SANS: A Scalable Architecture for Network Intrusion Prevention with Stateful Frontend

Fei He[1], Yaxuan Qi[1], Yibo Xue[1,2] and Jun Li[1,2]

[1]Research Institute of Information Technology, Tsinghua University, Beijing, China

[2]Tsinghua National Lab for Information Science and Technology, Beijing, China

hefei06@mails.tsinghua.edu.cn, {yaxuan, yiboxue, junl}@tsinghua.edu.cn

ABSTRACT

Inline stateful and deep inspection for intrusion prevention is becoming more challenging due to the increase in both the volume of network traffic and the complexity of the analysis requirements. In this work, we pursue a novel architectural approach, named SANS, which takes both the advantage of new generation network processors for packet-header-based processing and the advantage of commodity x86 platforms for packet payload data processing. A session table scheme is designed for the stateful frontend in SANS to achieve wire speed inline processing.

Categories and Subject Descriptors

C.2.0 [**Security and Protection**]

General Terms

Security

Keywords

Stateful Inspection, Deep Inspection, Network Processors, Intrusion Prevention, Session Table.

1. INTRODUCTION

The difficulties of building high performance network intrusion prevention systems (NIPS) stem mainly from the fact that NIPS needs to analyze not only packet headers but also packet payload, and that the operations to be performed on each packet is growing with the continuous evolving of network attacks. NIPS is both memory-usage and memory-bandwidth intensive.

The two dimensional growth in both data rate and analysis complexity (number of attack signatures) is outpacing the speed at which memory technologies advance. Therefore, high-performance NIPSes are now commonly implemented using cluster-based or chassis-based architectures, which can scale up with the growth by adding additional backend detection engines. However, the traditional cluster-based architectures using simple frontend have some limitations. First, it is complicated to implement inline intrusion prevention in this kind of architectures. Second, since some correlation information (e.g. to detect port scanning, flows of one host are correlated, and should not be dispatched to different backend nodes) is lost at the time of traffic distributing, traditional cluster-base NIPS needs an inter-detection-engine communication scheme which brings in a substantial amount of overhead. Even so, in order to be deployed on the backend nodes, existing intrusion prevention software, such as Snort and Bro, needs to be modified to support coordination.

ANCS'09, October 19-20, 2009, Princeton, New Jersey, USA.

In this work we pursue a different architectural approach, which uses network processors (NP) as a stateful inline device, and several general-purpose CPU platforms as backend nodes to perform deep inspection. The goal is to utilize the advantage of the new generation NPs in packet-header-based processing to reduce the workload and complexity of backend detection engines. We argue that a stateful frontend maintaining per-flow state offers not only the feasibility of the inline intrusion prevention functionality, but also provide several other architectural advantages to be described in the following section.

Session table is the key component that determines the throughput of the stateful frontend. In the multi-core network processor base environments, traditional flow table using chained hashing is less appealing as its use of dynamic memory allocation and indirection causes poor cache performance with mutual exclusive operations degrading performance furthermore. We design a session table scheme by combining the chained hashing with a small cache of the forwarding decision of active flows. This scheme achieves a 33~86 percent throughput improvement, while an increase of only 2.5-20MB of memory for one million concurrent sessions.

2. THE SANS ARCHITECTURE

The SANS based NIPS can be divided into two types of components: The stateful frontend nodes and the backend detection nodes. The frontend nodes act as an inline forwarding engine, as well as, a traffic dispatcher for a set of backend detection engines. The backend nodes, acting as the network intrusion detection engines, perform traffic analysis and update follow-state in the frontend nodes via fabric channel.

The main architectural choice is that the frontend needs to be stateful and active, rather than acting as a passive load balancing component. The stateful packet forwarding engine is the key component in SANS architecture. All incoming traffic that arrives at the system enters the forwarding engine. When a packet enters the forwarding engine, the engine looks up the session table with the usual 5-tuple of source and destination IP address, source and destination ports, and transport protocol (TCP or UDP). The packet is processed according to an associated action stored in the corresponding session state. The action includes forward, drop, and inspect. The inspect action can even be extend to a sequence of detections, called detection chain. If the packet is the first packet of a flow, a series of operations, including policy lookup, route lookup, correlation analysis, initial load balancing decision,

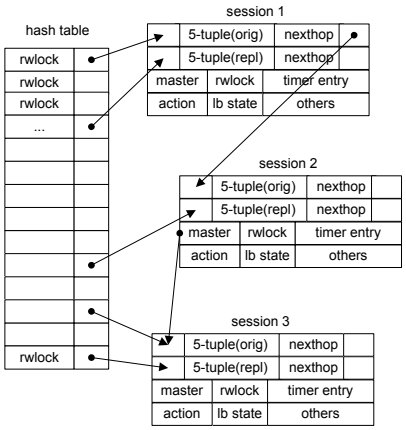

Fig1. Data Structure of BST

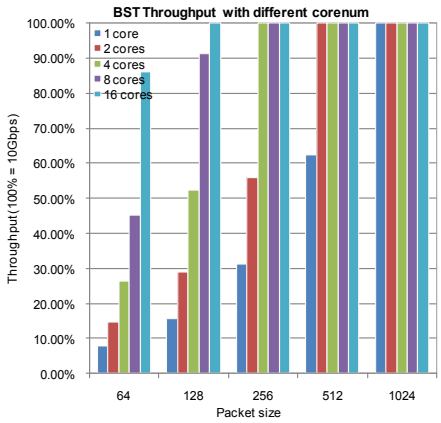

Fig2. Throughput of BST

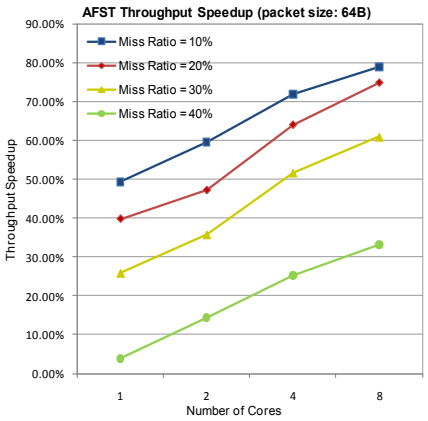

Fig3. AFST Throughput Speedup over BST

need to be performed. The results of all these operations are recorded in the session state of the corresponding session. After examining a packet, the detection engine will update the corresponding session state based on the detection result.

Session table is the most important component for achieving high performance in frontend, as it enables fine-grained, per-session decision-making instead of per-packet decision-making. Based on per-flow state management, several functions and optimizations can be implemented on the frontend. These optimizations include: 1) correlation analysis, 2) policy enforcement, 3) pre-filtering [1] and normalization [2], and 4) adaptive load balancing. These optimizations improve NIPS overall performance, reduce the complexity of backend nodes while increase the flexibility of backend nodes.

Unlike traditional cluster-based architecture, existing open source system such as Snort and Bro can be deployed on the backend detection engines with minor modification. In most of the traditional cluster-base NIPSes, simple hash-based load distributes packets with malicious payload over several different detection engines, and thus makes it difficult and computationally expensive to merge results from these multiple engines and recognize the attack. Given the stateful frontend traffic dispatcher in SANS, most of the correlation analysis is done at the frontend. Furthermore, a sophisticated load balancing scheme can support not only homogeneous backend detection engines, but heterogeneous engines. Thus, the configurations of backend detection engines are much more flexible.

3. THE STATEFUL FRONTEND DESIGN

Hash tables are often attractive implementations for session table since they result in constant-time, *O(1)*, query, insert and delete operations. We present the basic form of session table design called Basic Session Table (BST) in Figure 1. A session entry contains two parts: the first part (called wing) for direction-related states and the second part containing session-level information. A wing, which consists of 5-tuple of one direction of the session and direction-related information such as routing information, is the item actually inserted into the hash table. Each session entry has a master session pointer, pointing to its correlated session.

The BST design using chained hashing allows the load balancing module, correlation analysis module, and other modules to access correlated flows through one session table lookup. However, its use of dynamic memory allocation causes poor cache performance. Moreover, even through readers-writers locks are employed to achieve fine-grained synchronization, mutual exclusions add overhead in a multi-core environment. Most packets entering the frontend require only a lookup operation on the session table while only several packets at the beginning of a flow need to update the session table. Based on this observation, a session table scheme, called Active Flow Buffered Session Table (AFST), is designed to optimize for common cases. AFST employs a compact hash table without collision resolving mechanism as a buffer of active flows, i.e. Active Flow Buffer (AFB). The entry stored in the AFB only consists of 5-tuple, route entry and action which consume 20 bytes. When a packet does not match an entry in the AFB, a lookup is continued in the BST. Experiments using LBNL enterprise traces [3] show that when the load factors (size of AFB divided by number of concurrent flows) are 1, 0.5, 0.25, and 0.125, the miss ratios of AFB are 7.85%, 11.60%, 17.02%, and 24.84% respectively.

The performance of the BST and AFST is evaluated on a Cavium Octeon 5860 multi-core network processor, which has 16 MIPS cores, 2 MB L2 cache shared by the 16 cores, and 4 GB DDR2 memory. Figure 2 shows that the BST reaches 7.8 Gpbs throughput for 64-Byte packets. Figure 3 shows that the AFST achieves about 33~86 percent speedup over the BST using 8 cores for the AFB miss ratio from 40% to 10%. The extra memory needed by the AFB for a session table supporting one million concurrent session is only 2.5 MB for a load factor of 0.125, 20 MB for a load factor of 1.

4. ACKNOWLEDGMENTS

This work was supported by National High-Tech R&D 863 Program of China under grant No. 2007AA01Z468.

5. REFERENCES

[1] Sourdis, I., Dimopoulos, V., Pnevmatikatos, D. and Vassiliadis, S., 2006. Packet Pre-filtering for Network Intrusion Detection. ANCS 2006.

[2] Handley, M., Paxson, V. and Kreibich, C. 2001. Network intrusion detection: evasion, traffic normalization, and end-to-end protocol semantics. SSYM 2001.

[3] LBNL/ICSI Enterprise Tracing Project, http://www.icir.org/enterprise-tracing/index.html .