Network intrusion prevention on the network card

Herbert Bos[†]*, Li Xu*[*]*, Kees van Reeuwijk*[†]*, Mihai Cristea*[*]*, Kaiming Huang*[‡]
[†]*Vrije Universiteit Amsterdam and* [*]*Leiden Universiteit, Netherlands,* [‡]*Xiamen University, PRC*
{*herbertb,reeuwijk*}*@cs.vu.nl,* {*lxu,cristea*}*@liacs.nl, kmhuang@xmu.edu.cn*

Abstract

CardGuard **is a signature detection system for intrusion prevention that scans the entire payload of packets for suspicious patterns and is implemented in software on a network card. The hardware that is used on the card consists of an Intel IXP and various memories. One card can be used to protect either a single host, or a small group of machines connected to a switch.** *CardGuard* **is non-intrusive in the sense that no cycles of the host CPUs are used for signature detection and the system still operates at realistic link rates. It currently employs a parallelised version of an efficient string matching algorithm at the lowest level of the processing hierarchy. It is used for detecting the signatures corresponding to intrusion attempts in the packets' payloads. A new version supporting an advanced regular expression algorithm in the microengines of an Intel IXP network processor is under development. For TCP flows,** *CardGuard* **first reconstructs the TCP byte stream before applying the pattern matching engine. The system exploits the memory hierarchy of the network card by storing frequently needed data in fast on-chip memory, while data that is rarely accessed is kept in slower off-chip memory.**

1 Introduction

Intrusion detection and prevention systems (IDS/IPS) are increasingly relied upon to protect network and computing resources from attempts to gain unauthorised access, e.g., by means of worms, viruses or Trojans. To protect computing resources on fast connections, it is often desirable to scan packet payloads at line rate. However, scanning traffic for the occurrence of attack signatures is a challenging task even with today's networks. Moreover, as the growth of link speed is sometimes said to exceed Moore's law, the problem is likely to get worse rather than better in the future. Worms especially are difficult to stop manually as they are self-replicating and may spread fast. For example, the Slammer worm managed to infect 90% of all vulnerable hosts on the Internet in just 10 minutes [5].

Rather than performing signature scanning at a centralised firewall or on the end-host's CPU, we explore the feasibility of implementing a complete intrusion prevention system (IPS) in software on the network card. The notion of a distributed firewall, first proposed by Bellovin in 1999, has become fairly popular in recent years [1]. However, most of these systems do not implement payload inspection at all. Recently, Clark et al. proposed to use FPGAs for signature detection [4]. The disadvantage of FPGAs and other hardware solutions is that they are complex to modify (e.g., to modify the detection algorithm).

The contribution of our project is that we explore for the first time one of the extremes in the design space of an IPS, whereby the entire IPS is implemented in software on the network card. The card could either be a network card that is plugged directly in an end-user's PC, or a card sitting in a switch/router that is connected to the end-user's PC. Initial results obtained with a prototype implementation are promising.

2 Distributing the firewall

Most current approaches to IDS/IPS involve a high-performance firewall/IDS at the edge of the network. All internal nodes are assumed to be safe and all external nodes are considered suspicious. The firewall closes all but a few ports and in an advanced system may even scan individually pack-

Presented at the IXA Education Summit 2005, Hudson, MA, USA, September 2005.

ets for the occurrence of attack patterns. Compared to a distributed firewall, this approach has a number of drawbacks. First, it does not protect internal nodes from attacks originating within the intranet. Once an internal node has been compromised, by whatever means, all nodes in the intranet are at risk. Second, the volume of traffic is very large, which makes it rather hard to scan every single packet and manage state for every flow. Often such systems limit themselves to per-packet analysis rather than scanning the full (reconstructed) TCP byte stream. However, attacks may span a number of packets each of which may be harmless in and of itself. Hence, flow reconstruction is a requirement for reliable signature detection in the payload. A fourth drawback is that centralised firewalls tend to close all ports except a select few, such as those used for webtraffic. As a consequence, we observe that all sorts of new protocols are implemented on top of port 80, defeating the purpose.

So, one may ask, why not implement the IPS on the host CPU? There are several reasons why a software solution on the host processor might be problematic. First, scanning every byte of every network packet for complex attack signatures is expensive and presents a serious load on the CPU (some exerimental results are presented in [3]). Second, the software on the host processor is much more under control of end-users than a network card. This is certainly true for a card that is placed in a switch, physically away from the end user's PC. In essence, this problem has less to do with technology and more with policies: administrators are not too keen on trusting an end-user's machine.

3 *CardGuard*

Our IPS, known as *CardGuard*, is intended to protect a small set of hosts connected to a switch. Throughout this project, our goal is to make the IPS an inexpensive device with an eye on making it competitive with large firewalls. At the same time, the IPS should be fast enough to handle realistic loads. say a few hundred Mbps for individual users. Note that we aim to protect against unwanted content (e.g., intrusion attempts, or spam) and not against denial of service attacks. Finally, we focus on the computationally hard problem of pattern matching, rather than the less compute intensive problem of header inspection and anomaly detection which is often already found in commercial network equipment.

The IPS is in the process of being incorporated in the FFPF framework [2]. The advantage of FFPF is that it allows for very simple configuration and extension by administrators. For instance, administrators may place the IPS in a graph of traffic processing functions (such as filters, samplers, NATs, statistical functions, etc.) and press an instantiate button to install the customised packet processing at a specific site. FFPF automatically maps the functions on the different levels of the processing hierarchy: microengines, XScale, or host kernel. In other words, the IPS will be 'just' another module that can be activated or deactivated to protect hosts and servers according to company policy.

We have already implemented a prototype based on the IXP1200 which shows very promising results [3]. The prototype is based on an obsolete IXP1200 network processor and achieves a throughput of approx. 600 Mbps for UDP and 100 Mbps for TCP with full stream reconstruction and while testing thousands of rules. These results are achieved under worst-case conditions whereby *every* single packet is scanned for thousands of rules. In realistic scenarios this is never the case. After all, if there is no known vulnerability for application X, there is no reason to subject traffic to and from X to scans that correspond to rules for different applications.

However, the prototype uses the rather simplistic Aho-Corasick (AC) string matching algorithm. AC is fast, but too limited for intrusion detection purposes, because for every rule it spots only exact matches of single strings. In practice, we want to be able to detect regular expressions and multi-component strings. For this purpose, we have implemented a new regular expression language for intrusion detection that generates an optimised deterministic finite automaton to match much more complex signatures. The language is tailored to network packets and includes knowledge about protocols in separate header files. We are currently working on its implementation on the IXP2400. We have also made initial steps to implement it on an IXP2800 with an eye on providing a solution that is intermediate between a 'fully centralised' and a 'fully decentralised' firewall.

4 Exploiting locality of reference

Scanning every byte is very expensive (especially if we also have to reconstruct TCP byte streams), but the problem becomes especially daunting if we consider that there are many thousands of rules that need to be checked. Obviously, there is no time to

scan every packet thousands of times. Instead, we employ deterministic finite automata (DFA), constructed offline, to check all rules at once, one byte at a time.

For instance, the well-known Aho-Corasick algorithm incurs one state transition in the DFA for every one byte that is read from the packet. Assume for a very trivial example that we start in state 0 and read the following characters from the packet: 'w', 'o', 'r', 'm', and incur state transitions as follows: $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4$. Some states (e.g., state 4 in the example) are special and are marked as end states. End states indicate that one or more rules matched. For instance, if rule 1 is triggered by the occurrence of pattern 'worm' and rule 2 by pattern 'another worm', both rules are triggered if the latter pattern is encountered. End states trigger alerts and in the case of an IPS also lead to a connection drop.

However, while we may be able to match thousands of rules at once in the above manner, the large rule set is problematic for another reason: memory. Indeed, fast memory is the bottleneck for almost any high-speed network application. We summarise the problem as follows: fast (on-chip) memory is not large enough and large (off-chip) memory is not fast enough. Any access to off-chip memory is extremely costly and an implementation of *CardGuard* with all rules in off-chip SRAM suffered a collapse in performance, as every packet incurred a number of memory accesses for every state transition.

We observed that the problem is similar to the infamous 'memory gap' between the host processor and main memory in normal computing, which is solved by caching. For this reason we investigated to what extent the accesses to the DFAs exhibited locality of reference. By evaluating *CardGuard*'s behaviour for thousands of snort rules with real user traffic we found that this is indeed the case (some results for the AC algorithm can be found at: `http://www.cs.vu.nl/~herbertb/papers/ac_locality/`). Almost all of the memory accesses to the DFA occurred near the root of the DFA (close to state 0), while only a few visits were made to deeper levels. As a result, we were able to spread the DFA over multiple levels in our memory hierarchy. Indeed, the top levels were placed as code in the instruction store of the individual microengines to guarantee very fast access.

The code that is generated by our new intrusion detection language is similarly based on DFAs. While we are still evaluating the locality of reference of the code that is generated by our compiler, we expect similar results. In essence, we solve the 'memory gap' by employing a static cache, i.e., a cache without replacement.

5 Conclusion

CardGuard shows that the implementation of a complete IPS in software on the network card is feasible and offers some distinct advantages over both centralised solutions and solutions where detection is performed on the end host itself. A fully functional prototype has been implemented and we are close to finishing a more complex implementation on the IXP2400. More importantly, *CardGuard* explores for the first time one of the extremes in the design space for building and placing intrusion prevention systems. Intrusion detection and prevention is a very active field and a solution would be hugely beneficial to many stakeholders. For this reason alone it is important that the design space is explored to find the most optimal solution.

Acknowledgements. We would like to thank Intel for donating the IXP1200 cards. Part of this work was started in the EU FP6 Lobster project.

References

[1] Steven M. Bellovin. Distributed firewalls. *Usenix ;login:, Special issue on Security*, pages 37–39, November 1999.

[2] Herbert Bos, Willem de Bruijn, Mihai Cristea, Trung Nguyen, and Georgios Portokalidis. FFPF: Fairly Fast Packet Filters. In *Proceedings of OSDI'04*, San Francisco, CA, December 2004.

[3] Herbert Bos and Kaiming Huang. Towards software-based signature detection for intrusion prevention on the network card. In *Proceedings of Eighth International Symposium on Recent Advances in Intrusion Detection (RAID2005)*, Seattle, WA, September 2005.

[4] Chris Clark, Wenke Lee, David Schimmel, Didier Contis, Mohamed Koné, and Ashley Thomas. A hardware platform for network intrusion detection and prevention. In *Third Workshop on Network Processors and Applications*, Madrid, Spain, February 2004.

[5] David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, and Nicholas Weaver. The spread of the Sapphire/Slammer worm, technical report. Technical report, CAIDA, 2003. `http://www.caida.org/outreach/papers/2003/sapphire/`.