Chapter 1

Introduction

As the Internet grows at a very rapid pace, so does the incidence of attack events and documented unlawful intrusions. Recipes for these attacks are readily available, often in a ready-to-run format, and recent incidents with global dimensions are clear evidence that the average computer criminal is much less sophisticated than once was believed. Security personnel must respond quickly and proactively to the increasing number and severity of threats from viruses, worms and intrusion attacks. The Network Intrusion Detection Systems (NIDSes) are designed to identify attacks against networks or a host that are invisible to firewalls, thus providing an additional layer of security. From 2002, the most important feature of an NIDS is to work in inline mode, where all packets must pass through the device. The problem with working inline is that there is always the potential to affect the performance and reliability of the rest of the network. Another issue is broad and accurate attack coverage. False positive alarms result in senseless waste in human resource while false negative makes the network be compromised.

Generally two main methods are used for intrusion detection, namely Pattern Matching and Statistical Analysis [39]. The former method applies a static set of patterns and alerts to traffic sequences with known signatures. Meanwhile, the latter method detects anomalous events statistically by gathering protocol header information and comparing this traffic to known attacks, as well as by sensing anomalies. Modern NIDSes usually supports both methods. However, the performance of the NIDSes has been shown to be dominated by the speed of the string matching algorithms used to compare packets with signatures. An NIDS must employ an efficient string matching algorithm since an under-performing passive

system drops many packets and may miss many attacks, while an under-performing inline system creates a bottleneck for network performance [11].

This dissertation presents three pattern search algorithms that conduct matching sets of patterns in parallel. Generally speaking, most signatures in NIDSes are ASCII codes while network traffic is composed in binary data and signature matches rarely happen in real-world traffic. The more bytes we can skip during searching signatures in packets, the more performance we gain. The first pattern search algorithm, *FNP* [28], is a Network Processor-based algorithm and utilizes the hash engine of the Network Processor to achieve high performance. The rule memory in this design is too large to fit into local cache so that this algorithm is quite suitable for Network Processors. Network Processors usually lack of cache memory so that accessing main memory are quite expensive in this environment. Unfortunately, pattern matching algorithms usually need to access memory quite frequently. For example, the Aho-Corasick algorithm [1] needs to access main memory for every single byte in packet payload. The proposed *FNP* [28] algorithm outperforms other alternatives in this way so that its performance is quite good in our test.

The second algorithm we proposed is the FNP^2 [30] algorithm. This algorithm is not for Network Processor platform specifically but a general software-based solution. The FNP^2 algorithm is modified from Wu-Manber algorithm (*MWM*) [57] and needs less memory access than *MWM*, especially when the size of shortest pattern is small. According to current snort ruleset, there're a lot of patterns whose size is less then three, therefore FNP^2 is quite suitable to be implemented into a software-based NIDS. We implement FNP^2 into a Network Processor platform also, and its performance is better than other competitors according to our experiments.

Except to these two software-based pattern matching algorithm, we also proposed a *FNP*-like TCAM-based searching algorithm named *FTSE*. With an ASIC/FPGA

running with our algorithm, a 2.25Mbit TCAM, and a DDR SDRAM which cost less than fifty US dollars in total, we can achieve very high throughput in multiple gigabit. The FTSE can process multiple packets in the same time and by this way the TCAM latency could be hidden also. These characteristics make FTSE preferable to high-end Network Processors to implement gigabit-level NIDS.

On the other hand, a sophisticated TCP processing engine is a prerequisite since the attackers can use ambiguities in network protocol specifications to deceive network security systems [16, 31, 39, 40]. This engine makes sure the NIDS sees the same thing as the end system and prevents from ambiguities. In this TCP scrubbing engine, we check the integrity of the TCP headers, tracks the TCP state transitions, and reassembles TCP segments into meaningful data stream with a minimum cost. We implement this engine and test this engine with the same methodologies as in the evasion test of the most two reputable NIDS certificate (NSS [45] and OSEC [46]).

However, such a TCP processing engine itself might be vulnerable to Denial of Service (DoS) attacks, more specifically, the SYN Flood attacks. Since the TCP processing engine needs to allocate memory spaces for monitoring the whole lifetime of TCP connections, it could be exhausted in memory resources under SYN Flood attack. It has been shown that many security systems are vulnerable to SYN Flood attacks themselves [40], and over 90% DoS attacks are SYN Flooding [53]. This dissertation also presents an efficient mechanism, FSS filter, which can significantly mitigate the damage and it can work in conjunction with other methods also. The FSS filter can block and mitigate SYN Flood attacks, and it can co-work with other methods like Semi-Transparent [25] method as well. In our experiments, we collected 18 famous SYN Flood attacks and FSS filter can block they attacks successfully and immediately.

The rest of this dissertation is organized as follows, Chapter 2 introduces the SYN

Flood problem and several defending mechanisms and FSS filter as well. Next, Chapter 3 explains why a sophisticated TCP processing engine is necessary, and presents several mechanisms. Chapter 4 and Chapter 5 discuss two pattern matching algorithms for Network Processor platforms, respectively. Chapter 6 presents a novel hardware architecture to match patterns in multiple gigabit speed. Finally some conclusions are given in Chapter 7.