International Journal of Advanced Research in

Computer Science Engineering and Information Technology

Volume: 3 Issue: 3 18-Sep-2014,ISSN_NO: 2321-3337

A STUDY ON SCALABLE HIGH-PERFORMANCE VIRUS DETECTION PROCESSOR FOR ROOTED NETWORK SECURITY

[1] K.RAVIKUMAR, [2]K.KAVINILA,

[1]Associate Professor, Dept.of.Computer science, Tamil University Thanjavur, India.

[2]Reserts arch Scholar, Dept.of.Computer science, Tamil University Thanjavur.

ABSTRACT−*Network security applications generally require the ability to perform powerful pattern matching to protect against attacks such as viruses and spam. Traditional hardware solutions are intended for firewall routers. However, the solutions in the literature for firewalls are not scalable, and they do not address the difficulty of an antivirus with an ever-larger pattern set. Related works have focused on algorithms and have even developed specialized circuits to increase the scanning speed. However, these works have not considered the interactions between algorithms and memory hierarchy. Because the number of attacks is increasing, pattern-matching processors require external memory to support an unlimited pattern set. This method makes the memory system the bottleneck. However, when the pattern set is already intractably large, a perfect solution is unattainable.*

Keywords: ***Routing,hacker.***

1, INTRODUCTION

The main goal is to provide high performance in most cases while still performing reasonably well in the worst case. With an eye toward high performance, updatability, unlimited pattern sets and low memory requirements, a two-phase architecture is introduced so that it uses off-chip memory to support a large pattern set. The goal of this project is to provide a systematic virus detection hardware solution for network security for embedded systems. Instead of placing entire matching patterns on a chip, a new solution is to provide a two-phase dictionary-based antivirus processor that works by condensing as much of the important filtering information as possible onto a chip and infrequently accessing off-chip data to make the

matching mechanism scalable to large pattern sets. In the first stage, the filtering engine can filter out more than 93.1% of data as safe, using a merged shift table. Only 6.9% or less of potentially unsafe data must be precisely checked in the second stage by the exact- matching engine from off-chip memory. To reduce the impact of the memory gap, three enhancement algorithms are proposed to improve performance: 1) a skipping algorithm; 2) a cache method; and 3) a pre fetching mechanism.

2, EXISTING SYSTEM:

Many previous designs have claimed to provide high performance, but the memory gap created by using external memory decreases performance because of the increasing size of virus databases. Furthermore, limited resources restrict the practicality of these algorithms for embedded network security systems. Two-phase heuristic algorithms are a solution with a tradeoff between performance and cost due to an efficient filter table existing in internal memory; however, their performance is easily threatened by malicious attacks.

2.1. DISADVANTAGES:

1. The solutions in the literature for firewalls are not scalable, and they do not address the difficulty of an antivirus with an ever-larger pattern set.
2. Networking security has always been an important issue. End users are vulnerable to virus attacks, spam's and Trojan horses.
3. They may visit malicious websites or hackers may gain entry to their computers and use them as zombie computers to attack others.
4. The main goal is to provide high performance in most cases while still performing reasonably well in the worst case.

3, PROPOSED SYSTEM:

To ensure a secure network environment, firewalls were first introduced to block unauthorized Internet users from accessing resources in a private network by simply checking the packet head (MAC address/IP address/port number). This method significantly reduces the probability of being attacked. Virus Detection Processor is a two-phase pattern matching architecture mostly comprising the filtering engine and the exact-matching engine. The proposed

exact-matching engine also supports data prefetching and caching techniques to hide the access latency of the off-chip memory by allocating its data structure well. The other modules include a text buffer and a text pump that pre-fetches text in streaming method to overlap the matching progress and text reading. A load/store interface was used to support bandwidth sharing.

3.1. ADVANTAGES:

This work analyzes two scenarios of malicious attacks and provides two methods for keeping performance within a reasonable range.

In the first stage, the filtering engine can filter out more than 93.1% of data as safe, using a merged shift table.

Only 6.9% or less of potentially unsafe data must be precisely checked in the second stage by the exact matching engine from off-chip memory.

To reduce the impact of the memory gap, three enhancement algorithms are proposed to improve performance: 1) a skipping algorithm; 2) a cache method; and 3) a pre fetching mechanism.

4, MODULES

- Classification of Vulnerable Hosts
- Permutation-Scanning Worms
- Scanning Hosts at Different Layers
- Analytical Modeling or Simulation

4.1.Classification of Vulnerable Hosts

Two-phase pattern matching architecture mostly comprising the filtering engine and the exact-matching engine. The filtering engine is a frontend module responsible for filtering out secure data efficiently and indicating to candidate positions that patterns pos sibly exist at the first stage. The exact-matching engine is responsible for verifying the alarms caused by the filtering engine.

Exact-matching engine not only stores the entire pattern in external memory but also provides information to speed up the matching process. Our 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; in other words, it is the minimum size of the memory required to store the pattern set for the exact-matching engine.

Boyer-Moore algorithm uses a pattern pointer to locate a candidate position by assuming that a desired pattern exists at this position and then shifts its search window to the right of this pattern.

Wu and Manber (WM) modified the Boyer-Moore algorithm to search for multiple patterns.

4.2 Permutation-Scanning Worms

Number of attacks is increasing, pattern-matching processors require external memory to support an unlimited pattern set. This method makes the memory systemthe bottleneck.

Two-phase pattern matching architecture,

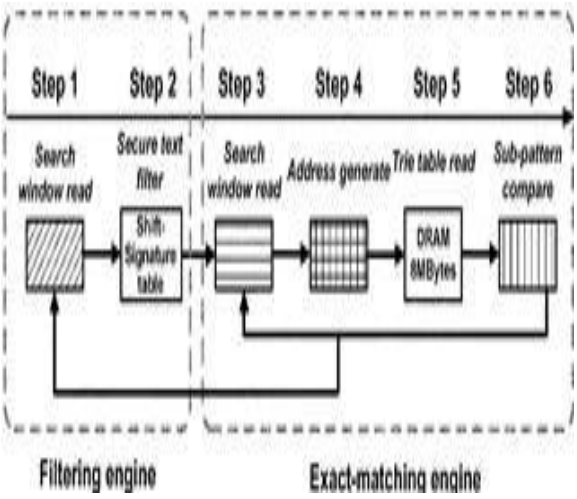
1. **Filtering engine**

A frontend module responsible for filtering out secure data efficiently and indicating to candidate positions that patterns possibly exist at the first stage.

2. **Exact-matching engine**

Responsible for verifying the alarms caused by the filtering engine. Only a few unsaved data need to be checked precisely by the exact-matching engine in the second stage.

Also supports data pre fetching and caching techniques to hide the access latency of the off-chip memory by allocating its data structure well.



4.3. ***Wu-Manber Algorithm***

High-performance, Multi-pattern matching algorithm based on the Boyer-Moore algorithm. It builds three tables in the pre processing stage: a shift table, a hash table and a prefix table.

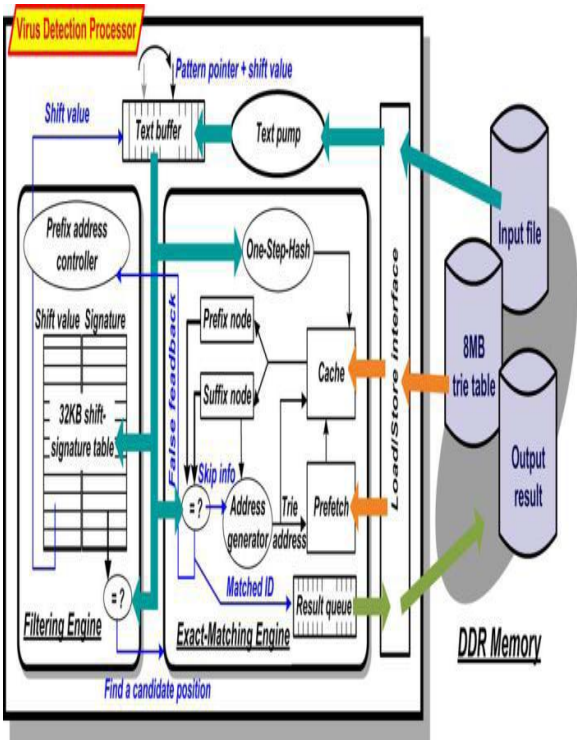
The Wu-Manber algorithm is an exact-matching algorithm, but its shift table is an efficient filtering structure. The shift table is an extension of the bad-character concept in the Boyer-Moore algorithm, but they are not identical.

4.4. Bloom Filter Algorithm

Composed of different hash functions and a long vector of bits. Initially, all bits are set to 0 at the pre processing stage. To add an element, the Bloom filter hashes the element by these hash functions and gets positions of its vector & then 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.

4.5.Sample Entry and Exit Criteria for Regression Testing

Entry Criteria



- The defect is repeatable and has been properly documented
- A change control or defect tracking record was opened to identify and track the regression testing effort
- A regression test specific to the defect has been created, reviewed, and accepted

Exit Criteria

- Results of the test show no negative impact to the application

4.6.Analytical Modeling or Simulation

Properly simulating the worm propagation on the Internet at the packet level is very difficult due to its sheer scale. Even for a rather simplified version of the Internet, without an analytical model, one will need to take the average of multiple runs of a simulator in order to get acceptably reliable propagation curves. Since each run could potentially take a long time for realistic values of and , the whole process could take an enormous amount of time. For an imagined attack targeting at the Windows ystem, it took 16 h on an Intel Xeon 2.8-GHz processor with 4 GB RAM to run a *single* round of a simulation involving around 400Mpotentially vulnerable windows hosts on IPv4 for one set of worm/network parameters. In order to run the same simulation for IPv6, it is easy to see that the runtime will be astronomical. On the contrary, a *single* run of the numerical simulation based on the analytical model takes just seconds and gives us the provably correct results. Moreover, it canhandle extremely large address spaces and vulnerable host populations. For any worm/network parameter change, new propagation curves can be recomputed in little time for comparison.

5, CONCLUSIONS

The propagation characteristics of different varieties of permutation-scanning worms. To verify the correctness of the model, we compare the results from our model with those obtained from actual worm simulations and show that they perfectly match. We analyze the model to demonstrate how each worm/network parameter will affect the worm's propagation behavior. Finally, although our analytical model was originally conceived by assuming ideal network conditions, we show that it can very well be extended to real-life scenarios with the consideration of variable host bandwidth, network congestion, Internet delay, host crash, and patching. In our future work, we will continue refining our model by considering other practical extensions. Many previous designs have claimed to provide high performance, but the memory gap created by using external memory decreases performance because of the increasing size of virus databases

REFERENCES

[1] Chieh-Jen Cheng, Chao-Ching Wang, Wei-Chun Ku, Tien-Fu Chen , and Jinn-Shyan Wang, "Scalable High-Performance Virus Detection Processor Against a Large Pattern Set for Embedded Network Security" Commun. vol. 51, pp. 62–70,2011.

[2] O. Villa, D. P. Scarpazza, and F. Petrini, "Accelerating real-time string searching with multicore processors," Computer, vol. 41, pp. 42–50,2008.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] R.-T. Liu, N.-F. Huang, C.-N. Kao, and C.-H. Chen, "A fast string matching algorithm for network processor-based intrusion detection system," ACMTrans. Embed. Comput. Syst., vol. 3, pp. 614–633, 2004.

[8] F. Yu, R. H. Katz, and T. V. Lakshman, "Gigabit rate packet pattern matching using TCAM," in Proc. 12th IEEE Int. Conf. Netw. Protocols, 2004, pp. 174–178.intrusion detection system," ACMTrans. Embed. Comput. Syst., vol. 3, pp. 614–633, 2004.

[9] R. S. Boyer and J. S. Moore, "A fast string searching algorithm,"Commun. ACM, vol. 20, pp. 762–772, 1977.

[10] V. Aho and M. J. Corasick, "Efficient string matching: An aid to bibliographic search," Commun. ACM, vol. 18, pp. 333–340, 1975