A design of virus detection processor for embedded network security

S.SHAMILI
M.E.Vlsi Design(PG Scholar),
Srinivasan Engineering College,
Perambalur-621212,
TamilNadu, India
sksshamili@gmail.com.

B.KARTHIGA
Associate Professor,
Srinivasan Engineering College,
Perambalur-621212,
TamilNadu, India
karthiga.jaya@yahoo.com.

***Abstract*-In an Intrusion Detection System (IDS) has emerged as one of the most effective way of furnishing security to those connected network and the heart of the modern intrusion detection system is a pattern matching algorithm. A network security application needs the ability to perform the pattern matching to protect against attacks like viruses and spam. The solutions for firewall are not scalable; they don't address the difficult of antivirus. The main work is to furnish the powerful systematic virus detection hardware solution with minimum memory for network security. Instead of placing the entire patterns on a chip, a two phase antivirus processor works by condensing as much of the important filtering information as possible onto a chip. Thus, the proposed system is mainly concentrated on reducing the memory gap in on chip memory along with the stage of Exact-matching engine.**

Keywords-memory gap, virus detection, network security, embedded system

IJERT

I. INTRODUCTION

To make a network environment, firewalls were first announced to block unauthorized Internet users from accessing resources in a network by checking the packet head (MAC address/IP address/port number). This method significantly reduces the probability of being attacked. Therefore, traditional firewalls no longer furnish enough protection. Initially, the solutions were implemented at the end-user side but tend to be merged into firewalls to provide multi-layered protection.

The Fig. 1 shows a typical architecture of a firewall router. When a new connection is established, the firewall router should scan the connection and forwards these packets to the host after confirming that the connection is secure.

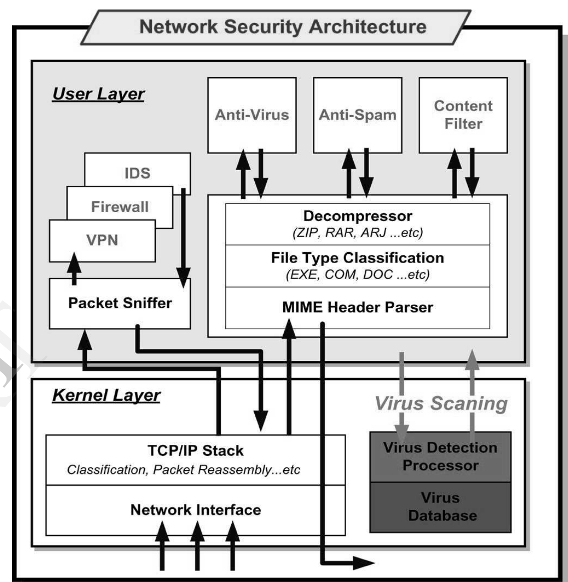

Fig.1 Architecture of firewall router

The router might initially disclaim some connections from the firewall based on the target's IP address and the connection port. Then, the router would monitor the content of web pages to prevent the user from accessing any page that connects to malware links.

When the user wants to download a file, to ensure that the file is not infected, the firewall must decompress this file and check it using anti-virus programs. The firewall routers require several time-consuming steps to provide a secure connection.

The major contribution is to reduce the memory gap in the on-chip memory while using external memory. Here the

proposed algorithm used to reduce the memory gap of on-chip memory.

II. VIRUS DETECTION PROCESSOR

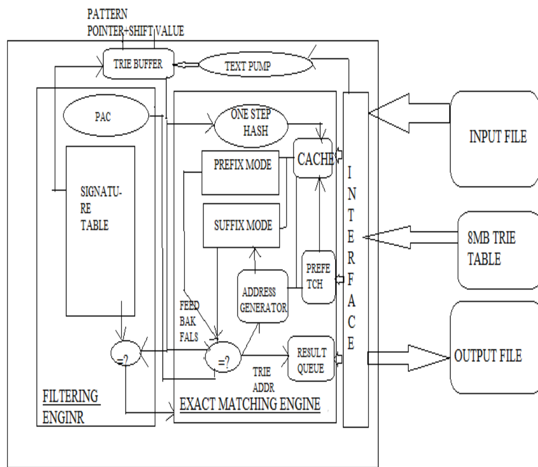

Fig.2 virus detection processor architecture

The design, shown on Fig. 2 is a two-phase pattern-matching architecture mostly comprising the filtering engine and the exact-matching engine. The filtering engine is a front-end module responsible for filtering out secure data efficiently and indicating to candidate positions that patterns possibly exist at the first stage. The exact-matching engine is a back-end module responsible for verifying the alarms caused by the filtering engine. In the second stage of exact-matching engine, only a few unsaved data need to be checked.

Both engines have individual memories for storing significant information. For cost reasons, only a small amount of significant information regarding the patterns can be stored in the filtering engine's on-chip memory. In this case, they used a 32-kB on-chip memory for the ClamAV virus database, which contained more than 30 000 virus codes and localized most of the computing inside the chip.

Conversely, the exact-matching engine not only stores the entire pattern in external memory but also provides information to speed up the matching process. The exact-matching engine is space-efficient and requires only four times the memory space of the original size pattern set. The size of a pattern set is the sum of the pattern length for each pattern in the given pattern set.

III. FILTERING ENGINE

In this work, the overall performance strongly depends on the filtering engine. Here, introduce two classical filtering algorithms for pattern matching in the following sections, then show how to merge their structures in the same space to improve the filter rate.

A. *Wu-Manber Algorithm*

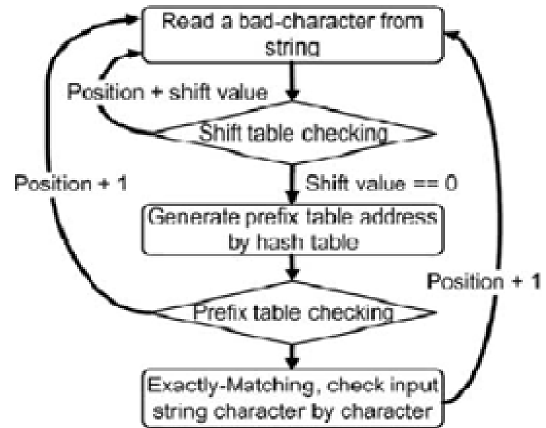

Fig. 3 Wu-Manber Matching process- Matching flow

IJERT

TheWu-Manber algorithm is a high-performance, multi-pattern matching algorithm based on the Boyer-Moore algorithm. It builds three tables in the preprocessing stage: a shift table, a hash table and a prefix table. The Wu-Manber algorithm is an exact-matching algorithm, but its shift table is an efficient filtering structure. The shift table is an extension of the bad-character concept in the Boyer-Moore algorithm, but they are not identical.

The fig.3 shows the matching process of matching flow algorithm. The shift table gives a shift value that skips several characters without comparing after a mismatch. After the shift table finds a candidate position, the Wu-Manber algorithm enters the exact-matching phase and is accelerated by the hash table and the prefix table. The performance of the Wu-Manber algorithm is not proportional to the size of the pattern set directly, but it is strongly dependent on the minimum length of the pattern in the pattern set.

B. *Bloom Filter Algorithm*

A Bloom filter is a space-efficient data structure used to test whether an element exists in a given set. This algorithm is composed of different hash functions and a long vector of bits. Initially, all bits are set to 0 at the preprocessing stage. To

add an element, the filter hashes the element by these hash functions and gets positions of its vector.

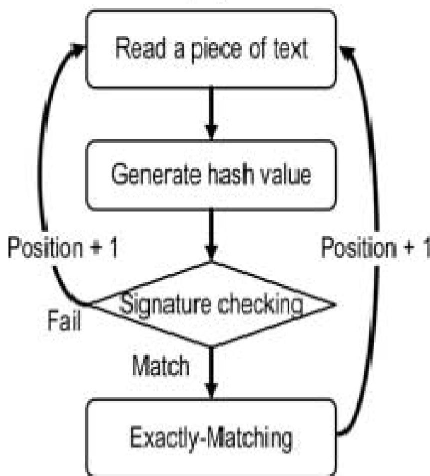

Fig. 4 Bloom filter matching process- matching flow

The Bloom filter sets the bits at these positions to 1. The value of a vector that only contains an element is called the signature of an element. To verify the membership of a particular element, the Bloom filter hashes this element by the same hash functions at run time, and it also generates positions of the vector.

The output of the Bloom filter can be a false positive but never a false negative. Therefore, some pattern matching algorithms based on the Bloom filter must operate with an extra exact-matching algorithm. However, the Bloom filter still features the following advantages: 1) it is a space-efficient data structure; 2) the computing time of the Bloom filter is scaled linearly with the number of patterns; and 3) the Bloom filter is independent of its pattern length.

The Fig. 4 describes a typical flow of pattern matching by Bloom filters. This algorithm fetches the prefix of a pattern from the text and hashes it to generate a signature. Then, this algorithm verifies whether the signature exists in the bit vector. If it is yes, it shifts the search window to the right by one character for each comparison and repeats the above step to filter out safe data until it finds a candidate position and launches exact-matching.

C. Shift-Signature Algorithm

It re-encodes the shift table to merge the signature table into a the shift-signature table. This table has the same size as the original shift table as its width and length. There are two fields, S-flag and carry, in the shift signature table. The carry field has two types of data: a shift value and a signature. The S-flag is used to indicate the data type of a carry. The filtering engine can then filter the text using a algorithm while providing a higher filter rate.

First, the algorithm generates two tables, a shift table and signature table.The generation of the shift table is the same as in the Wu-Manber algorithm. The shift table is used as the primary filter. The signature table could be considered a set of the bit vector of the Bloom filter, and it is used for the second-level filtering. The signature table's generation is similar to the Bloom filter but is not identical.

IV. EXACT-MATCH ENGINE

The EME must verify the false positives when the filtering engine alerts. It also identifies patterns for upper-layer applications. Most exact-match algorithms use the two kinds of trie structures shown in Fig. 5 loose and compact tries, to establish their pattern databases. Both trie structures have their merits.

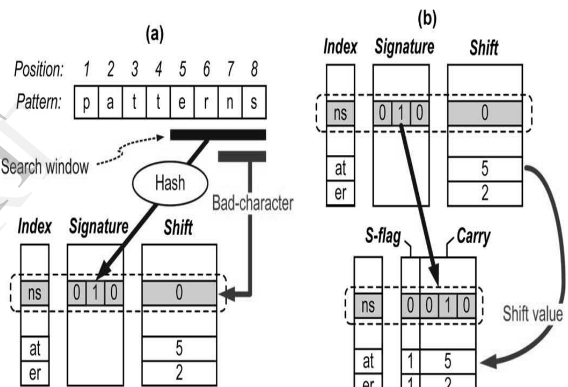

Fig.5 (a)table generation and (b) re-encodes of shift-signature table

Unlike loose tries, compact tries construct pattern databases with two pointers, sibling and child, to reduce their memory requirements. However, this method has potential performance problems because it may search link lists formed by sibling pointers. Attacks can be avoided by removing patterns that cause attacks before constructing the pattern database. For this reason, our exact-matching engine's algorithm use compact tries and propose several solutions to mitigate the effect of algorithmic attacks.

A compact trie usually has only one entrance. However, for multiple patterns, this method needs a significant amount of time to search the prefix node of a pattern in the entrance's sibling list. To reduce search time, dividing a huge trie into several lightweight tries to generate multiple entrances by hashing the root node of each

IJERT

lightweight trie. The generated hash values are root addresses for each lightweight trie tree. Therefore, the exact-matching engine can easily get more entrances and have first nodes with short sibling lists. The exact-matching engine overlaps computation with memory access time. Therefore, simply hashing the trie to generate multiple entrances and carefully arranging the data in the memory efficiently solves the memory gap problem at the algorithmic level.

The trie-skip mechanism is implemented by four major fields for each trie node. The skip value is eight, meaning that the closest candidate pattern is behind the current candidate by eight characters. However, because the exact-matching engine does not always start from the beginning of a pattern, the jump node field indicates the first node that the exact-matching engine should compare for the new candidate pattern after a mismatch occurs. The suffix offset fixes the search window and notifies the exact-matching engine, which fetches characters behind the new pattern pointer with four characters. The jump-enable bit and jump node are used to implement this jumping idea.

V. PROPOSED SYSTEM

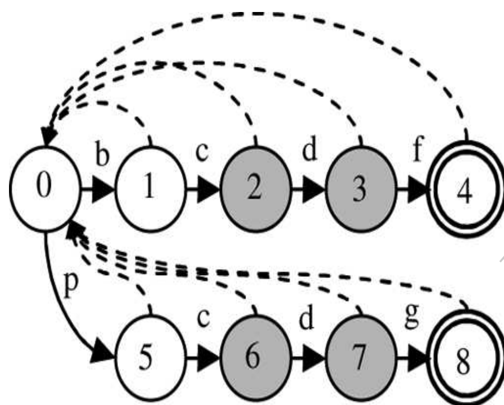

Fig .6 state diagram of AC algorithm

The Fig.6 shows the state transition diagram derived from the AC algorithm where the solid lines represent the *valid* transitions while the dotted lines represent a new type of state transition called the *failure* transitions. The failure transition is explained as follows.

Given a current state and an input character, the AC machine first checks whether there is a valid transition for the input character; otherwise, the machine jumps to the next state where the failure transition points. Then, the machine recursively considers the same input character until the character causes a valid transition.

Due to the common substring of string patterns, the compiled AC machine have states with similar transitions.despite the similarity, those similar states are not equilent states and cannot be merged directly. In this, the functional errors can be created if those similar states are merged directly. So a new mechanism can be proposed to rectify those functional errors after merging those similar errors.

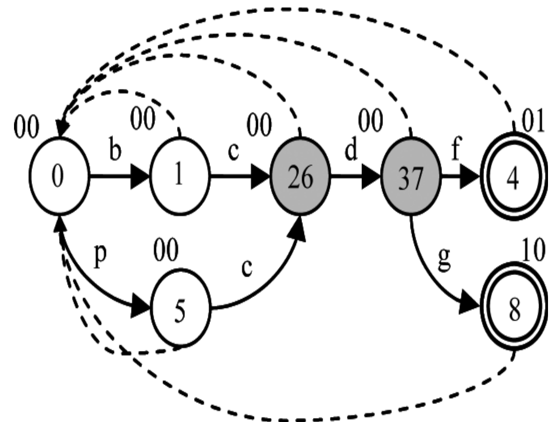

Fig.7 Merging similar states

The merg_FSM is a different machine from the original state machine but with a smaller number of states and transitions. A direct implementation of merg_FSM has a smaller memory objective is to modify the AC algorithm. It can store only the state transition table of merg_FSM in memory while the overall system still functions correctly as the original AC state machine does. The new state traversal mechanism guides the state machine to traverse on the merg_FSM and provides correct results as the original AC state machine. The Fig.7 shows the merged patterns that reduces the memory gap which is compared with the AC algorithm used.

IJERT

VI. SIMULATION RESULT

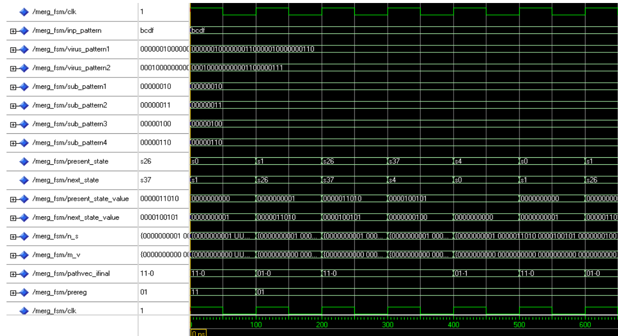

Fig .8 simulation result for merg_fsm algorithm

VI. CONCLUSION

In this paper, a novel architecture for pattern matching for network intrusion detection system. Thus here, the expected memory gap can be reduced with high performance, while using AC algorithm created in on-chip memory.

REFERENCES

[1] A. V. Aho and M. J. Corasick, "Efficient string matching: An aid to bibliographic search," *Commun. ACM*, vol. 18, pp. 333–340, 1975.
[2] R. S. Boyer and J. S. Moore, "A fast string searching algorithm," *Commun. ACM*, vol. 20, pp. 762–772, 1977.
[3] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, pp. 422–426, 1970.
[4] S. Dharmapurikar, P. Krishnamurthy, and T. S. Sproull, "Deep packet inspection using parallel bloom filters," *IEEE Micro*, vol. 24, no. 1, pp. 52–61, Jan. 2004.
[5] D. P. Scarpazza, O. Villa, and F. Petrini, "Peak-performance DFA based string matching on the Cell processor," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, 2007, pp. 1–8.
[6] O. Villa, D. P. Scarpazza, and F. Petrini, "Accelerating real-time string searching with multicore processors," *Computer*, vol. 41, pp. 42–50, 2008.
[7] D. P. Scarpazza, O. Villa, and F. Petrini, "High-speed string searching against large dictionaries on the Cell/B.E. processor," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, 2008, pp. 1–8.
[8] R.-T. Liu, N.-F. Huang, C.-N. Kao, and C.-H. Chen, "A fast stringmatching algorithm for network processor-based intrusion detection system," *ACMTrans. Embed. Comput. Syst.*, vol. 3, pp. 614–633, 2004.
[9] L. Tan and T. Sherwood, "A high throughput string matching architecture for intrusion detection and prevention," in *Proc. 32nd Annu. Int. Symp. Comput. Arch.*, 2005, pp. 112–122.
[10] F. Yu, R. H. Katz, and T. V. Lakshman, "Gigabit rate packet patternmatching using TCAM," in *Proc. 12th IEEE Int. Conf. Netw. Protocols*, 2004, pp. 174–183.
[11] R. Sidhu and V. K. Prasanna, "Fast regular expression matching using FPGAS," in *Proc. 9th Ann. IEEE Symp. Field-Program. Custom Comput. Mach. (FCCM)*, 2001, pp. 227–238.
[12] L. Tan and T. Sherwood, "A high throughput string matching architecture
for intrusion detection and prevention," in *Proc. 32nd Annu. Int. Symp. Comput. Arch. (ISCA)*, 2005, pp. 112–122.

IJERT