9 |
| 3.1 | The comparisons between different hardware-based methods | 11 |
| 5.1 | Average alphabet size, number of state, number of transition, and speedup | 29 |
| 5.2 | Mix double stride NFA and variable stride NFA. | 30 |



Chapter 1

Introduction

The network is the mainstream media for communication between computers today. However, the large amount of data exchange brings the high risk of malicious intrusion. Finding patterns, one of the oldest research topics, is developed to detect and prevent those malicious contents from the network. The requirement generated by the network for the computation of a very large amount of data in a very short time gives this old topic a new challenge. In addition to the needs of performance, the flexibility to describe malicious payloads is another requirement. This demands that we research not only string matching but pattern matching also.

There are many applications for network intrusion detection and prevention:

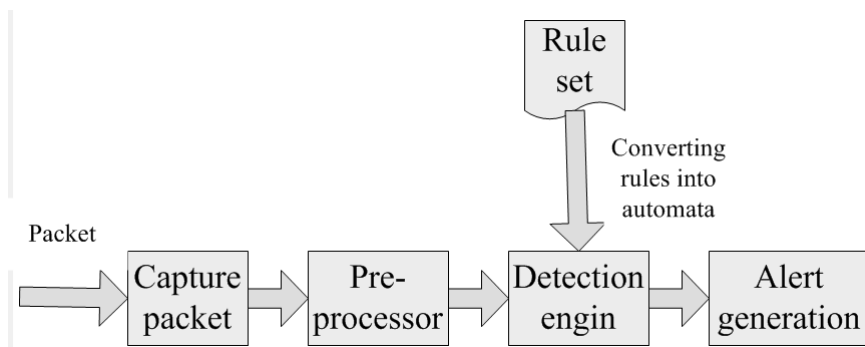


Figure 1.1: network instruction detection architecture

snort [1], Bro [2], and Cisco Adaptive Security Appliance [3]. They can be deployed in end devices or routers and may be for general payloads or special purposes like e-mail (ClamAV [4]). As shown in the figure 1.1, many signatures of malicious content are stored in a database, and each of them is represented as one regular expression. When the incoming network traffic arrives, it has to be checked according to all rules in the data base. If one segment of traffic matches one of the rules, then the corresponding pre-defined action is performed.

The rules are grouped and each group is converted into one machine. The machines are mostly deterministic finite automata (DFA) which have relatively higher performance than nondeterministic finite automata (NFA). However, DFA suffers memory explosion when its corresponding regular expression is complex. Thus, some researches focus on reducing the usage of memory [5, 6]. Other researches use NFA which is more memory efficient, and try to overcome its low performance [7].

1.1 Motivation

As the number of rules in the data base of NID increases, the needs of memory becomes a problem in building automata because it limits the number of rules. The high throughput of DFA makes it the most popular kind of automata, but it also suffers from memory explosion. Unlike DFA, NFA is more memory efficient, but its defect is low throughput. When NFA is chosen as the basis for implementation, some schemes which can improve throughput have to be applied. Scanning multiple bytes at each transition is one of techniques to increase throughput. [8] proposed this scheme called multi-stride and [9] extends it to NFA.

Paper[10] proposed a scheme which is based on Winnowing algorithm [11] and is called variable-stride, and this scheme has more memory efficiency than the multi-stride method when it has the same throughput improvement. However, this scheme

is only applied in string matching. Thus the goal of this dissertation is to extend variable-stride to NFA, and keep its advantage at the same time.

1.2 Contribution

The contribution of this dissertation is as follow:

- Winnowing algorithm is applied in pattern matching to convert NFA to variable stride NFA (VSNFA).
- A generating process that can decrease the side effect of VSNFA is provided.
- Variable stride NFA and normal NFA have the same structure, so the other scheme that proposed for normal NFA can also be applied in VSNFA.
- The process from rule sets to variable NFA are completely implemented and the variable NFA can be used to detect malicious content in traffic.

1.3 Organization

The remainder of the dissertation is as follow. In chapter 2, background on regular expression and automata is provided. In chapter 3, related works are introduced. In chapter 3, an algorithm is proposed to build a variable stride NFA. In chapter 4, the implementation and the result of experimentation are demonstrated. In chapter 5, the conclusion is summarized.

Chapter 2

Background

When the signature of malicious content is represented by regular expression, it often has to be converted into automata (also referred to as ‘machine’ in the remaining paper) for high speed detecting. When an automaton is stored in memory or implemented in ASIC, its storage cost is proportionate to the total number of transitions and states of automata, and its speed is determined by the number of transitions that it goes through from start state to accepting state, and the spending time of each transition. The performance of detecting processing is highly dependent on automata, so the research of pattern matching focuses on improving the performance of automata. The two basic automata, deterministic finite automata (DFA) and nondeterministic automata (NFA), are introduced as follows.

2.1 NFA and DFA

DFA and NFA can be represented as $\{Q, \Sigma, \delta, q_0, F\}$. Q is a finite set of states, Σ is a finite set of input symbols, δ is a transition function, q_0 is a start state in Q , and F , a finite set of accepting states, is a subset of Q . The only difference between DFA and NFA is δ . When given an input symbol and a state, δ in NFA returns a

set of states and δ in DFA returns a state.

NFA that can be generated from regular expression of length n by Thompson's algorithm [12] has $O(n)$ states. The subset construction [13] can convert NFA into DFA, and when NFA has n states, its corresponding DFA has $|\Sigma|^n$ states in the worst case. In addition, when some researches compile m rules of length n into an machine, the generated NFA has $O(mn)$ states or the generated DFA has $O(|\Sigma|^{mn})$ states. This means that NFA may be more memory efficient than DFA in theory, DFA has more states to represent the same regular expression than NFA.

The processing time of each input symbol is also different between DFA and NFA. The δ function of DFA returns only one state, so only one state is active during the whole process. Thus, each input symbol only needs to determine if the one next state is active; its time cost is $O(1)$. Instead, δ function of DFA returns a set of states which may include all state of NFA in the worst case, and each state has $O(n)$ outgoing transitions to all states. Thus each input symbol needs to determine if $O(n^2)$ states are active. Thus, the processing time of NFA is longer than the processing time of DFA.

2.1.1 String Matching and NFA

String matching, a special case of pattern matching, is focused on finding exact multiple strings, and also appears as an individual feature in NID. In string matching, the Aho-Corasick algorithm [14] is used to construct DFA. The rule set of string matching is relatively small and simple, so the generated DFA has fewer states and transitions and suffers no problems with memory explosion. However, the method to speed up string matching can be used in pattern matching based on NFA because the researches of NFA also focuses on increase throughput of NFA.

```
alert tcp any any -> 140.112.1.0/24 111:(pcre:"/BLAH/i"; msg:"hello world!");
```

Figure 2.1: A sample snort rule

2.2 Snort

Snort is an open source intrusion detection system (IDS). It is designed for monitoring small TCP/IP network and detecting malicious traffic from outside. Snort is called a lightweight IDS because it can easily be deployed on most of nodes in a network.

The signatures of vulnerabilities and malicious activities are represented as a set of rules; these rules are stored in a database called a ruleset. A simple language describes testing and action for each packet. Snort interacts with the TCP/IP stack and scans the packet payload directly by decoding the application of a packet to detect activities such as Buffer Overflow attacks, Denial of Service Attacks, Man in the Middle attacks, etc. Snort used only string based matching engines for deep payload inspection at the beginning. As the number of signatures increase, Snort uses regular expression based matching engines to represent signatures, because one regular expression signature can replace tens or hundreds of string signatures.

A snort rule consists of a header field and an option field. A header field includes action, protocol, source id address, source port, source MAC, destination port, destination source, and destination MAC; an option field includes many rule options which are separated by semicolon (;). The option field describes the detail of detection e.g. the specific content to be searched, the message of alerts, the identification of what part of Snort generates the event, and commands for plug-ins. Figure 2.1 is a sample snort rule. The left part is the header file, and the right part is the option field. The option field tells snort to print “hello world!” when the payload contains BLAH.

```
pcre:"/<regex>/[ismxAEGRUB]";
```

Figure 2.2: The Format of PCRE in Snort

Snort includes three subsystems: the packet decoder, the detector, and logging/logging system. These subsystems work on the top of libpcap [15] sniffer library which provides the ability to sniffer and filter the packets. The packet decoder has many subroutines, and each subroutine maps to one layer of protocol stack from the data-link layer to the application layer. Each routine imposes order on the packet data by overlaying data structure. The detector searches each rule recursively and searches each option of a rule recursively. When the first rule matches the packet, the defined action in rule is triggered and returns. The alerting/logging human-readable message and system log are generated which can be analyzed by other tools such as swatch.

2.3 PCRE

Perl compactable regular expression (PCRE) which has the same syntax and semantics as Perl5 is used in regular expressions based rules for Snort. Figure refPCRE show the format of PCRE field in Snort. A PCRE field consists of a regular expression and modifiers. Operators which are shown in Table 2.1 and symbols of the alphabet compose a regular expression. Modifiers set compile time flags for the regular expression. Table 2.2 shows important modifiers of PCRE.

The PCRE engine executes quantifiers in regular expressions to excess, so when many targets match a rule, the first and longest target is the final result. Each rule is individually compiled into NFA. Backtracking is used to trace NFA. In backtracking states are traced in deep first order, so only one path is traced at a time. If the path cannot reach the accepting state, then the tracing state is taken back to the last

Table 2.1: Important operators of PCRE

| Operator | Mean and Example |
|----------|---|
| | Alternation |
| | /aa ba/ matches aa or ba. |
| ^ | Match the beginning of the line |
| | /^ab/ matches abbb, but not babb. |
| \$ | Match the end of the line |
| | /ab\$/ matches bab, but not abb. |
| [] | Bracketed Character class |
| | /a[abc] matches aa,ab, and ac. |
| [^] | The class matches any character not in the list |
| | [^abc] matches any character except a,b, and c. |
| . | Match any character except newline |
| | /..a/ matches consecutive three characters whose ending is a. |
| * | Match 0 or more times |
| | /ba*/ matches b, ba, and baaa. |
| + | Match 1 or more times |
| | /ba+/ matches ba, baaa, but not b. |
| ? | Match 0 or 1 times |
| | /ba?/ matches b and ba. |
| {n} | Match exactly n times |
| | /a{3}/ matches aaa. |
| {n,} | Match at least n times |
| | /a{3,} matches aaa, aaaa, aaaaa, etc. |
| {n,m} | Match at least n but not more than m times |
| | a{2,3} matches aa and aaa, but neither a nor aaaa |

state that has another branch. When reaching the accepting state, then the trace is finished and the according action is performed. Thus Snort can only detect one matching rule even if many rules should be matched. For example, if the rules are /12*/ and /12*3/, and input string is 1223, the result is only 122, but neither 12 nor 1223.

Table 2.2: Important modifiers of PCRE

| Modifier | Mean |
|----------|---|
| m | The means of <code>^</code> and <code>\$</code> are changed from matching the begin and ending of string to match the begin of line and ending of any line. |
| s | The mean of <code>.</code> is change to match any character includes newline. |
| i | Do case-insensitive pattern matching. |

Chapter 3

Related work

The methods to improve the efficiency of pattern matching are classified into three different dimensions: hardware-based method, general method, and compression algorithm. When a method focuses on the implementation of automata, it is called hardware-based method, and it improves the performance of pattern matching by parallel processing. When a method, such as History FA [16], XFA [17], and Hybrid FA [18], focuses on the algorithm of automata, it is called general method, and improves the efficiency of pattern matching by extending NFA or DFA by changing the manner of state transitions; When a method, such as default transition [19], alphabet reduction [5], focuses on the data structure of automata, it is called compression algorithm, and reduces the usage of memory by compressing the states or transitions of automata which is NFA, DFA, or the extended automata; some different methods may be applied in the same system. For example, a system may have hybrid-FA and default transition, and is implemented in ASIC.

The following sections introduced the three dimensions of researches. Although multi-stride and variable-stride methods are classified in general method, they are introduced in individual section because these methods are the base of our methods.

Table 3.1: The comparisons between different hardware-based methods

| | ASIC | TCAM | FPGA | GPU | General purpose CPU |
|-------------|---------|--------|-------------|--------|---------------------|
| Cost | Highest | High | Medium | Low | Low |
| Flexibility | Worst | Medium | Medium | Good | Best |
| Design time | Longest | Medium | Medium | Short | Shortest |
| Performance | Highest | High | Medium High | Medium | Lowest |

3.1 Hardware-Based Method

Hardware-based methods improve the performance of pattern matching by parallel processing. Instead of combining rules, hardware-based methods prefer processing rules in parallel. There are four architectures: software-based, ASIC-based, FPGA-based, and TCAM-based. Table 3.1 shows comparisons between different architectures used for pattern matching.

3.1.1 Software-Based Method

Software-based approaches are implemented in general-purpose processors [18, 20, 6, 5, 21, 22] or general-purpose graphic processors [23, 24, 25], so these approaches are also called general-purpose approaches. Most software-based approaches are based on DFA, because DFA has high throughput to make up for the lower throughput of architectures of general-purpose processors than other hardware-based methods. However, the high throughput and restricted memory space of GPU make some researches [23, 26] use NFA-based approach in GPU architecture.

3.1.2 ASIC-Based Method

ASIC-based approaches design the specific chip only for execution of pattern matching. The ASIC based approaches have high throughput, but require a long development cycle, and, after producing, chips are hard to upgrade. The implemen-

tation cost of ASIC is very high, so much so that, as far as I know, no research implements the ASIC-based method, but some methods [8, 27] claim that their methods can be applied in both FPGA and ASIC architectures, and 3com and Cisco have their own ASIC production.

3.1.3 FPGA-Based Method

FPGA-based approaches [28, 29, 30, 31, 8, 27] have short matching cycle and support parallel matching operations. However, the fewer resources of FPGA restrict the number of rules. FPGA-based approaches are usually based NFA, because NFA has wide bandwidth requirement and low memory consumption.

3.1.4 TCAM-Based Method

Ternary Content Addressable Memory (TCAM) is a type of memory that can search stored strings in parallel and high speed. The size of TCAM is small (about 1MB), so TCAM-based approaches [32, 33, 34, 35] compress automata and the parts of data structure are stored in TCAM, and others are stored in normal memory.

3.2 General Method

3.2.1 History FA

History FA proposed in Paper [16] uses flags and counters to remember which parts of signatures have been dealt with so that the number of states are reduced. However, the side effect of additional variables is an increase in computation time per state, and, when the rule set becomes large and complex, the size of the additional data structure increases considerably.

3.2.2 XFA

XFA proposed in Paper [17] uses the same strategy as history-FA but uses dot-star terms to separate sub-patterns from regular expressions, because these sub-patterns are the cause of memory explosion. History-FA seeks only to reduce blowup, but XFA begins with a formal characterization of blowups, so some variables and instructions are attached to each state. Variables cannot affect state transition but can affect acceptance, so a string is accepted only when reaching one of accepting state and variables are specific values.

3.2.3 Hybrid-FA

Hybrid-FA proposed in Paper [18] combine the advantages of NFA and DFA. When converting NFA into DFA, if a state causes blowup, then the part of automata after this state stays in the form of NFA, so a hybrid-FA has a head DFA and multiple tail-NFA. These tail-NFA are converted into DFA. When processing, the head-DFA is always active, but each tail-DFA is active when the border state is traversed.

Hybrid-FA has three advantages as mentioned below. First, the average case the traverse of Hybrid-FA stays in the head-DFA. Second, the worst case is bound by the number of tail-DFAs. Third, the compression algorithms proposed for DFA can be applied in Hybrid-FA.

3.3 Compression Algorithm

3.3.1 Default Transition

Paper [19] observed that many states have similar or identical transitions so that compress transitions by replacing many similar transition tables with one transition table to reduce memory. During pattern matching, default transitions are followed

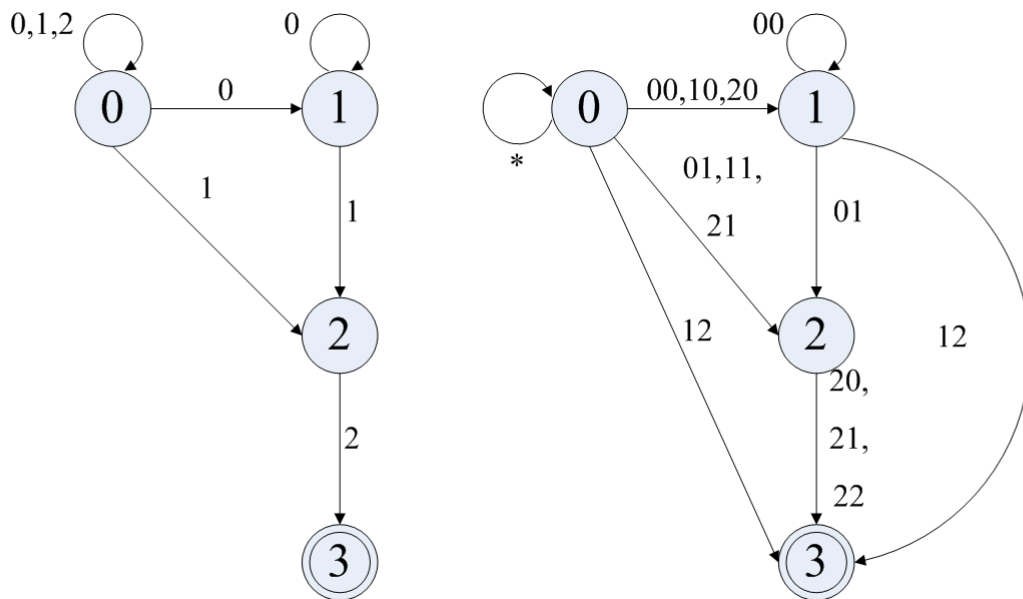


Figure 3.1: An normal NFA and its double stride NFA

from state to state until a compressed table entry is found that corresponds to the current input symbol and the number of default transitions is bounded. The disadvantage of default transition is that the matching time is longer.

3.4 Multi Stride

The n-stride method can improve NFA throughput n times by processing n symbol at the same time [9]. However, the target sequence may have n difference start bias. This results in generating n instances of NFA. An integrated NFA which the n instances combine to form is used to match the pattern. The integrated NFA costs many times the memory of original NFA.

Figure 3.1 shows an example of double stride. The normal NFA is on the left side of figure and its corresponding double stride NFA is on the right side of the figure. The alphabet has 0, 1 and 2. The state 0 is the start state and the transition to itself means that the accepting strings can start at every position in the input

string. For state 1, all possible strings of length 2 have to be tested in the normal NFA, and 00 reaches state 1, and 01 reaches state 2, and 12 reaches state 3, while other strings can't reach any state, so three transitions are created from state 1 to state 1, state 2 and state 3 in double stride NFA. Most transitions are created like this except the transition from state 2 to state 3. State 3 is an accepting state and 2 makes a transition from state 2 to state 3 in the normal NFA. When string goes through state 3, a string is accepted by this NFA, and the remaining input is ignored. Although the string 20, 21 and 22 can't reach state 3 from state 2, but they go through state 3, so they still make transitions from state 2 to state 3 in double stride NFA.

Paper [10] pointed out a problem where the target string which is accepted in normal NFA may divide into different blocks when its start position changes, and the number of kinds of division is equal to the number of stride. Each kind of division has a group of transitions to identify them, and different groups actually accept the same language. For example, the two input strings are 2122 and 1222, and 12 which appear in two strings are accepted by NFA in figure 3.1. The first string 2122 is divided into 21|22 and the second string is divided into 12|22. In double stride NFA, 21|22 would go to state 2 and then go to state 3, and 12|22 would go to state 3 and stay there. Although NFA detects the same part string 12 in two strings, it uses two different paths to trace it.

3.4.1 Winnowing Algorithm

The Winnowing algorithm is a local fingerprinting algorithm and it divides an input stream into many blocks. It guarantees that the individual block isn't altered by changes in another part of the stream [11]. This means that the division of the stream is only determined by the local symbol. Conversely, multi-stride has to decide

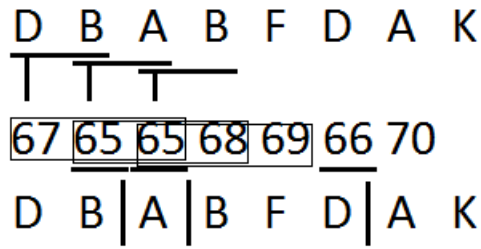


Figure 3.2: Winnowing with $k=2$ and $s=3$

the division globally.

The original algorithm is as mentioned below. A hash value is calculated by every s consecutive symbols. For a slide window of given size w , a minimal hash number is chosen in the window. A tie is broken in the right of the minimal hash value.

The segmentation from a right number of a minimal value to the next minimal value is called a block. The block size k is bounded by window size w and the expected block size is $(w + 1)/2$ which is also the expected speedup of detecting.

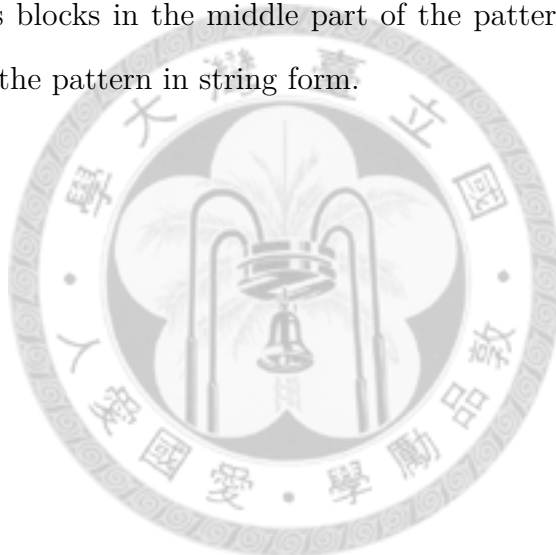
For our scheme, s is set to 1, because a window is calculated by $w+s-1$ input symbols, and when the number of considered symbols increases, the execution time of our algorithm rises exponentially. In addition, calculating a hash value for all symbols is done before a variable stride step, so a hash of a symbol is equal to its definition value. The process is shown in Figure 3.2.

3.5 Variable Stride DFA

To avoid the mass usage of memory to recognize the position bias of a target sequence, variable stride method was introduced in [10]. Variable stride method is based on Winnowing algorithm. Variable stride method uses Winnowing algorithm to divide the string pattern and input string into blocks, and then uses a block as a

symbol in normal DFA to form DFA by Aho-Corasick algorithm. In variable stride DFA, a block would make a state transition, so every state transition can deal with more than one symbol.

To apply Widdowings algorithm effectively, one problem has to be solved. Firstly, when string pattern divided into blocks, the block in the head or tail of the pattern can't be decided because it needs to know the part of the input string which is adjacent to the head or tail of the pattern. To solve this problem, Paper [10] only compares the middle part of the pattern in the block form, and, when part of the input string matches blocks in the middle part of the pattern, the string matches the head and tail of the pattern in string form.



Chapter 4

The proposed scheme

In this chapter, a scheme based on winnowing algorithm [11] and variable stride DFA [10] is proposed. The scheme tries to expand on this theory to apply to pattern matching. The basic idea would be propose, and, however, basic idea is a slow generating method, so the notation called limit is introduced to speedup up generating process.

4.1 Converting NFA to Variable-Stride

The rules in Network Intrusion Detection (NID) are represented by regular expression. To detect malicious patterns, the rules are transferred into NFA by the Thompson algorithm [12]. The method presented in [9] is used to eliminate ϵ and reduce NFA.

NFA without ϵ is chosen to be converted into variable stride form, because states and transitions are the only two factors to be considered. If regular expression is converted into variable stride form, each operator would need to be dealt with through a different strategy and the process would be more complicated.

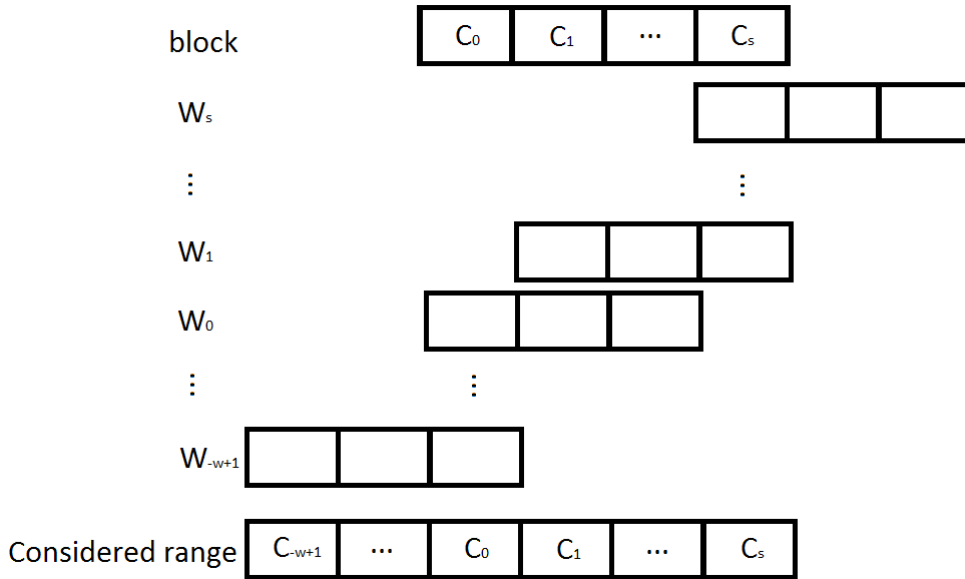


Figure 4.1: show the range to be considered to generate one block.

4.2 Considered Range

The window size is w and the hash value is calculated by every symbol in Winnowing algorithm. As Figure 4.1 shown, if a block is $C_1C_2\dots C_s$, and C_0 is a symbol before a block and divides the string such that ensure the beginning of block. $W_{-w+1}\dots W_0W_1\dots W_s$ are all windows that contain at least one of the symbols $C_0C_1\dots C_s$. At least one of the windows from W_{-w+1} to W_0 has C_0 as the minimal number so that the beginning of the block is C_1 . C_s is the last symbol of the block, so C_s must be the minimal number in one of windows from W_1 to W_s , but the minimal numbers in $W_2, W_3\dots W_s$ must be after the minimal number in W_1 or be the same number of W_1 , so $W_2, W_3\dots W_s$ can be ignored. The symbols which $W_{-w+1}\dots W_1$ contain are all symbols that affect one block.

In conclusion, for generating one block of size s , the w symbol before has to be considered and the $w-1$ symbols after the first symbol of the block are considered, so total $2w$ symbols are considered.

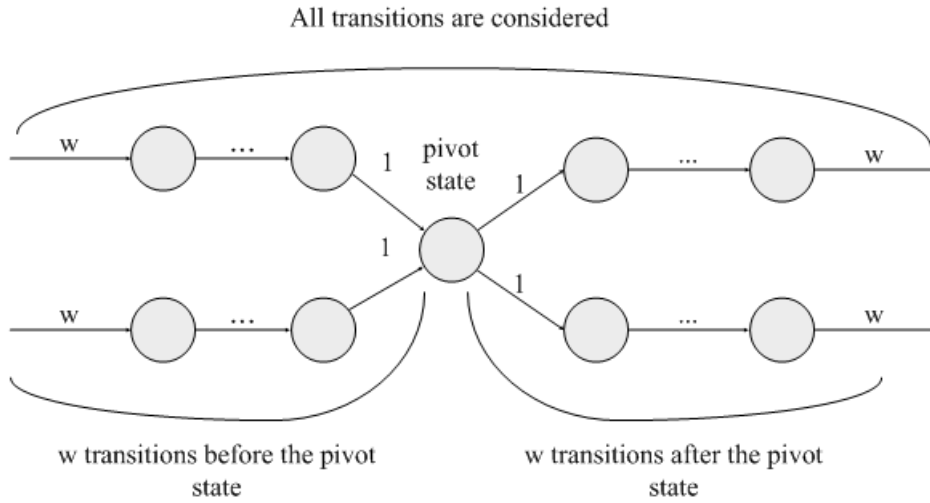


Figure 4.2: show the range to be considered for a pivot state

4.3 Basic Idea

For a given NFA without ϵ ($nfa = (Q, \Sigma, \delta, q_0, F)$) and window size w , a variable stride NFA $nfa' = (Q, \Sigma^w, \delta', q_0, F)$ is created after our scheme. Every state of nfa is considered individually as a pivot state. All accepting symbols that cause outgoing transitions in a pivot state are supposed to be the first symbol of a block. All possible sequences composed of w symbols that are both before and after the pivot state have to be considered to divide the sequence into a block $c_1c_2 \dots c_k$ by winnowing algorithm as shown in Fig 4.2. As figure 4.3 shown, if the generated block starts from pivot state q_1 , a transition whose value is the symbol sequence of block is added from q_1 to q_2 in variable stride NFA when the block reaches state q_2 in the original NFA. When all states are processed as above, nfa' is created.

Although this method can convert an NFA to a variable NFA, it spends $O(|\Sigma|^{2w})$ to consider each state.

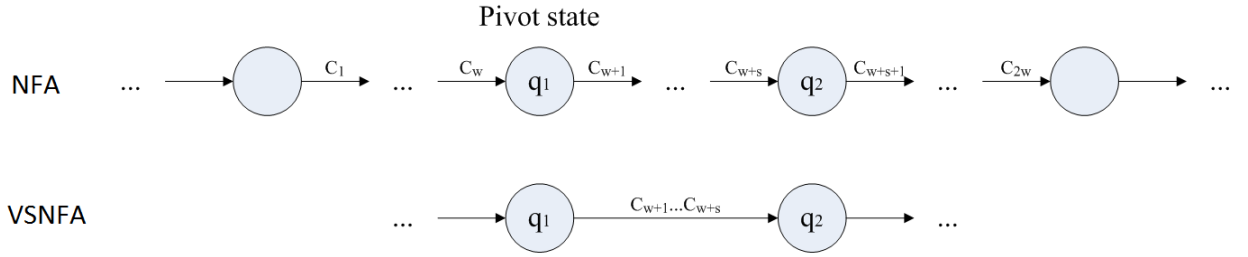


Figure 4.3: divide one of sequences of symbols into one block

4.4 Limit of a Pivot State

The relationship of two adjacent blocks has to be considered to speed up the converting process, because there is overlap between the considered ranges to generate two adjacent blocks. If the overlap is computed only one time, the process is executed faster. The goal is to keep some information when the first block is computed, and then use this information to speedup the generating process of the second block.

There are two adjacent blocks B_1 and B_2 , B_1 is $C_1 C_2 \dots C_{k_1}$, and B_2 is $C_{k_1+1} C_{k_1+2} \dots C_{k_1+k_2}$. By Winnowing algorithm, C_{k_1} must be the smallest number of $C_1 C_w$, so the first $w-k_1$ symbols in B_2 have to be larger than C_{k_1} . C_{k_1} is called the limit of B_2 and $w-k_1$ called the effective range of limit for B_2 . The above property can be described as below:

- Property1: if symbols of block in effective range are smaller, than the block does not exist.

. The limit is the smallest value in the window, so when the symbol in B_2 which is out of effective range, and is the first number that is smaller than the limit, then this symbol must be the smallest in the window where the symbol is the last symbol. The last symbol of B_2 should be this symbol.

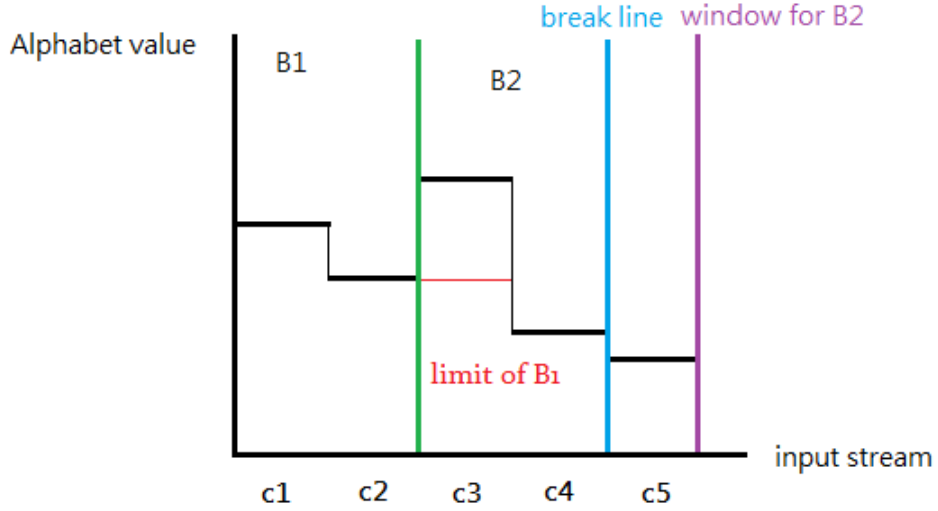


Figure 4.4: The limit can verify the existence of the sequence and divide the sequence.

- Property2: if a symbol of a block is out of effective range and the value of the symbol is smaller than the limit, this symbol is the last symbol of the block.

When the limit of a block and its range are known and Property1 and Property2 are used, the w transitions before the pivot state are replaced by the limit. The combination of one of the limits and w transitions after the pivot state generate a block by the following three steps. Firstly, Property1 is used to verify the existence of w transition. Secondly, Property2 is used to divide w transitions after the pivot state. Thirdly, if Property2 can't generate a block, the transition whose value is smallest is the end of the block.

The figure 4.4 shows an example of the limit. When $w=3$ and B_1 is c_1c_2 , the limit generated by B_1 is c_2 and the effective range is $w-2$. C_4 in B_2 is out of effective range and smaller than the limit. Although c_5 in the window for B_2 is smaller than c_5 , the tie is broken by c_4 .

For a pivot state, every incoming transition will bring a limit and an effective range corresponding to this limit. When all incoming transitions of a pivot state are calculated, the limit and the effective range, the w transitions after the pivot state can only be considered how to be divided into a block, and let the limits and their effective range verify the existence of the block, so the w transitions don't need to be considered again. If there is one limit and all symbols of the block within effective range are larger than or equal to the limit, the block exists.

4.5 Choose a Processed State

Although the notation of the limit can reduce the converting time of each state, the notation works well only when the order of the pivot state is proper. A state is best evaluated after all preceding states have been considered. However, when a state transition diagram has cycles, some states can't be considered after all preceding states have been evaluated. There are two types of cycle: self loops and large loops.

A self loop is a transition the destination and target of which are the same state. A self loop is converted from a symbol, wildcard or bracket expression with a Kleene star $ex a^*$. A large loop is a path for which the destination and target are the same state but contain more than one transition. A large loop is converted from the expression with a Kleene star $ex (aba)^*$. When a state transition diagram has a large loop, a state in the large loop has to be chosen to verify all incoming transitions of that state and the loop is broken. Fortunately, there is no rule including a large loop in the present snort rule set.

In conclusion, a method for choosing a state based on breadth first search (BFS) is to decide the order of visiting states. Unlike original BFS, a state is not added to the queue until all its incoming transitions (except self loop) are traversed. When

the queue is empty and some states are still not visited, there is at least one cycle. A non-visited state which has the most visited incoming transitions would be chosen to calculate its limits from all incoming transitions. After calculating limits, the state is added to the queue and the cycle is broken.

4.6 Accepting State

To ensure that the variable stride NFA and its original NFA accept the same language, the mapping between the accepting states of two NFAs have to be monitored like [9].

A state s_0 has transitions B which is $c_1 c_2 \dots c_k$, and goes through a set of accepting states S_a . Although when input is $c_1 c_2 \dots c_k$, s does not arrive at state of S_a , a transition B is also created from s_0 to all states in S_a .

4.7 Alphabet Reduction

Alphabet-reduction presented in [5] reduces the alphabet size of a machine and thus decreases the number of transitions of the machine. The different symbols in a machine may have the same behavior, so the symbols can be classified by the test *if $\delta(q, c_i) = \delta(q, c_j), \forall \text{state } q \text{ and } c_i, c_j \in \Sigma$, then c_i, c_j belong to the same class*. After classifying, the index of class can be used as the symbol replacing its members. The new size of symbols is equal to the number of class.

The alphabet-reduction can also be used for variable stride NFA because it is a normal NFA made by viewing a block as a symbol. When the window size is w and the symbol set size of an original NFA is $|\Sigma|$, the block set size of the variable stride NFA is $|\Sigma|^w$, but most symbol combinations never appear or appear only for wildcard symbols. Thus alphabet-reduction is necessary for variable-stride NFA to

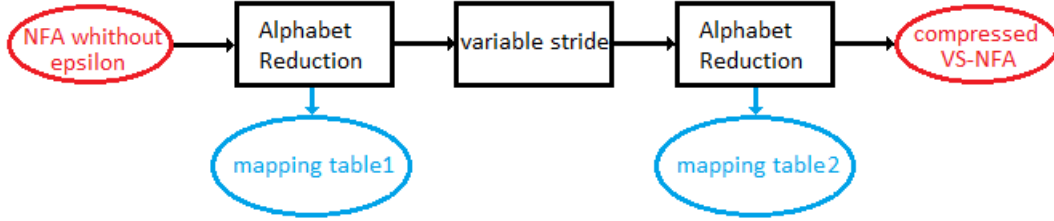


Figure 4.5: Alphabet reduction

remove these redundant transitions.

As shown in Figure 4.5 Alphabet-reduction is used for NFA without ϵ and variable stride NFA and generates two mapping tables. Mapping table1 maps the Σ into a smaller Σ' and mapping table2 maps the σ^w into σ'' . The two tables combine into a single a mapping table and the input stream is translated according to this table.

4.8 Combine Single Symbol Blocks

The expected block size determines the factor of speedup, so when the consecutive single symbol blocks combine into one block, the factor of speedup rises. Paper [10] proposed the above idea and a scheme for string matching. Combination Rule 1 applied to the input data stream can be used directly here, but Combination Rule 1 applied to patterns has to be extended to pattern matching.

There are possible multiple following blocks of one block in a variable NFA, and each single symbol block tries to combine with its following single symbol block. However, when one of following blocks is multi-symbol, the preceding block stays in the state machine.

Figure 4.6 shows an example how combined single symbol block, and the maximum size of combined block is 3. C_1, C_2 , and C_3 are single symbol block in the original variable stride NFA. The process of combination starts at C_1 , and then C_1

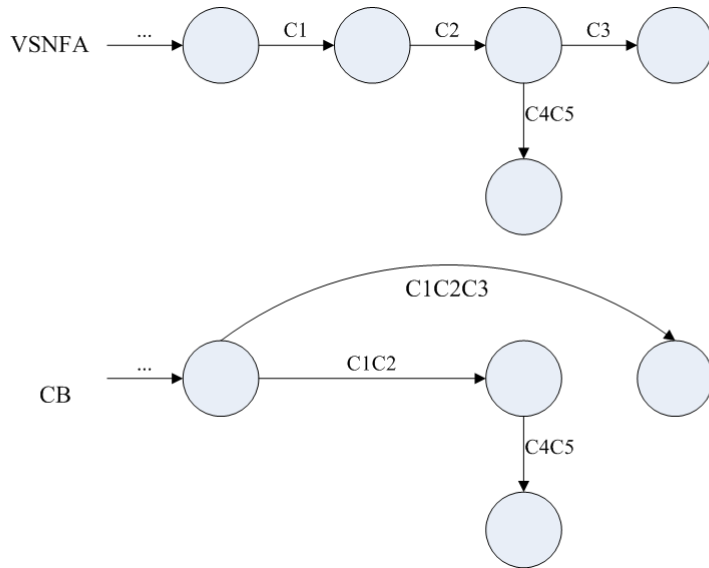


Figure 4.6: Combine single symbol block into one block

and C_2 combine to form C_1C_2 which is from q_1 to q_3 in variable stride NFA, because C_1 and C_2 are single symbol block. C_1C_2 does not reach the maximum size of block, so C_1C_2 and C_3 can still combine together. However, block C_4C_5 is from q_3 to q_5 , so C_1C_2 has to stay in automata after combining block so that the input stream can still reach q_5 from q_1 . The result of the combination is shown at the bottom of Figure 4.6.

For pattern matching, the pattern must start at the beginning of the file so that the ambiguity at the beginning of the pattern does not exist, but the ambiguity at the end of the pattern still happens.

Chapter 5

Implementation and Result

5.1 Rule Set and Traffic

The rules in experiment are from Snort [1], and are released in 2011/06/02, and are represented as Perl Compactable Regular Expression. The rules which match the header $\{tcp, \$HOME_NET, any, \$EXTERNAL_NET, any\}$ is found, and the regular expressions after the keyword “pre” in these rules are extracted. There are 1395 rules which include regular expression in the rule set, and the number of the extracted rules is 209; the rules converted into NFA are chosen from these rules randomly. The extracted rules include most features defined in PCRE, except the Unicode and back reference. The chosen traffic is from the darpa98.

One of the rules of matching requirement is shown at the top of Figure 5.1 . This rule is a tcp[The regular expression is the only part used in the experiment, so the regular expression is extracted at the bottom of Figure 5.1.

```
alert tcp $HOME_NET any -> $EXTERNAL_NET 25 (msg:"BACKDOOR dkangel
runtime detection - smtp"; flow:to_server,established; content:"Subject|
3A|"; nocase; content:"|BA DA B0 B5 CC EC CA B9| 2.41 "; distance:0;
nocase; pcre:"/^Subject\x3A[^\r\n]*2\x2E41/smi";
flowbits:set,DKangel_Email; flowbits:noalert; metadata:policy security-ips
drop; reference:url,www3.ca.com/securityadvisor/pest/pest.aspx?id=
453076278; classtype:trojan-activity; sid:6125; rev:3;)
```



```
pcre:"/^Subject\x3A[^\r\n]*2\x2E41/smi"
```

Figure 5.1: An example shows that a regular expression is extracted from a rule.

5.2 Implementation of NFA

Regular expressions are parsed by a self-made parser generated by Flex and Bison. Some features of Perl regular expression about back reference are not implemented, because these functions are not used in the chosen regular expressions. Each rule is converted into NFA with ϵ individually and rules which are in the same group connect to the same start point of NFA that is suggested in [9] to avoid state reputation for the anchored pattern with m-modifier. ϵ in NFA can be removed by the algorithm in [9] and this algorithm can reduced the NFA state and transition. However, there are some states that cannot reach any accepting state in the NFA after reducing, so an algorithm to remove these redundant states is added after reducing NFA. These NFAs are converted into variable stride NFAs and double stride NFAs which are used to compare with variable NFAs.

After generating variable-stride NFA, this machine can be used to scan the input stream by the detector. The detector uses Libpcap [15] to capture the packet from tcpdump file. Unlike Snort, the packet decoder is not implemented, so the detector can only detect the payload of a packet. The payload also has to be divided by Winnowing algorithm (with or without combination rule) as mentioned before, and

Table 5.1: Average alphabet size, number of state, number of transition, and speedup

| #RE per NFA | 1 | | | | 3 | | | |
|-------------|------|-------|-------|-------|------|-------|-------|-------|
| type | NFA | 2-NFA | VSNFA | BC | NFA | 2-NFA | VSNFA | BC |
| $ \Sigma $ | 17 | 67 | 41 | 81 | 31 | 202 | 230 | 360 |
| #state | 1102 | 1102 | 825 | 773 | 996 | 996 | 714 | 685 |
| #tx | 4590 | 10392 | 10734 | 15067 | 4725 | 15688 | 21534 | 29458 |
| Speedup | 1 | 2.00 | 1.31 | 3.25 | 1 | 1.98 | 1.41 | 3.56 |

two mapping tables generated by alphabet reduction are also applied to the input stream.

5.3 Result

Table 5.1 shows the comparisons between normal NFA, double stride NFA, variable stride NFA and variable stride NFA with single symbol block combination. The window size in variable stride method is 3. The speedup of normal variable stride is smaller than double stride and the speedup of variable stride with block combination is larger than double stride. The speedup of variable stride is determined by the average block length. The theoretical value of the average block length is 2, but its real value is about only 1.45, so the speedup of variable stride is smaller than the predicted value. The speedup of variable stride with block combination is significantly larger than normal variable stride, because there are many single symbol blocks in the normal variable stride NFA, and when blocks are combined, the average length of a block becomes larger. This proves that block combination scheme can work well.

The usage of memory of normal variable stride is more than double stride because the variable stride NFA needs more transitions to present regular expression $.^*$ and $[\wedge]^*$; this cancels out the self synchronized benefit of variable stride. When blocks are combined, new blocks which have never been used in normal variable stride appear

Table 5.2: Mix double stride NFA and variable stride NFA.

| | DSNFA | VSNFA | Total |
|--------|-------|-------|-------|
| #rules | 12 | 18 | 30 |
| #state | 501 | 435 | 936 |
| #tx | 6750 | 4108 | 10858 |

in variable stride with block combination, so the size of the alphabet increases, and variable stride NFA needs more transitions to present regular expressions $.^*$ and $[^]^*$.

When one regular expression is converted into one NFA, not all of the variable stride NFAs need more memory than double stride. Some variable stride NFAs only need 60%-75% memory space relative to double stride NFAs. That suggests that some rules which use less memory in variable stride method can be converted into variable stride NFA and other rules can be converted into double stride NFA. It is easy to keep two kinds of NFA in the same system because the difference between variable stride method and double stride method is the division of input stream except generating process. Another good side effect is that variable stride NFA has better throughput when those variable stride NFAs are chosen. Table 5.2 shows the result of mixing the two kind of NFAs. The number of rules is 30 ,and the rules are the same as Table 5.1. The result of this mix can reduce 5% memory space relative to original double stride NFA.

Figure 5.2 shows the distribution of the number of outgoing transitions per state. The states without transition are accepting states. For both kinds of VSNFA, the 90% states have fewer than 16 outgoing transitions and 60% states have fewer than 7 transitions. This suggests that variable stride NFA can be properly encoded to distinguish between the state with fewer transitions and the state with more transitions and make NFA more efficient.

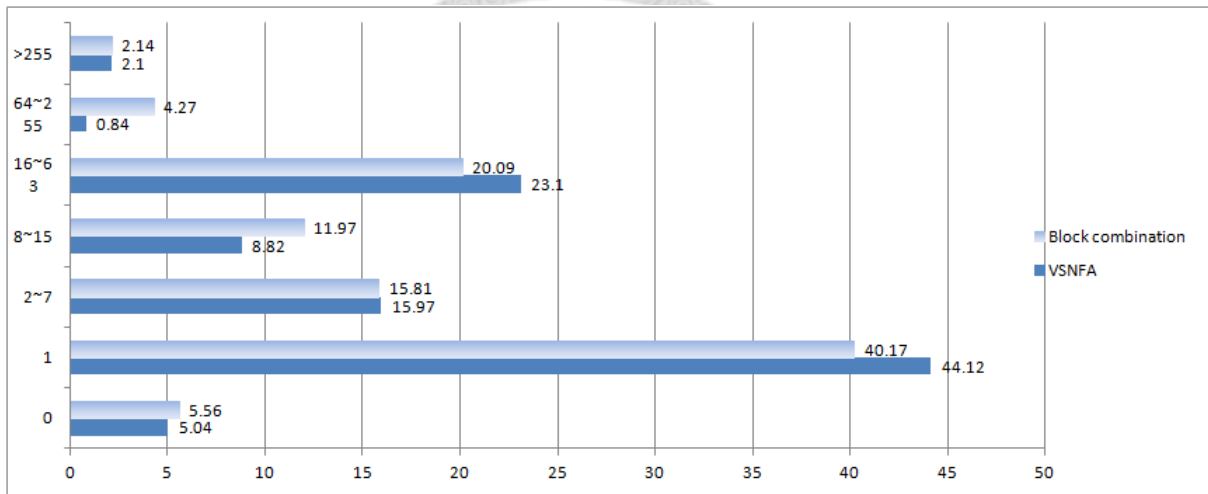


Figure 5.2: shows the distribution of the number of outgoing transitions per state

Chapter 6

Conclusion

Winnowing algorithm was extended to pattern matching in this paper. For improving its efficiency: the notation of limit was introduced to speed up the generating process of variable stride NFA; alphabet reduction was employed to decrease the usage of memory; block combination was utilized to increase the throughput of variable stride NFA.

As a result of the experiment shown, variable stride and block combination scheme are shown to improve scanning time relative to both the original method and the double stride method. However, this also increases the usage of memory.

In addition, the time requirements for generating variable stride NFA remains a problem. The future practicality of this whole algorithm will rest on developing improvements within the generating process.

References

- [1] Snort. [Online]. Available: <http://www.snort.org/>
- [2] Bro. [Online]. Available: <http://bro-ids.org/>
- [3] Cisco Adaptive Security Appliance. [Online]. Available: <http://www.cisco.com/>
- [4] Clamav. [Online]. Available: <http://www.clamav.net/>
- [5] M. Becchi and P. Crowley, “An improved algorithm to accelerate regular expression evaluation,” in *Proc. of ACM ANCS'07*, 2007, pp. 145–154.
- [6] D. Ficara, S. Giodano, G. Procissi, F. Vitucci, G. Antichi, and A. D. Pietro, “An improved DFA for fast regular expression matching,” *ACM SIGCOMM'08 Computer Communication Review*, vol. 38, Issue 5, pp. 29–40, Oct. 2008.
- [7] L. Yang, R. Karim, V. Ganapathy, and R. Smith, “Improving NFA-based signature matching using ordered binary decision diagrams,” in *Proc. of RAID'10*, 2010, pp. 58–78.
- [8] B. Brodie, R. Cytron, and D. Taylor, “A scalable architecture for high-throughput regular-expression pattern matching,” in *Proc. of ISCA'06*, 2006, pp. 191–202.
- [9] M. Becchi and P. Crowley, “Efficient regular expression evaluation: Theory to practice,” in *Proc. of ACM/IEEE ANCS'08*, 2008, pp. 50–59.

- [10] N. Hua, H. Song, and T. Lakshman, “Variable-stride multi-pattern matching for scalable deep packet inspection,” in *Proc. of IEEE INFOCOM’09*, 2009, pp. 415–423.
- [11] S. Schleimer, D. S. Wilkerson, and A. Aiken, “Winnowing: Local algorithms for document fingerprinting,” in *Proc. of ACM SIGMOD’03 on Management of data*, 2003, pp. 76–85.
- [12] K. Thompson, “Regular expression searching algorithm,” *Communication of ACM*, vol. 11, Issue 6, pp. 419–422, Jun. 1968.
- [13] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 1979.
- [14] A. V. Aho and M. J. Corasick, “Efficient string matching: An aid to bibliographic search,” *Communications of the ACM*, vol. 18, Issue 6, pp. 333–340, Jun. 1975.
- [15] Libpcap. [Online]. Available: <http://www.tcpdump.org/>
- [16] S. Kumar, B. Chandrasekaran, J. Turner, and G. Varghese, “Curing regular expressions matching algorithms from insomnia, amnesia, and acalculia,” in *Proc. of ACM/IEEE ANCS’07*, 2007, pp. 155–164.
- [17] R. Smith, C. Estan, S. Jha, and S. Kong, “Deflating the big bang: Fast and scalable deep packet inspection with extended finite automata,” in *Proc. of ACM SIGCOMM’08 conference on Data communication*, 2008, pp. 207–218.
- [18] M. Becchi and P. Crowley, “A hybrid finite automaton for practical deep packet inspection,” in *Proc. of ACM CoNEXT’07*, 2007.

- [19] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, and J. Turner, “Algorithms to accelerate multiple regular expressions matching for deep packet inspection,” in *Proc. of ACM SIGCOMM’06*, 2006, pp. 339–350.
- [20] Y. Sun, H. Liu, V. C. Valgenti, and M. S. Kim, “Hybrid regular expression matching for deep packet inspection on multi-core architecture,” in *Proc. of 19th International Conference on Computer Communications and Networks*, Aug. 2010, pp. 1–7.
- [21] S. Kumar, J. Turner, and J. Williams, “Advanced algorithms for fast and scalable deep packet inspection,” in *Proc. of ACM/IEEE ANCS’06*, 2006, pp. 81–92.
- [22] M. Becchi and S. Cadambi, “Memory-efficient regular expression search using state merging,” in *Proc. of IEEE INFOCOM’07*, May 2007, pp. 1064–1072.
- [23] N. Cascarano, P. Rolando, F. Risso, and R. Sisto, “iNFAnt: NFA pattern matching on GPGPU devices,” *ACM SIGCOMM’10 Computer Communication Review*, vol. 40, Issue 5, pp. 20–26, Oct. 2010.
- [24] R. Smith, N. Goyal, J. Ormont, K. Sankaralingam, and C. Estan, “Evaluating GPUs for network packet signature matching,” in *Proc. of IEEE International Symposium on Performance Analysis of Systems and Software*, Apr. 2009, pp. 175–184.
- [25] G. Vasiliadis and S. Ioannidis, “GrAVity: a massively parallel antivirus engine,” in *Proc. of Proceedings of the 13th international conference on Recent advances in intrusion detection*, 2010, pp. 79–96.
- [26] Y. Zu, M. Yang, Z. Xu, L. Wang, X. Tian, K. Peng, and Q. Dong, “Gpu-based nfa implementation for memory efficient high speed regular expression

- matching,” in *Proc. of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, 2012, pp. 129–140.
- [27] J. van Lunteren, “High-performance pattern-matching for intrusion detection,” in *Proc. of INFOCOM’06*, Apr. 2006, pp. 1–13.
- [28] W. Lin and B. Liu, “Pipelined parallel AC-based approach for multi-string matching,” in *Proc. of 14th IEEE International Conference on Parallel and Distributed Systems*, Dec. 2008, pp. 665–672.
- [29] I. Bonesana, M. Paolieri, and M. D. Santambrogio, “An adaptable FPGA-based system for regular expression matching,” in *Proc. of Design, Automation and Test in Europe*, Mar. 2008, pp. 1262–1267.
- [30] N. Yamagaki and R. S. S. Kamiya, “High-speed regular expression matching engine using multi-character nfa,” *International Conference on Field Programmable Logic and Applications*, pp. 131–136, Sep. 2008.
- [31] A. Mitra, W. Najjar, and L. Bhuyan, “Compiling PCRE to FPGA for accelerating SNORT IDS,” in *Proc. of ANCS’07*, Dec. 2007, pp. 127–136.
- [32] Mansoor and M. M. V. Kumar, “High speed pattern matching for network IDS/IPS,” in *Proc. of the 2006 14th IEEE International Conference on Network Protocols*, Nov. 2006, pp. 187–196.
- [33] A. Bremler-Barr, D. Hay, and Y. Koral, “Compactdfa: Generic state machine compression for scalable pattern matching,” in *Proc. of IEEE INFOCOM’10*, Mar. 2010, pp. 1–9.
- [34] F. Yu, R. H. Katz, and T. V. Laksman, “Gigabit rate packet pattern-matching using TCAM,” in *Proc. of the 12th IEEE International Conference on Network Protocols*, Oct. 2004, pp. 174–183.

- [35] Y. Sun, V. C. Valgenti, and M. S. Kim, “NFA-based pattern matching for deep packet inspection,” in *Proc. of 20th International Conference on Computer Communications and Networks*, Jul. 2011, pp. 1–6.

