Detecting and Blocking Network Attacks at Ultra High Speeds
DE-FG02-04ER25638
Final Report

PI: Dr. Vern Paxson

November 29, 2010

1 Project Description

Stateful, in-depth, in-line traffic analysis for intrusion detection and prevention has grown increasingly more difficult as the data rates of modern networks rise. One point in the design space for high-performance network analysis—pursued by a number of commercial products—is the use of sophisticated custom hardware. For very high-speed processing, such systems often cast the entire analysis process in ASICs.

This project pursued a different architectural approach, which we term *Shunting*. Shunting marries a conceptually quite simple hardware device with an Intrusion Prevention System (IPS) running on commodity PC hardware. The overall design goal is was to keep the hardware both cheap and readily scalable to future higher speeds, yet also retain the unparalleled flexibility that running the main IPS analysis in a full general-computing environment provides.

The Shunting architecture we developed uses a simple in-line hardware element that maintains several large state tables indexed by packet header fields, including IP/TCP flags, source and destination IP addresses, and connection tuples. The tables yield decision values the element makes on a packet-by-packet basis: forward the packet, drop it, or divert ("shunt") it *through* the IPS (the default). By manipulating table entries, the IPS can, on a fine-grained basis: *(i)* specify the traffic it wishes to examine, *(ii)* directly block malicious traffic, and *(iii)* "cut through" traffic streams once it has had an opportunity to "vet" them, or *(iv)* skip over large items within a stream before proceeding to further analyze it.

For the Shunting architecture to yield benefits, it needs to operate in an environment for which the monitored network traffic has the property that—after proper vetting—much of it can be safely skipped. This property does *not* universally hold. For example, if a bank needs to examine all Web traffic involving its servers for regulatory compliance, then a monitor in front of one of the bank's server farms cannot safely omit a subset of the traffic from analysis. In this environment, Shunting cannot realize its main performance benefits, and the monitoring task likely calls for using custom hardware instead.

However, in many other environments we find Shunting holds promise for delivering major performance gains. This arises due to the the widely documented "heavy tail" nature of most forms of network traffic [10, 11, 2, 17, 16, 1], which we might express as "a few of the connections carry just about all the bytes." The key additional insight is ". . . and very often for these few large connections, the very *beginning* of the connection contains nearly all the information of interest from a security analysis perspective." We argue that this second claim holds because it is at the beginning of connections that authentication exchanges occur, data or file names and types are specified, request and reply status codes conveyed, and encryption is negotiated. Once these occur, we have seen most of the interesting facets of the dialog. Certainly the remainder of the connection might also yield some grist for analysis, but this is *generally less likely*, and thus if we want to lower analysis load at as small a loss as possible of information *relevant*

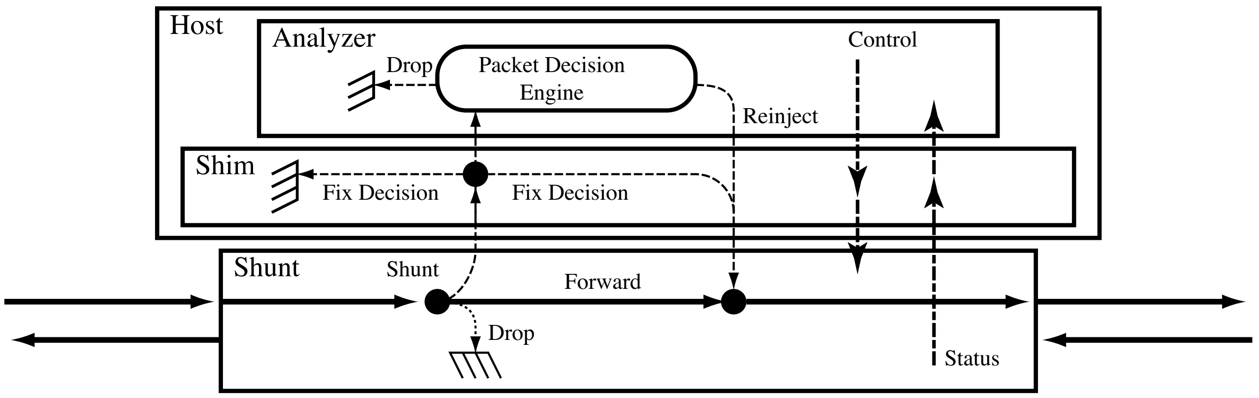

Figure 1: Shunting Main Architecture

to security analysis, we might best do so by skipping the bulk of large connections. In a different context, the “Time Machine” work by Kornexl and colleagues likewise shows that in some environments we can realize major reductions in the volume of network traffic processed, by limiting the processing to the first 10–20 KB of each connection [8].

As a concrete example, consider an IPS that monitors SSH traffic. When a new SSH connection arrives and the Shunt fails to find an entry for it in any of its tables (per-address, per-port, per-connection), it executes the default action of diverting the connection through the IPS. The IPS analyzes the beginning of the connection in this fashion. As long as it is satisified with the dialog, it reinjects the packets forwarded to it so that the connection can continue. If the connection successfully negotiates encryption, the IPS can no longer profitably analyze it, so it downloads a per-connection table entry to the Shunt specifying that the action for the connection in the future is “forward.” For heavy-tailed connections, this means a very large majority of the connection’s packets will now pass through the Shunt device without burdening the IPS with any further analysis load. On the other hand, if the IPS is dissatisfied with some element of the initial dialog, it downloads a “drop” entry to terminate the connection. Note that by providing for reinjection, we can promote an intrusion *detection* system into an intrusion *prevention* system, one that does not merely detect attacks but can block them before they complete. Reinjection also allows the IPS to *normalize* traffic [6] to remove ambiguities that attackers can leverage to evade the IPS [13]. Finally, if the IPS is unable to resolve whether the connection can progress without further analysis, it simply leaves the Shunt’s tables unmodified and continues to receive the connection’s packets due to the Shunt’s default action.

2 Summary of Project Results

Figure 1 shows the final architecture we developed for enabling use of inexpensive, highly flexible commodity PCs for inline packet processing. A *Shunt* consists of two elements, a software packet processor (the *shunt engine*) and a hardware forwarding element (the *shunt device*). The shunt engine is the decision mechanism, and is itself composed of two parts: an analyzer that processes the traffic (for example, an IPS), and a thin layer that mediates between the analyzer and the shunt device (the *Shim*).

We can think of the shunt device as a smart NIC with which the shunt engine communicates over a bus; but we can also realize it as a physically separate device with which the shunt device communicates using a network link such as Gbps Ethernet. The Shim isolates the IPS from the nature of this interconnect.

When a packet arrives, the device chooses from one of three possibilities: (a) forward the packet to the opposite interface (thick, solid line), (b) drop it (thin, dashed line), or (c) divert (*shunt*) it to the analyzer (thin, dotted line). For packets diverted to the shunt engine, the analyzer makes another decision regarding the packet's fate: (c.1) inject the packet back to the network interface, or (c.2) drop it. It may optionally at this point also update the shunt device's tables to offload similar decisions in the future.

We implemented the Shunt on top of the NetFPGA2 [9] research and education platform. This platform contains four Gbps Ethernets, two 2 MB SRAMs, and a Virtex 2 Pro 30 FPGA, all located on a single PCI card which fits inside a standard host. We began by modifying an existing design, a 4-port Ethernet NIC that used only one of the SRAMs as a buffer, to create *RNET*, a framework for in-place packet manipulation and routing. The RNET framework provides a shim between each receiving MAC and the main controller. Each shim buffers one packet at a time, and can manipulate the packet before routing it to any output MAC or to the host.

We then built the Shunt using the RNET infrastructure. The design centers around two primary caches: a connection cache of 2^{16} entries and an IP address cache of 2^{16} entries. The connection cache uses multilocation associativity, a variant of a design by Song et al [14], where two separate hash functions are used to provide two different possible locations for each entry, to allow the host to move entries to free up space. The IP address cache is a multilocation *permutation* cache: rather than using a conventional tag/index structure, we use a 32-bit block cypher to encrypt the IP address to create the tag and index, which can result in a 50% savings in memory by allowing part of the tag to be implicitly stored.

For both these caches, we encode an action (shunt, sample, forward, or drop) and a priority. Additional rules also encode actions and priorities based on fixed-header fields. The hardware selects the highest priority match, or, if no match, defaults to shunting the packet to the host. Additionally, the rules for connections can have an optional record that specifies an alternate destination MAC, VLAN, and/or output port to which connections should be forwarded, and an alternate rule that applies if the TCP sequence number is within a specified range (to skip over items within TCP streams).

The resulting design requires 21,200 4-LUTs for logic, 2,770 LUTS for route-through (87% of the available resources), and 135 out of 136 available BlockRams. It requires only 41 cycles to make a decision when unloaded (and no more than 101 cycles when fully loaded), running at 62.5 MHz. Packets that pass directly through the hardware path see only 5 μs of additional network latency.

We tested the Shunt's ability to process large data rates using ipperf [7] in the DETER testbed [3]. Using a single sending host and a receiving host on the other side, each Shunt port proved capable of receiving and processing data at 480 Mbps. This figure is below a full Gbps because of a bug we discovered in the input FIFO that causes a lockup condition when attempting higher data rates. Additionally, one other bug we found prevents the reading of correct VLAN-tagged packets by the device, but works fine for untagged packets.

3 Dissemination of the Research

The initial development of the Shunting architecture, including a detailed trace-driven simulation, formed the second half of the Ph.D. dissertation of Jose Maria Gonzalez, a doctoral student at the University of California, Berkeley [5]. The following year we explored the role of Shunting for supporting intrusion detection/prevention on massively parallel systems [12]. At the conclusion of the project, we published a detailed description of our concrete FPGA-based implementation in [15], and a comprehensive examination of the issues regarding its use for network intrusion detection and prevention in [4].

4 Future Work

If we were to continue, the natural next steps for this project would be investigation of FPGA platforms suitable for 10 Gbps operation on specialized networks, such as scientific networks dominated by large, unanalyzable data transfers. Doing so would require two 10-Gigabit interfaces and a substantially larger FPGA.[1] The most stringent requirement would concern memory, however. If we assume a mean packet size of at least 500 bytes, bidirectionally we must process 5 million packets/sec. Assuming QDR SRAM, each packet requires random accesses to 4 32-bit words for the IP table[2] and 8 words for the connection table (128b entry size, 2-way associative). In total, we require access to 12 words per packet, or 60M words/sec. We anticipate achieving this using a single QDR SRAM with 2-4 MB devoted to the Shunt, or two 2-4 MB independent ZBT SRAMs. We believe these requirements are reasonable for such a high-performance board.

References

[1] M. Crovella. Performance evaluation with heavy tailed distributions. In *JSSPP '01: Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing*, pages 1–10, London, UK, 2001. Springer-Verlag.

[2] M. Crovella and A. Bestavros. Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes. In *Proceedings of SIGMETRICS'96: The ACM International Conference on Measurement and Modeling of Computer Systems.*, Philadelphia, Pennsylvania, May 1996. Also, in Performance evaluation review, May 1996, 24(1):160-169.

[3] The DETER security testbed, http://www.deterlab.net.

[4] Jose Gonzalez, Vern Paxson, and Nicholas Weaver. Shunting: A hardware/software architecture for flexible, high-performance network intrusion prevention. In *CCS '07: Proceedings of the 7th ACM Conference on Computer and Communications Security*, 2007.

[5] Jose Maria Gonzalez. *Efficient Filtering Support for High-Speed Network Intrusion Detection*. PhD thesis, University of California, Berkeley, 2005.

[6] M. Handley, C. Kreibich, and V. Paxson. Network intrusion detection: Evasion, traffic normalization, end end-to-end protocol semantics. In *Proceedings of the 9th USENIX Security Symposium*, 2001.

[7] National Laboratory for Applied Network Research, Distributed Applications Support Team, Iperf, the TCP/UDP Bandwidth Measurement Tool. http://dast.nlanr.net/projects/iperf/.

[8] S. Kornexl, V. Paxson, H. Dreger, A. Feldmann, and R. Sommer. Building a time machine for efficient recording and retrieval of high-volume network traffic. In *Proceedings of the ACM Internet Measurement Conference,* 2005.

[9] N. McKeown and G. Watson. Netfpga 2.0, http://klamath.stanford.edu/nf2/.

[10] V. Paxson. Empirically derived analytic models of wide-area TCP connections. *IEEE/ACM Transactions on Networking*, 2(4):316–336, 1994.

[11] V. Paxson and S. Floyd. Wide area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, 1995.

[12] Vern Paxson, Krste Asanovic, Sarang Dharmapurikar, John Lockwood, Ruoming Pang, Robin Sommer, and Nicholas Weaver. Rethinking hardware support for network analysis and intrusion prevention. In *Proceedings of the USENIX Hot Security Workshop*, August 2006.

[1] 10 Gbps FPGA designs do not run at faster rates than 1 Gbps designs; rather, they simply use 8x larger datapaths.

[2] QDR SRAM fetches pairs of words.

[13] T. H. Ptacek and T. N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks, Inc., Calgary, Alberta, Canada, 1998.

[14] Haoyu Song, Sarang Dharmapurikar, Jonathan Turner, and John Lockwood. Fast hash table lookup using extended Bloom filter: An aid to network processing. In *SIGCOMM*, 2005.

[15] Nicholas Weaver, Vern Paxson, and Jose Gonzalez. The Shunt: An FPGA-Based Accelerator for Network Intrusion Prevention. In *15th International Symposium on Field Programmable Gate Arrays*, February 2007.

[16] W. Wilinger, V. Paxson, and M. Taqqu. Self-similarity and heavy tails: Structural modeling of network traffic. In R. Adler, R. Feldman, and M. Taqqu, editors, *A Practical Guide To Heavy Tails: Statistical Techniques and Techniques*. Birkhauser, 1998.

[17] W. Willinger, M. Taqqu, R. Sherman, and D. Wilson. Self-similarity through high-variability: Statistical analysis of Ethernet LAN traffic at the source level. *IEEE/ACM Transactions on Networking*, 5:71–86, 1997.