ISSN:1991-8178

Australian Journal of Basic and Applied Sciences

Journal home page: www.ajbasweb.com

Lookup Table for Hardware Based Signature Matching using Finite Automata

[1]Jasmine C. and [2]Dr. Latha T.

[1]Assistant Professor / Computer Science and Engineering, Mar Ephraem College of Engineering and Technology, Marthandam, India
[2]Professor / Electronics and Communication Engineering, St. Xavier's Catholic College of Engineering, Nagercoil, India

ARTICLE INFO

Article history:
Received 20 January 2015
Accepted 02 April 2015
Published 20 May 2015

Keywords:
Network Intrusion Detection System, signature matching, finite automata, Content Addressable Memory, hashing techniques.

ABSTRACT

Signature matching is a dominant area in Network Intrusion Detection System (NIDS), which can be implemented in both hardware and software. Researchers, now focus on hardware implementation of NIDS, since the software system is slow when the number of loop increases and also, the software NIDS itself is vulnerable to security attacks. In the hardware implementation, the signature matching plays an important role, where the better performance in terms of the space complexity can be achieved using the Finite Automata (FA) based approach. In the signature matching, the searching of data in the memory is the vital part. In most of the researches, the authors focused in Content Addressable Memory (CAM) based searching and hashing techniques. The Objective is to implement a less power consuming and less complex hardware for searching. The proposed Lookup Table consumes less power and it is simple. Hence, instead of using CAM and Hashing Circuitry for searching, Lookup Table can be used in low cost hardware based NIDS.

To Cite This Article: Jasmine C. and Dr. Latha T., Lookup Table for Hardware Based Signature Matching using Finite Automata. ***Aust. J. Basic & Appl. Sci.,*** *9(16): 155-161, 2015*

INTRODUCTION

The Network Intrusion Detection System (NIDS) is a booming research area in the Information Security domain. It helps in preserving the valuable data and information of an individual or organization. The NIDS has two major sections, signature matching and anomaly detection. The signature matching is to find the known malwares and the anomaly detection for the unknown malwares. Nowadays, the software based NIDS are dominant which are vulnerable to security attacks. They are slow when compared to the hardware systems. In the hardware implementation of signature matching, the memory architectures are providing the better performance. Those architectures need searching, which can be implemented in many ways. The most common methods for searching are Content Addressable Memory (CAM) based searching and searching using the hashing techniques.

A signature is the group of characters found along with the source code of malwares. This signature in the malware is compared with the stored signature patterns. This process is called signature matching. In hardware architectures, we can't use any database for storing large number of signatures in a compact way. So, we need a special structure named Finite Automata or State Machines. Here, the signatures are compiled into Finite Automata. The automata is then reduced by certain algorithms (Lin and Chang, 2009) and converted into state transition table and this compact state transition table is stored in the hardware's memory. The Figure 1 shows a Finite Automata. Here the signatures are, "pcd", "fgh", "bcm". The dotted lines indicate the failure transitions. The node 1 is the starting state and the double circled node is the ending state. The Figure 2 shows the corresponding memory architecture. Here, the inputs are the initial state and the input character from network packet. For the particular state and the input character, the address decoder finds the RAM address where the transition table is stored and the next state is determined. This process will be repeated until there is a match. If there is a match, malicious pattern is available in the incoming packet and NIDS will block the particular packet or inform the administrator.

Corresponding Author: Jasmine C, Assistant Professor, Computer Science and Engineering, Mar Ephraem College of Engineering and Technology, Marthandam, India
Phone:+919698857391, E-mail: rojasmin@gmail.com,

Australian Journal of Basic and Applied Sciences, 9(16) Special 2015, Pages: 155-161

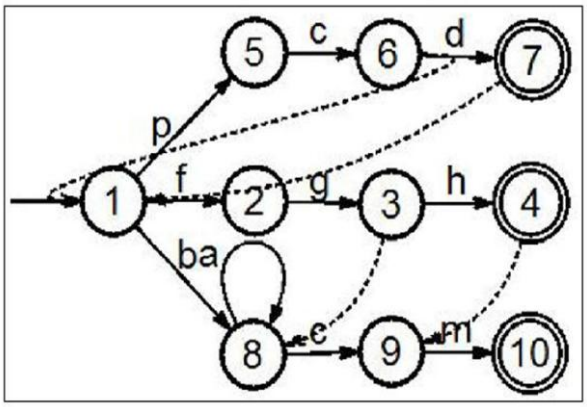

Fig. 1: A Finite Automaton

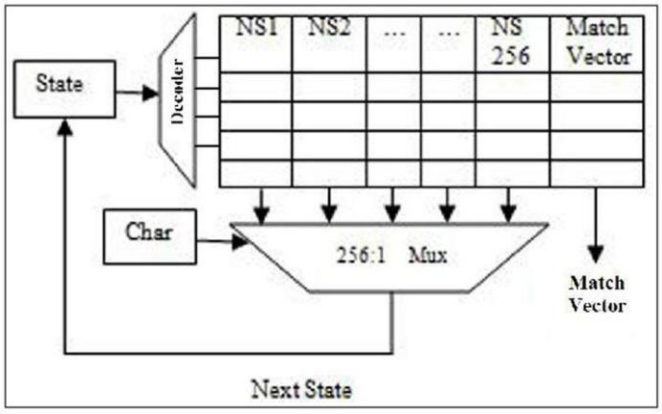

Fig. 2: Basic Memory Architecture

In the basic memory architecture (Lin and Chang, 2009), the RAM address mapping is done by the address decoder. This address decoder can be a component of a CAM or it can be replaced by a hash table. The Figure 3 shows the conceptual view of a CAM (Pagiamtzis and Sheikholeslami, 2009). Here the search data register is n bits long and CAM contains n number of storage cells and N number of search lines which work in parallel in a single clock signal. The stored word is also n bits long. Each storage cell is connected with match lines. When the n bits present in the search data register matches with the n bits in the stored word, the corresponding match line will be "ON" and the corresponding RAM address will be generated with the help of the encoder. Since the CAM search lines work in parallel, the power consumption is more.

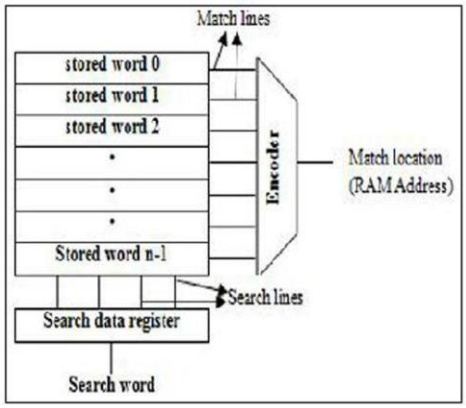

Fig. 3: Conceptual view of a CAM

Another option for address mapping is by using the hashing techniques. In hashing, the key or the input data is divided (modulo division - %) by the table (memory) size. The result of this computation

will give the address of the RAM. For example, if the input data is 6 and the table size is 10, then 6%10=6. This result 6 is used for finding the RAM address where the data 6 is stored. This will not work always, since there is possibility for collision. If we store the data 16 also, we will get the same memory address 6. So, we need to adopt collision resolution strategies, which will increase the number of computations done. A number of hashing techniques are available. But they are computationally complex. The CAM based searching is costly in terms of hardware. The hash based searching is costly in terms of time complexity, which requires complex computations. This can be overcome in our Lookup Table based searching.

Related Works:

The CAM based searching (Cheng and Chang, 2009) is used to find the next state and the path vector, for a particular transition in the Finite Automata. The CAM does the searching in one clock cycle which needs some complex circuitry and also requires more power and more silicon area (Beamer and Akgul, 2009; Anh-Tuan *et al*., 2013; Nelson *et al*., 2004; Pagiamtzis and Sheikholeslami, 2009). The hash based searching (Lin and Chang, 2013) includes complex computation known as perfect hashing for the signature matching in the GPU architectures. The Wu-Manber algorithm (Zhang *et al*., 2009; Xiao-shan *et al*., 2006; Hong, 2006), needs hash table in the filter mechanism when the shift value is zero. The Bloom Filter algorithm (Soliman and El-Helw, 2005) requires "k" different hash functions to check all the locations in the set S. The trie table generation and one step hash for virus detecting processor (Cheng *et al*., 2012) requires the hashing technique to generate hash values which gives addresses of the root for each light weight trie tree.

CAM is widely used in implementing lookup operations because of its speed (Sheikholeslami, 2003). The speed of the CAM comes from the increased silicon area and power consumption. These are the two parameters that the designers are trying to reduce. For a large capacity CAM, the more power will be needed. Reducing the power without sacrificing the speed and area is a major challenging area among the researchers. In CAM, the word to be searched is the input, which will be broadcasted to all the search lines. Each stored word in the CAM is connected to a match line. The match line will indicate whether the word to be searched and the word stored in the CAM are same. The match lines are connected to an encoder which will generate the binary address or memory location where the data is stored in the RAM. Sometimes, more than one stored word may match with the search word. At that time, we should use a priority encoder.

In CAM, the entire contents will be searched in a single clock cycle. The binary CAM will search for exact matches. But there is another type of CAM called the ternary CAM (TCAM), which will consider don't cares also. The TCAM is helpful for searching longest prefix matches, in which the don't cares acts as the wild card. In a paper, (Beamer and Akgul, 2009), the CAM is implemented in such a way that, the sensing scheme saves the power. We can achieve a slightly low power CAM by reducing the match line activity factor using NAND type CAM cells. But it will increase the delay. On the other hand, NOR type CAM can be used, but the match line activity will be high, which consumes more power, but the delay is reduced. So, the CAM was built with a combination of NAND and NOR cells (TuanDo *et al*., 2013), and also use pre-computation technique to reduce the number of search lines.

The match lines of conventional CAM are pre charged and then discharged based on match or not. But this will reduce the performance. The search line in conventional CAM is complementary search lines, which increase power consumption and delay. CAM consumes more power due to parallel match line comparison. Alternatively, a parity bit technique was used (Pedro *et al*., 2013), which reduces 39% sensing delay at the cost of 1% power and area. Moreover, CAM is having low memory density, but the power consumption is more when compared with DRAMs and SRAMs. In CAM, it is very difficult to implement the column address. The power consumption is more in two circumstances. One is, while performing parallel search. During parallel search, it is needed to supply power to all memory locations and to all the cell arrays. Second is the match detection circuit.

MATERIALS AND METHODS

The overall architecture is given in Figure 4. From that architecture, we are focusing on the Lookup Table in this paper. The proposed method for building the Lookup Table is divided into two sections. The first section deals with finding the number of outgoing edges of the nodes of a Finite Automaton. And the second section deals with the formation of Lookup Table.

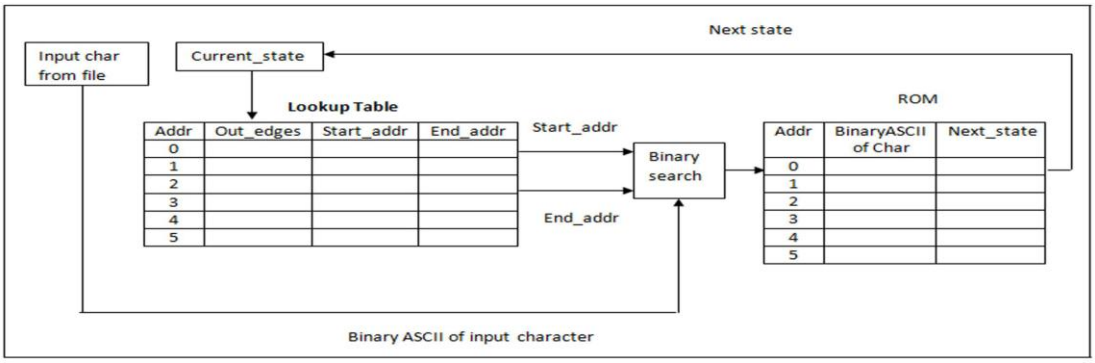

Fig. 4: Overall Architecture

Finding the no of outgoing edges:

In this architecture shown in Figure 5, every character of a Finite Automaton is represented as 8-bit binary of its ASCII. The data of automaton is stored in file, one character in each row and then moved to the input memory. The input memory is 8 bits wide, which represents 1 character which is fetched in one clock cycle. Then the 8bit ASCII of the character is given to the first comparator. The output of the comparator is given to the counters. The first counter counts non-zero characters and the second counter counts the zero characters. The output of two counters is given to an adder which adds the counts. This is helpful to count the number of non-zero characters present in a row of two dimensional array. Here we are considering the automata as a graph which consists of 6 nodes. So, there will be 6X6 entries when represented in the 2 dimensional array. But the RAM is a one dimensional memory array. So, we need to represent the data in two dimensional array format into one dimensional array format. Hence, when the output of the adder becomes 6, the data present in the non-zero counter will be written into the first location of the output memory. Also, the counters will be cleared and the counting will be repeated and cleared after six comparisons. Thus, the first location of the output memory contains the number of outgoing edges of node0 and the second location of the output memory contains the number of outgoing edges of the node1 and so on.

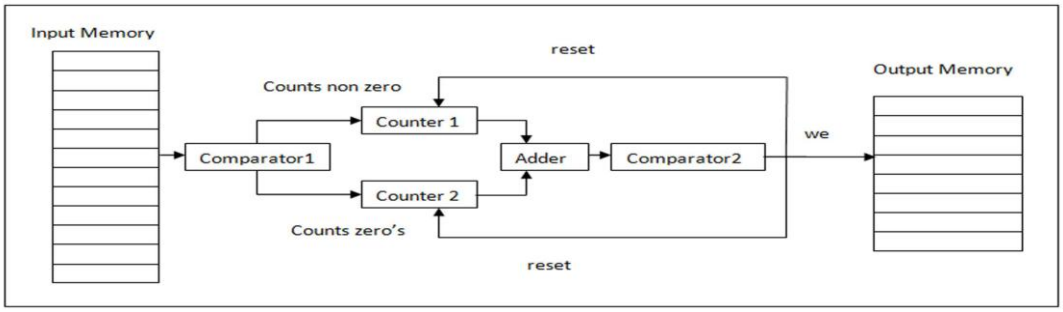

Fig. 5: Finding the Outgoing edges

Formation of the Lookup Table:

The Lookup Table can be formed by the following logic. It consists of four entries, the node, no of outgoing edges, starting address and the ending address. The starting address of the first location will be zero. The starting address for the remaining locations will be formed by adding a one to the previous ending address. Similarly the ending address can be formed by subtracting a one from the sum of the current starting address and the current outgoing edges. For explanation purpose, the automaton given in the Figure 6 is taken, which consists of 6 nodes. The signatures compiled in this automaton are "ab", "ac", "de". The double circled nodes are the ending states and the node0 is the starting state. The corresponding Lookup Table is given in Table I.

Australian Journal of Basic and Applied Sciences, 9(16) Special 2015, Pages: 155-161

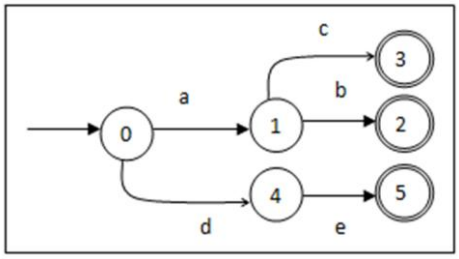

Fig. 6: Sample Automaton

Table I: Lookup Table

Node	Number of Outgoing Edges	Starting Address	Ending Address
0	2	0	1
1	2	2	3
2	0	-	-
3	0	-	-
4	1	4	4
5	0	-	-

RESULTS AND DISCUSSION

The Lookup Table is implemented in Spartan3 family, XC3S50 device. The device utilization summary is shown in the following Table II.

The Figure 7 shows the Simulation waveform. The Figure 8 shows the RTL schematic of the design and the Figure 9 shows the placement of the proposed design in the device. The portions marked as black are the utilized portions. The total power consumption is 24mW.

Table II: Devices Utilization

Components	Used	Utilization
No of slice Flip Flops	23	1%
No of 4 input LUTs	22	1%
No of occupied slices	24	3%
No of slices containing only related logic	24	100%
No of slices containing unrelated logic	0	0%
No of bonded IOB	10	8%
No of Block RAMs	1	25%
No of GCLKs	1	12%

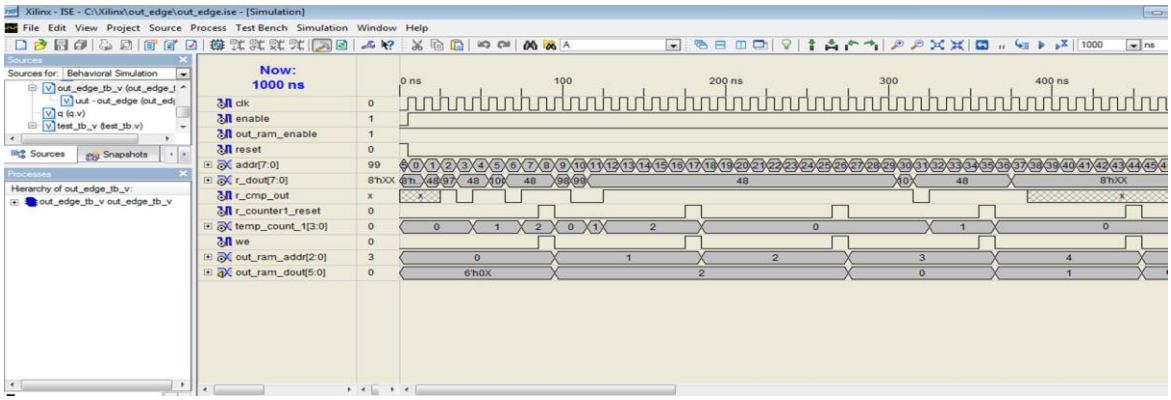

Fig. 7: Simulation Waveform

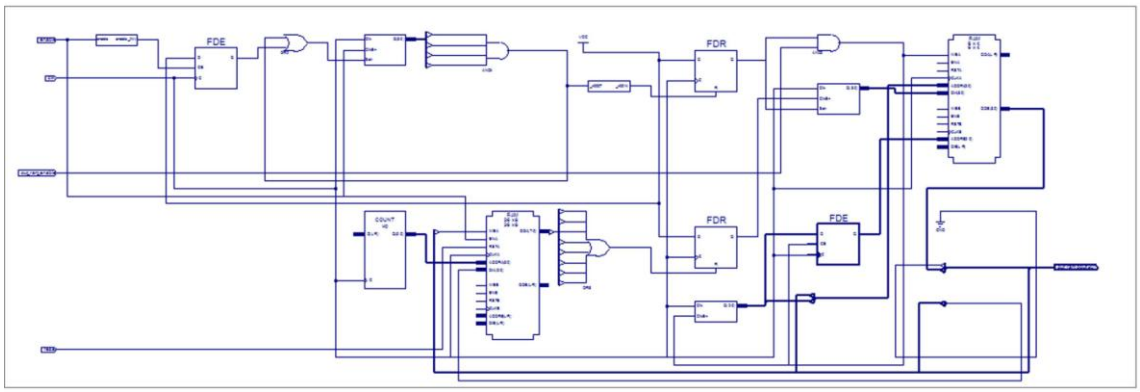

Fig. 8: RTL Schematic

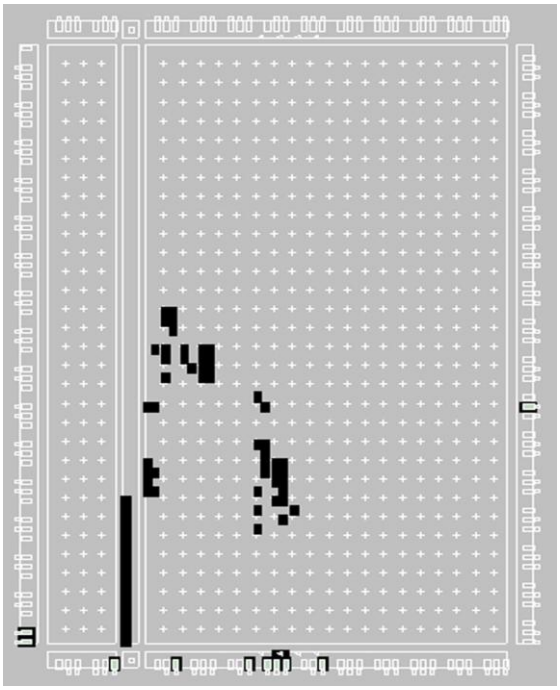

Fig. 9: Placement of design in device

Conclusion:

This Lookup Table based searching will use less power while compared to the CAM architectures and less complex, when compared to hashing techniques. Here the Lookup Table provides the starting address and the ending address of the particular block of the RAM. So, it is efficient to search only in those areas using binary search or parallel search rather than searching in all the areas as in the case of CAM. In this paper, only Lookup Table is focused. Our future work is to implement the complete signature matching using this Lookup Table based searching.

REFERENCES

Ali Sheikholeslami, Senior Member, 2003. IEEE, A Ternary Content-Addressable Memory (TCAM) Based on 4T Static Storage and Including a Current-Race Sensing Scheme, IEEE Journal of Solid-State Circuits, 38: 1.

Anh-Tuan Do, Shoushun Chen, Zhi-Hui Kong and Kiat Seng Yeo, 2013. A High Speed Low Power CAM With a Parity Bit and Power-Gated ML Sensing, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 21: 1.

Baojun Zhang, Xiaoping Chen, Xuezeng Pan and Zhaohui Wu, 2009. "High Concurrence Wu-Manber Multiple Patterns Matching Algorithm", Proceedings of the 2009 International Symposium on Information Processing (ISIP'09).

Cheng-Hung Lin, Chen-Hsiung Liu, Shih-Chieh Chang, Wing-Kai Hon, 2012. "Memory-Efficient Pattern Matching Architectures Using Perfect Hashing on Graphic Processing Units", 2012 Proceedings IEEE Infocom.

Cheng-Hung Lin, Member, IEEE and Shih-Chieh Chang, Member, IEEE, 2009. "Efficient Pattern Matching Algorithm for Memory Architecture", IEEE Transactions on Very Large Scale Integration (VLSI) Systems.

Chieh-Jen Cheng, Student Member, IEEE, Chao-Ching Wang, Member, IEEE, Wei-Chun Ku, Student Member, IEEE, Tien-Fu Chen, Member, IEEE, and Jinn-Shyan Wang, Member, IEEE, 2012. "A Scalable High-Performance Virus Detection Processor Against a Large Pattern Set for Embedded Network Security", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 20: 5.

Geir Nilsen, Jim Torresen and Oddvar Søråsen, 2004." A Variable Word-Width Content Addressable Memory for Fast String Matching", Department of Informatics, University of Oslo, P.O. Box 1080 Blindern, 0316 Oslo, Norway.

Igor Arsovski, Student Member, IEEE, 2006. Trevis Chandler, Student Member, IEEE, and Kostas Pagiamtzis, Student Member, IEEE, and Ali Sheikholeslami, Senior Member, IEEE, "Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey", IEEE Journal of Solid-State Circuits, 41: 3.

Kostas Pagiamtzis, Student Member, IEEE, and Ali Sheikholeslami, Senior Member, IEEE, 2006. "Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey", IEEE Journal of Solid-State Circuits, 41: 3.

Mohamed Ali Soliman and AMR El-Helw, 2005. School of Computer Science, University of Waterloo "Network Intrusion Detection System Using Bloom Filters", Algorithmic Foundations of the Internet - Winter.

Pedro Echeverr´ıa, Jos´e L. Ayala, Marisa L´opez-Vallejo, 2004. Practical Implementation of a Low-Power Content-Addressable Memory, Design of Circuits and Integrated Systems.

Scott Beamer, Mehmet Akgul, 2009." Design of a Low Power Content Addressable Memory (CAM)", University of California, Berkeley.

SUN Xiao-shan,WANG Qiang, GUAN Yi, WANG Xiao-long, 2006. "An improved Wu-Manber multipattern matching algorithm and its applications", Journal of Chinese Information Processing.

Yang Dong hong, 2006. "An improved Wu-Manber algorithm", Performance, Computing, and Communications Conference, 2006. IPCCC 2006. 25th IEEE International.