Exploiting Temporal Locality in Network Traffic Using Commodity Multi-cores

Govind Sreekar Shenoy*, Jordi Tubella* and Antonio González†
*Department of Computer Architecture, Universitat Politècnica de Catalunya, Barcelona, Spain.
†Intel Barcelona Research Center, Barcelona, Spain.
Email: {*govind,jordit*}*@ac.upc.edu, antonio.gonzalez@intel.com*

***Abstract*—Network traffic has traditionally exhibited temporal locality in the header field of packets. Such locality is intuitive and is very well studied over the years. In this work we study temporal locality in the packet payload. Temporal locality can also be viewed as redundancy and we observe significant redundancy in the packet payload.**

We investigate mechanisms to exploit temporal locality in a networking application and choose Intrusion Detection Systems (IDS) as a case study. An IDS like the popular Snort[4] operates by scanning packet payload for known attack strings. It first builds a Finite State Machine (FSM) from a database of attack strings, and traverses this FSM using bytes from the packet payload. So we propose a redundancy-aware FSM traversal that skips the processing of redundant bytes. We have deployed our redundancy-aware FSM traversal in Snort, and we observe important performance benefits.

I. Introduction

The constant evolution of Internet has demanded an increased functionality from the network-layer, the layer in the network stack where routers operate. Network-layer applications like packet forwarding are computationally very intensive. So network-layer applications exploit temporal locality to speed up execution time. Locality in the header of packets is well known and is exploited using application specific caches like in packet forwarding. While locality in header fields is intuitive and well studied, to the best of our knowledge there have not been significant studies on locality in the packet payload. Recently, however, there have been works [2] investigating temporal locality in the packet payload. They observe significant locality and use it for efficient transmission of data. Temporal locality can also be viewed as redundancy, and they propose mechanisms to compress the transmission of these redundant bytes. This work, on the other hand, focus on exploiting redundancy to accelerate the processing time of an Intrusion Detection System (IDS).

IDS is an example of a network-layer application that detects attacks by inspecting packet payload for known attack strings. So pattern matching of attack strings is performed on the payload bytes. This is performance critical since the payload size can be huge, and also due to the huge database of attack strings. The broad approach we take in this work is to skip the redundant processing of redundant payload bytes in the packet. So we propose mechanisms to dynamically identify redundant bytes in the payload. While this work focusses on accelerating IDS processing, the approach can be extensible to other payload processing applications. So our work on IDS can be viewed as a demonstration to exploit redundancy in payload processing.

II. Background

An IDS like Snort uses the Aho-Corasick algorithm[1] to detect attack strings in the packet payload. This algorithm first constructs a FSM from attack strings and traverses the FSM with payload bytes. Further, we observe that up-to 62% of the execution time of Snort IDS is spent in the FSM traversal. So the FSM traversal is critical to the execution time of an IDS.

The main advantage of the Aho-Corasick algorithm is that it guarantees linear-time search irrespective of the number of strings. While linear-time search is a powerful advantage, however there are performance/memory issues associated with the algorithm. Earlier works in this area, have broadly either investigated mechanisms to compactly store the FSM, and/or accelerate its traversal. So earlier works propose a compact FSM state representation or explore optimizing the failure edges of the Aho-Corasick FSM. In order to accelerate the FSM traversal, earlier works investigate using a multi-byte FSM traversal or explore speculation and parallelization techniques. The redundancy-aware FSM traversal proposed in this work is orthogonal and complements all these mechanisms.

IDSs also commonly use regular expressions to specify attack strings. Regular expressions are converted either to Nondeterministic Finite Automata (NFA) or Deterministic Finite Automata (DFA). These automatas are very similar to the Aho-Corasick FSM, and in fact the Aho-Corasick FSM can even be viewed as a DFA. In this work we concentrate on the Aho-Corasick FSM since it dominates the execution time of Snort. However, it is important to note that our proposed redundancy-aware FSM traversal is equally applicable to NFAs and DFAs.

III. Our Contribution

We aim to eliminate the redundant processing of payload bytes, so we focus on skipping redundant FSM traversals. A redundancy table is used for this purpose, and is indexed using a chunk of payload bytes and the current FSM state. The chunk of bytes forms the unit of redundancy and we term Redundancy Length (RL) as the length of this chunk. Note that payload bytes alone are not sufficient, the FSM state is also needed to guarantee correctness in the FSM traversal. The redundancy table is implemented in software using the Boost

Unordered library. In our evaluation, we observe that the table look-up incurs a significant overhead. So, we perform table look-up at an interval equal to RL. Figure 1 shows an example of our redundancy-aware FSM mechanism with $RL = 3$. In this example when the payload bytes (**s h i**) are encountered the second time, their FSM traversal is skipped.

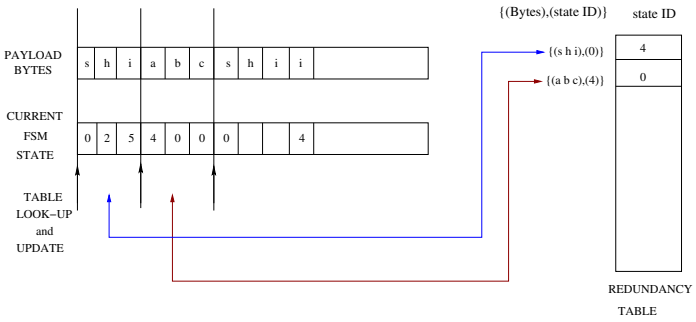

Fig. 1: Redundancy-aware FSM Traversal

In Figure 1 table look-ups and updates are performed in tandem with the FSM traversal. However, these are overheads that add to the execution time of the FSM traversal. If we examine table operations in more detail, we observe that only the table look-up is needed with the regular FSM traversal. The table update can be delayed after the FSM traversal. Furthermore, the redundancy table update operation is also completely independent of any IDS processing. So table updates are performed simultaneously with other IDS functionalities. This can also be viewed as two parallel threads. The Snort thread which is also the main thread, performs the regular IDS processing including the redundancy-aware FSM traversal. On the other hand, the Redundancy thread identifies and captures the redundancy by updating the redundancy table. The threads need synchronization since there are data-structures like the redundancy table shared between them.

IV. Results

We have implemented our proposed mechanism in the Snort March-2011 release and evaluated it on an Intel Core i3. We compare the execution time of our redundancy-aware FSM traversal with the FSM traversal implemented in the Snort release. The execution time is measured using the POSIX clock_gettime() clock. We report the execution time on a per payload byte basis. We use a variety of traces for our evaluation, but due to space constraint we report the result from the MIT[3] Week-2 1999 data-set.

Figure 2 shows the execution time of the redundancy-aware mechanism for varying RL and table sizes. The table size has been varied from 40K entries to 120K entries. In this Figure we also compare the performance of our proposal with the base Snort implementation (referred to as Baseline).

The execution time shows an interesting trend, namely, that an increase in RL speeds up the pattern matching. Furthermore, $RL = 8$ is the most clock consuming configuration. It is also interesting to note that $RL = 8$ also captures a very high redundancy (52% redundancy captured). This very unusual behaviour of a very high redundancy with a very low performance is due to the overhead in table look-up. We also observe that this overhead gets amortized on increasing RL, and for large RLs the look-up overhead is minimal. So for $256 \leq RL \leq 1280$, the redundancy-aware FSM traversal outperforms the standard Snort FSM traversal. In particular, we observe that $RL = 1280$ is the best performing configuration and provides a 15% speed up in execution time. This performance gain is due to the large chunk of bytes being skipped with relatively lesser table look-up overhead. Note that for $RL > 1280$ very few bytes are skipped of FSM traversal, and so there is no performance gain.

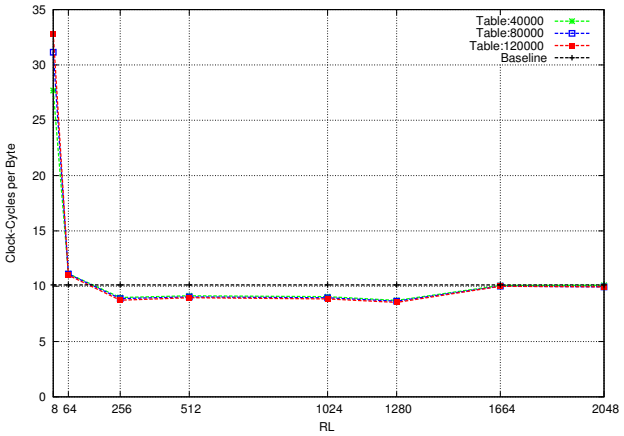

Fig. 2: Execution Time Comparison

V. Conclusion

In this work, we have investigated exploiting temporal locality in the packet payload. IDS is an example of payload processing application which is critically dependent on the processing of payload bytes. An extension of this work is to study a redundancy-aware mechanism in the IPSec algorithms used by VPNs. Note that IPSec like IDS also performs payload processing and so payload locality can be exploited in it.

VI. Acknowledgements

This work has been supported by the Generalitat de Cataluyna under grant 2009SGR-1250, the Spanish Ministry of Science and Innovation under grants TIN2007-61763 and TIN 2010-18368 and Intel Corporation.

References

[1] A. V. Aho and M. J. Corasick. Efficient String Matching: An Aid to Bibliographic Search. *Communications of the ACM*, 1975.

[2] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee. Redundancy in Network Traffic: Findings and Implications. *In Proceedings of the 11th International Conference on Measurement and Modeling of Computer Systems*, 2009.

[3] MIT Lincoln Labs, DARPA Intrusion Detection Evaluation. http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/.

[4] M. Roesch. Snort - Lightweight Intrusion Detection for Networks. *In Proceedings of the 13th USENIX conference on System Administration*, 1999.