Design of a Secure Packet Processor

Danai Chasaki and Tilman Wolf
Department of Electrical and Computer Engineering
University of Massachusetts, Amherst, MA, USA
{dchasaki,wolf}@ecs.umass.edu

ABSTRACT

Programmability in the data path of routers provides the basis for modern router implementations that can adapt to new functional requirements. This programmability is typically achieved through software-programmable packet processing systems. One key concern with the proliferation of these programmable devices throughout the Internet is the potential impact of software vulnerabilities that can be exploited remotely. We present a design and proof-of-concept implementation of a packet processing system that uses two security techniques to defend against potential attacks: a processing monitor is used to track operations on each processor core to detect attacks at the processing instruction level; an I/O monitor is used to track operations of the router to detect attacks at the protocol level. Our prototype implementation on the NetFPGA system shows that these monitors can be implemented to operate at high data rates and with little additional hardware resources.

General Terms

Design, Performance, Security

Categories and Subject Descriptors

C.2.6 [**Computer-Communication Networks**]: Internetworking—*Routers*; C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*

1. INTRODUCTION

The Internet is a critical component of modern communication infrastructure. While security concerns were not a priority during the design of the Internet [6], society's reliance on the Internet today requires that the network can be protected from malicious attackers. Thus, it is essential that the network architecture is extended to integrate the core principles of information security specified by the CIA triad of confidentiality, integrity, and availability [23].

ANCS'10, October 25–26, 2010, La Jolla, CA, USA.

End-to-end security protocols have been developed to provide confidentiality and integrity (e.g., Transport Layer Security (TLS) protocols). However, providing assurance of availability is a more challenging problem as it involves the entire network infrastructure. In the current Internet, there are very few techniques available to protect the network from denial-of-service attacks (even for nation states [12]). Since most large-scale denial-of-service attacks are of distributed nature and generated by botnets [11], defense mechanisms aim to prevent end-system intrusion and thus to limit the access to platforms from which attacks can be launched. Widely deployed intrusion prevention systems include firewalls [16] and deep packet inspection [22].

In today's Internet, intrusion prevention mechanisms (and thus defenses against denial-of-service attacks) aim at end-systems. However, there are several important trends in networking that indicate that such defenses are insufficient against novel attacks that target vulnerabilities in the *data plane* of the network infrastructure itself. These vulnerabilities emerge because modern networks use numerous devices that are based on programmable components in the data path:

- High-performance router implementations use embedded multi-core processing systems (i.e., network processors) for packet forwarding to accommodate advanced data plane functions [8]. On these systems, packet forwarding is implemented as software operations on general-purpose processor cores.
- Recent clean-slate network architectures consider various new protocols and data communication paradigms. To accommodate networks with different protocol stacks on a single network infrastructure, network virtualization has been proposed [1, 24]. Since network slices are deployed dynamically on a virtualized network substrate, the routers in the substrate need to be able to support custom packet processing functions. This programmability can be achieved using network processors [25] or operating system virtualization on conventional workstation processors [13].

As with any software-programmable system, there are potential vulnerabilities that can be exploited by an attacker to gain access to and/or modify the operation of the system. A study of vulnerabilities in network devices in the current Internet indicates that there are large numbers of potential attack targets [7]. When considering the potential impact of a denial-of-service attack that is launched from a core router that is connected to dozens of links with 40 Gbps

data rates, it becomes clear that there is a need to protect these systems.

In this paper, we present a design and results from a prototype implementation of a secure packet processor. This packet processor can perform custom packet forwarding functions to support a wide range of protocols. A special processing monitor is used to track the instruction-level operations of each processor core. These operations are compared to a representative model of the packet processing task that has been obtained through offline analysis of the packet processing binary. Under normal conditions, the operations reported by the processor match the model in the monitor. If an intrusion attack occurs that changes the operations of the processor (e.g., to execute malicious code), then these operations no longer match the model in the monitor. The secure packet processor can detect these conditions and initiate a recovery step that resets the processor core and allows continued operation.

We demonstrate the effectiveness of this secure processor design by presenting results from an implementation on a NetFPGA platform. The prototype can detect an example attack where the control flow of the packet processor deviates from that in the original processing binary. The system can correctly detect this attack, halt the processor, drop the packet, and restore the system within 6 instruction cycles. This very small time for recovery allows our secure packet processor to operate at full data rate even when under attack. The overhead for adding a monitoring system to the packet processor is very small (0.8% increase on slice LUTs and 5.6% on memory elements). The specific contributions of our paper are:

- The design of a secure packet processor that uses existing monitoring techniques to detect the effects of an intrusion attack. The system can quickly recover from such attacks by resetting the processor system.
- A prototype implementation of this processing system on a NetFPGA platform. The processor core is based on a Plasma core that is extended with our monitoring and recovery system.
- Results from the operation of the prototype system that illustrate the correct detection of an attack and the fast recovery mechanism that allows the system to run a full speed even under attack.

We believe that this work presents an important first step towards designing router systems that provide the necessary flexibility in packet processing but also provide sufficient security mechanisms to protect modern network infrastructure.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 introduces the design of the secure packet processor. The prototype implementation is presented in Section 4. Results from the operation of the prototype are presented in Section 5. Section 6 summarizes and concludes this paper.

2. RELATED WORK

Network security research has focused on topics ranging from secure end-to-end protocols to anomaly detection heuristics [9]. To protect networks and end-systems from denial-of-service attacks, packet marking strategies have been proposed to identify spoofed sources [21]. Capabilities-based networks require positive authentication of traffic before forwarding is performed [3]. Extensions protect from denial-of-capabilities attacks [18] and provide one-hop containment [26]. These approaches defend against the effects of denial-of-service attacks.

The premise of a denial-of-service attack is that an attacker has a large number of systems from which traffic can be directed to a target. Therefore, an attacker needs to gain control over a large number of systems. This access is accomplished through intrusion techniques (e.g., trojan horse, etc.) that make the system (or parts of it) remotely controllable. From the network side, firewalls [16] and intrusion detection systems [22] can protect some systems from some known attacks. On end-systems, virus scanner software can also identify some attacks.

The use of virus scanner software as defense against intrusion assumes that a sufficiently powerful processor and operating system are available. This assumption does not hold when considering embedded packet processors on routers. These systems frequently use network processors, which are embedded multi-core systems-on-a-chip [4,5,10] that operate without operating system support to maximize throughput performance. These embedded processing systems are vulnerable to intrusion just as conventional end-systems are [7]. Our work focuses on methods to protect these packet processors through hardware extensions that do not impact the overall processing performance.

We use processing monitors to track the operations on the packet processor. The monitor can determine if attacks occur because the processor's operations deviate from the operations that are valid (as determined by offline analysis of the processing binary). The techniques we use have been developed in prior work by us and others for the use in embedded systems in general. Like network processors, embedded systems in general are characterized by the lack of operating system support for intrusion detection (and often the lack of performance to do so in software). Our previously designed hardware monitor for embedded systems can track each instruction of the processor and compare it to the processing model used by the monitor [15]. Other monitors [2, 19] use similar techniques, but operate at the granularity of basic blocks and thus are slower in detecting attacks. Similarly, the system in [30] determines correct operation across blocks of instructions. Other techniques extend the processor instruction set and micro-architecture to support special verification steps [17, 20]. Our approach differs from related work in that it does not require changes to the processing binary (which is beneficial for third-party code) and that attacks can be detected within a few instructions rather than at the end of a longer code block.

Our idea of using monitoring techniques to protect networking systems has first been published in [27]. In this paper, we make these general ideas more concrete by presenting a detailed design of the processing monitor and results from a prototype implementation on a NetFPGA system. Thus, this work provides the first definite evidence that the proposed system can indeed detect and recover from attacks and can do so at speeds that do not degrade overall throughput performance. This last point is particularly important since the recovery mechanism should not present a target for a new type of denial-of-service attack where a single malicious packet can trigger time-consuming recovery operations.

3. SECURE PACKET PROCESSING

The key concepts behind the design of a secure packet processing system are discussed in this section. More details on the implementation of the security features are covered in Section 4.

3.1 Security Model

To provide a basis for the discussion of security in our system, we state the security requirements and attacker capabilities. For security requirements, we assume that

- An attacker should not be able to make a router perform any action that deviates from normal forwarding behavior.
- Intrusion attempts trough the data path of the router should lead to a drop of the offending packet.
- If an intrusion attempt has changed the internal state of the router, a recovery mechanism should reset the system to a secure state.
- Intrusion attempts should not lead to denial-of-service due to recovery overhead.

In the context of these security requirements, we assume that a malicious entity is able to perform the following actions:

- Send packets (control or data) to the packet processor, possibly triggering abnormal behavior.
- Gain remote access to the system and change the data memory, the instruction memory of the processor, log files, or extract and modify secret keys.
- Launch Denial-of-service attacks by sending massive traffic or by directly disabling links.
- Use reprogramming interfaces to control the entire router.

However, an attacker does not have physical access to the router and cannot access the binary file of the application currently executed on the packet processor, because it resides outside the platform. Once it is launched on the instruction memory of the hardware platform though, memory modification is considered a potential attack scenario.

One potential attack example is the following: An attacker transmits a data packet that contains malicious code (e.g., within the packet header or the packet payload). If the packet is carefully crafted, a buffer overflow on the packet processor may occur. This can lead to a stack smashing attack, which can be used to modify the control flow of the packet processing program. One possible target for the redirected control flow is the malicious piece of code contained in the packet. If that code is executed, the attacker can execute nearly arbitrary operations on the packet processor. For example, the attacker can launch a denial-of-service attack by repeatedly sending the attack packet towards a victim node. With access to high-performance packet processors and high-speed network links on the router, this type of denial-of-service attack could be very effective, even if only a few systems can be compromised.

The attacks mentioned in this section are not a comprehensive enumeration of all likely scenarios. We just outlined the general context of the possible ways in which a misbehaving user can attack a packet processing system. We tried to be as general as possible in our assumptions, and include most of current and next-generation network vulnerabilities.

3.2 Attack Detection through Monitoring

The main idea behind our secure packet processor is to integrate monitoring functionality into the hardware of the packet processing system. When an attacker attempts to hack into the software-programmable processor cores of the system, they may succeed in changing the processing behavior of the system. However, the monitoring systems in the packet processor can detect this change and trigger a response (i.e., packet drop and system recovery). Since our monitoring components are embedded in the system hardware, it is difficult for a hacker to attack both the processor and the (hard to access) monitors at the same time. Thus, this approach by design provides more security than a conventional general-purpose processing system.

There are several important challenges that need to be met when using hardware monitors in a processing system:

- Correct detection: The monitoring system needs to be able to correctly identify intrusion attacks. Our system achieves this by checking for any deviation from the known correct operation of the packet processor.
- Fast detection: When intrusion occurs, it is important to detect it quickly to reduce its potential impact. Our system can detect (and recover from) deviations in processing operations within only four processing cycles.
- Low overhead: The resource requirements for a monitor should be small to limit the impact of monitoring on system cost. Our monitoring system only requires single-digit percent additional hardware resources compared to a conventional packet processor implementation.

Before discussing the detailed operation of the monitoring system, we describe the high-level system architecture.

3.3 System Overview

The main design goal of our system is to provide security techniques to defend against potential attacks on a software-programmable packet processing system. Our system builds on the four-core prototype of a packet processor as described in previous work [29].

Figure 1 presents a system (high) level view of the software-programmable packet processor that uses security modules to ensure the correct functionality of a modern router. As discussed in [29] next-generation routers are expected to have multiple packet processing units, in order achieve fast and balanced processing of packets that belong to different flows. A flow classification unit assigns packets to specific flows, and an output arbiter module sends the processed packets to the corresponding outputs.

Each packet processing unit will be possibly executing a different program in order to process the packets that are distributed to it. If we assume that an attacker is able to access and modify the instruction memory of each individual PPU (processing attack), the whole router will be compromised. To prevent that from happening, we can have an instruction-level security monitor attached to each PPU,

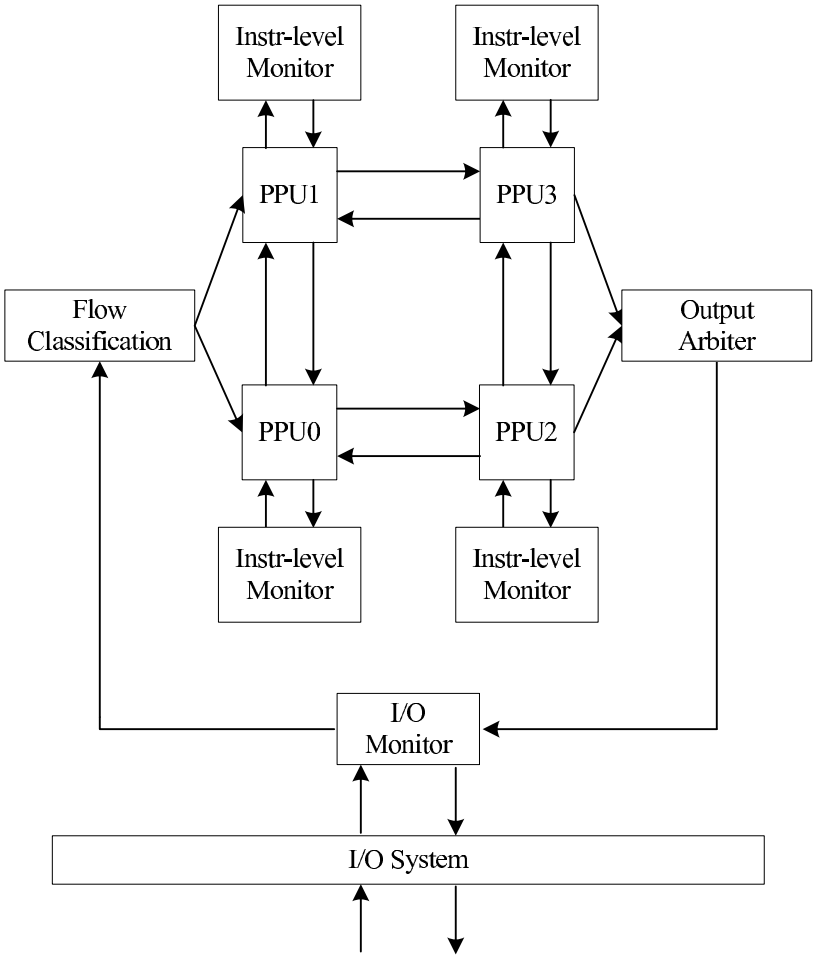

Figure 1: Security Monitoring on the Packet Processor

which checks in real time each and every instruction executed on the processor core and determines if it is valid or not. We will explain the way this check is performed in the next section.

Moreover, due to vulnerabilities in the data path, we would expect the packet processor to be attacked at the protocol level as well. We can imagine a situation where valid processing instructions are executed on the PPUs, but still the overall router behavior is abnormal. For example, if the IP-forwarding routine is running on one of the cores, a Denial-of-Service attack could flood the system by requesting a large amount of duplicate packets to be forwarded to the output interfaces. To counter such kinds of attacks, we could use an I/O monitor attached to the I/O interface of the packet processor, which checks certain characteristics of incoming and outgoing packets: payload checksum, packet count, tags or time-stamps in the header of the packets etc.

3.4 Monitoring Functionality

Both monitoring systems function independently from the packet processor. They use separate hardware resources, which makes sure that an attack targeting the processor will not affect the security monitors' operation. Moreover, they use up as few resources as possible, while keeping the monitoring speed synchronized with the packet processor's speed.

The main idea behind the instruction-level monitor is illustrated in Figure 2.

- Prior to installing a specific protocol processing routine on the packet processor, we analyze the binary file of the application by breaking it down to basic blocks of instructions and determining all the possible execution paths.
- The derived information is stored in a 'basic block' data structure on the hardware platform.
- The processor, during runtime, keeps updating the security monitor with the monitoring stream.
- If the processor's current execution path deviates from the correct one - as instructed by the basic block data structure - the security monitor detects an error.

In case of error detection the monitoring system assumes that the instruction store must have been modified, and it takes the recovery route. It interrupts the packet processor's operation on the specific packet, reloads the protocol processing code on the processor from a storage place that is assumed to be secure and not accessible by the attacker.

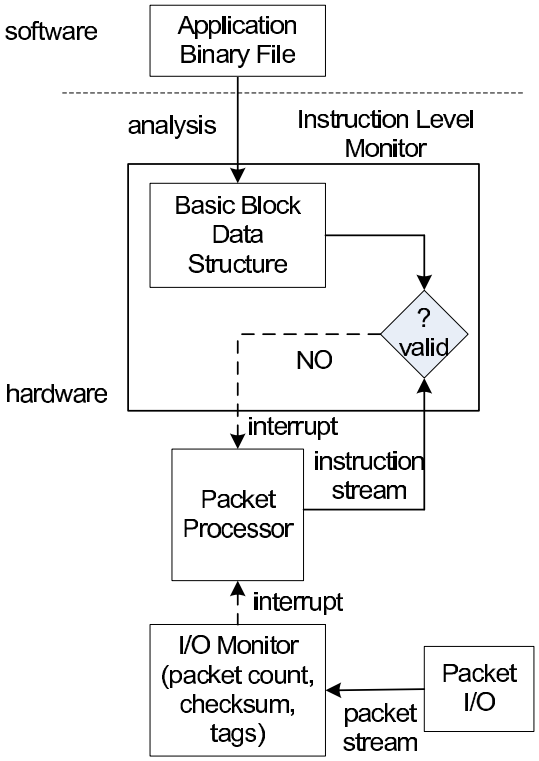

Figure 2: Monitoring System Overview

To enhance the instruction-level monitor functionality, an I/O monitor [28] can be used to track the I/O behavior of the packet processing system. This is a higher level monitor that records protocol specific characteristics. It is not connected to the packet processor, but to the input and output interfaces. The I/O monitor correlates the stream of incoming packets to the stream of outgoing packets to detect cases where the router is not operating as expected. The abnormality here is detected from a network protocol perspective, whereas the router's function can look legitimate from a processing point of view (e.g. correctly executed instructions during a DoS attack).

There are several types of information that can be collected by just observing the incoming and outgoing packets. The most intuitive thing to document is the number of incoming and outgoing packets, and check for imbalance. Of course, if the flow of packets increases in the output interface, this does not always indicate an attack condition. The I/O monitor should be able to account for protocols that allow for multicast or broadcast services. Moreover, the checksum of the packet's payload can be computed to identify unauthorized alterations. By placing tags or time-stamps in the headers of the incoming packets, the I/O monitor will be able to detect whether a specific incoming packet is directed to the output interface with significant delay. Such an event signals abnormal degradation of processing performance.

4. PROTOTYPE IMPLEMENTATION

In this section, we provide in detail the design and proof-of-concept implementation of a packet processing system that uses the two security techniques we have described above. Our prototype is implemented on a NetFPGA [14], which contains a Virtex2-Pro FPGA device and is used for experimental purposes. Our design is scalable and can be ported on other FPGA platforms or ASICs.

4.1 Instruction-level Monitor

For this prototype, to be consistent with the NetFPGA design and the packet processor speed, we used 64-bit data path and designed all the units to operate at 62.5MHz. The security monitor runs in parallel to the packet processing unit, and is designed to use four pipeline stages.

The first task is to decide what the monitoring stream, which the PPU continuously sends to the security monitor, should be. According to our assumptions, an attacker can abuse the packet processor's operation, either by modifying the current protocol routine to execute malicious code, or by adding some piece of code that performs malicious operations. We can monitor such malicious behavior by making the packet processor stream information regarding the current execution path in realtime. There is a variety of options to choose from:

- Opcode: By sending to the monitor opcode information, we monitor the operations performed on the processor, which indicate the functionality of the executed application. For an attack to become possible, the attacker will have to replace the instruction set, with another malicious set of instructions that use the same opcodes in the exact same sequence.

- Instruction address: Since the memory address used to store the instruction set is unique, the attacker would have to write malicious code that stores the new instructions in the same location in the instruction memory as the original application does. This would also require the malicious code to branch at the same exact points with the legitimate code.

- Instruction address+Instruction word: This kind of streaming pattern combines two pieces of information, and makes it harder for an attacker to come up with attack code that goes undetected. We could also add the opcode, or control flow information to the monitoring stream, but this will cause a significance increase in the system's resource consumption.

- Hash of any of the above: The processor is streaming a compact hashed value of any of the above combinations. The more bits we use to compute the hash, the stronger the monitoring pattern is. However, the number of used bits will affect the memory utilization. After all, it is a trade off between available memory on the hardware platform and the strength of security features.

Depending on the information we choose to stream, the software analysis and the contents of the basic block data structure shown in Figure 2 have to be adapted accordingly. For our prototype, the instruction address information was used. Before we load a specific protocol processing routine

on one of the processor cores, we analyze the application binary file off-line and break it down to a number of basic blocks. We place instructions that are executed the one after the other in the same basic block, which ends with a conditional or unconditional jump instruction. We use a block RAM memory on the FPGA to store information about the program's execution path. This memory (data structure) is indexed by the instruction address sequence of the application and contains two blocks for each entry. The first one is the basic block each instruction memory address belongs to, and the second is the potential next hop address the instruction could jump to. This BRAM is used as a guide to the correct processor core operation.

The implementation level details of the instruction level monitor are shown in Figure 3. Each pipeline stage takes once clock cycle to complete.

In the first pipeline stage, we extract the address of the currently executed instruction on the packet processing unit. We use this address to index the BRAM, which takes one cycle to output the basic block in which this instruction resided, and the next hop address (in case of a jump instruction), if there is one.

In the second stage, we propagate the current basic block and next hop information we get from the BRAM, and give those values as input to the third stage. At the same time, we record the current basic block information into a FIFO module. This FIFO is used to keep track of the execution path, by storing the previously and currently executed basic block numbers. This module has minimal memory requirements, because it only contains two values at a time. When we read from it, the head of the FIFO outputs the previous basic block, and when we write in it, we record the current basic block, which we read in the next clock cycle and so on.

The third and fourth stages are the most important, since they implement our monitoring algorithm:

- Check if the current basic block number matches the basic block of the previous instruction.
- If it does, it means it is a valid instruction – continue to the next one.
- If not, check if this instruction is within the next basic block.
- In case it is, this is a valid basic block jump – continue with the next instruction.
- If not, go to the fourth stage and use the next hop address information to index the BRAM. Verify that the currently executed instruction is a valid jump instruction.
- If it is, it denotes correct operation – continue.
- Otherwise, signal that the packet should be dropped.

In the final step, the monitor sends the packet drop signal to the packet memory unit, which stops the processing of the current packet. Then, the reset process is initiated to ensure that the system can continue operating. The reset process includes the following steps:

- The packet buffer, where the current packet is stored, drops the packet and resets to receive a new packet.
- The processor stack and register file is reset.

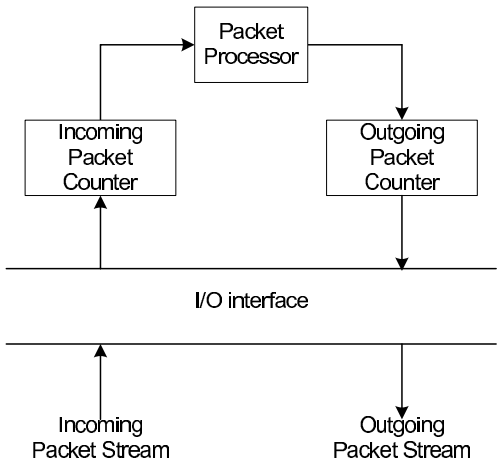

Figure 4: I/O Monitor Design

- The instruction memory is reset, so that potentially tampered code is overwritten and does not affect the next packet to be processed by the packet processor.

To efficiently implement the reset operation of the instruction memory, we first switch to a backup memory (which resides safely inside the hardware platform) and then start reloading the memory initialization file back to the infected instruction memory of the processor. In this way, in case another attack happens during the processing of the next packets, we can immediately switch back to the corrected instruction memory.

4.2 I/O Monitor

The I/O monitor design in our prototype is very simple and is shown in Figure 4. The purpose of this implementation is to illustrate just one example of how a high-level (protocol related) monitor should function, and show that it is possible to implement this feature on our packet processing system with no additional overhead, and without having to compromise on the throughput of the processor. We use resources (logic) that are already present on the reference NetFPGA router, to design two counters, one attached to the input queue interface keeping track of the incoming packets, and the other one connected to the output queues, counting the outgoing packets (within a specific window).

Thanks to the flow classification module implemented on our router, we can easily extend the packet count functionality of the I/O monitor, and make the monitoring operation protocol-aware. It is possible to check the header of every packet to determine which protocol is currently running on each processor core. Based on this information, we can make the I/O monitor treat traffic in a protocol specific way. For example, if it is determined that a unicast protocol is executed (http, ftp etc.) the I/O monitor would expect the incoming packet count to be equal or greater than the outgoing packet count. In the case that the protocol allows for multicast capability, an increase in the outgoing packet count should be considered legitimate protocol be-

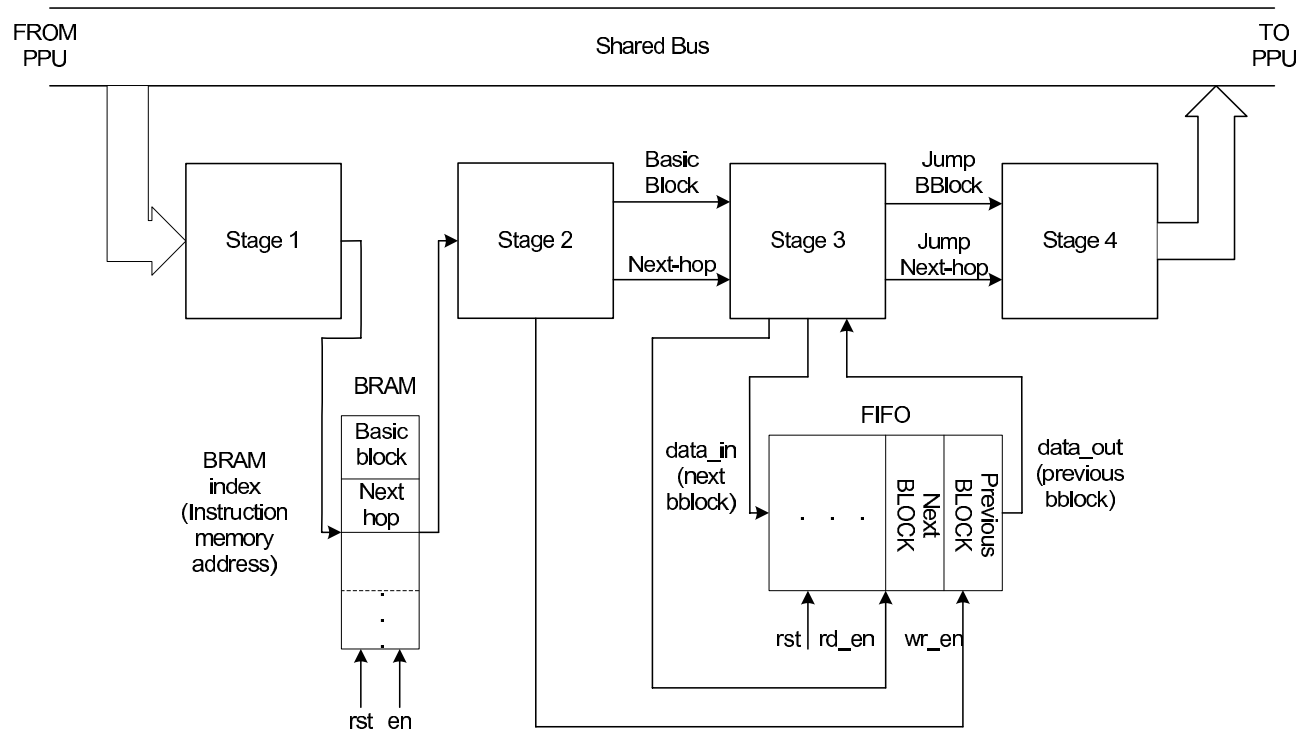

Figure 3: Instruction Level Monitor Design

havior (one incoming packet corresponds to N outgoing). In either scenario, once unusual operation is observed, the packet memory should be flushed and the packet processor reset.

5. EVALUATION

In this section, we demonstrate the correct functionality of the implemented security features, by crafting an attack example. We also discuss resource utilization and performance results. For the system's performance, we use the delay in detecting abnormal instructions and the system's throughput as metrics.

5.1 Triggering Invalid Behavior

In this scenario, we first load a unicast IP-forwarding routine on the packet processor. For our prototype, we have chosen to store instruction addresses for monitoring purposes. Alternatively, as mentioned in Section 4.1, we could have implemented a security monitor that contains the hash value of the instruction address and the corresponding instruction word. Such a monitoring pattern uses less memory resources and makes it even harder for the attacker to come up with attack code [15].

Lets assume that our application code at some point executes a branch instruction, with two possible basic blocks where the program can jump to. The corresponding disassembled application code is shown in Figure 5. As we can see, the potential jump targets are memory addresses 0x0218 or 0x01e4. The former contains valid code, whereas the latter corresponds to attack code. On the other hand, inside the security monitor we store instruction addresses only for the valid basic blocks.

Since we do not inform the security monitor that the program can possibly branch to memory address 0x01e4, the monitor will consider a jump to this particular address as an attack. So, in order to trigger an attack, we can send to the processor packets that cause our program to take this specific execution path (address 0x01e4). This path is unknown to the instruction level monitor, thus, the monitor treats the received packets as attack packets.

5.2 Security Monitor Operation

In Figure 7, we show how the whole system (packet processor and security monitor) operates under the attack scenario. The horizontal axis denotes the clock cycle at which each operation takes place. In our experiment, we send 4 small packets (around 60 bytes long) into the router. The first is an 'attack' packet and the remaining 3 are valid packets. We should note here that under normal operation the total time the packet processor takes to forward a small packet is around 600 clock cycles [29].

The packet processing system functions as follows: The four packets come back to back into the packet processor's packet buffers. At some point, while the first packet is getting processed, the program jumps to the memory address 0x01e4, which is unknown to the monitor and triggers an attack response. The monitor starts performing the operations we described at Figure 3 in four pipelined stages, and 5 cycles later packet 1 is dropped. At this point our instruction-level monitor stalls the processing of the current packet, drops the packet, and in the same clock cycle the instruction memory is reset. The recovery phase of the system takes approximately 6 cycles. After those few cycles, the processing resumes and the remaining three packets are

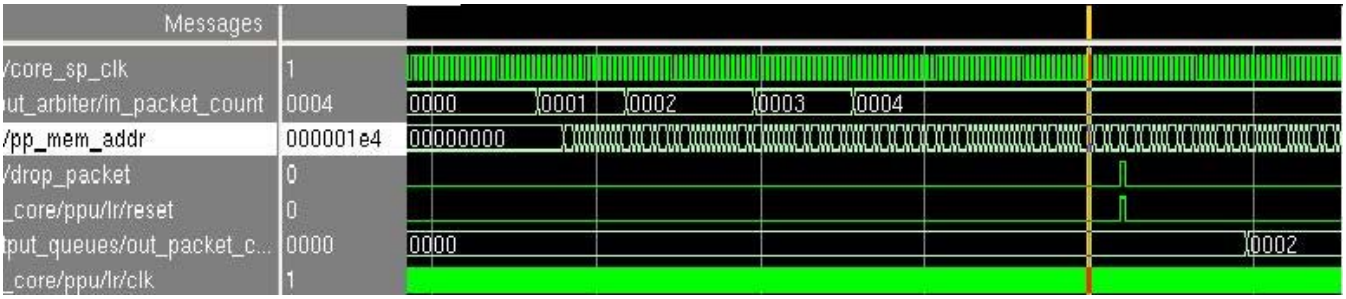

Figure 6: Simulation Results.

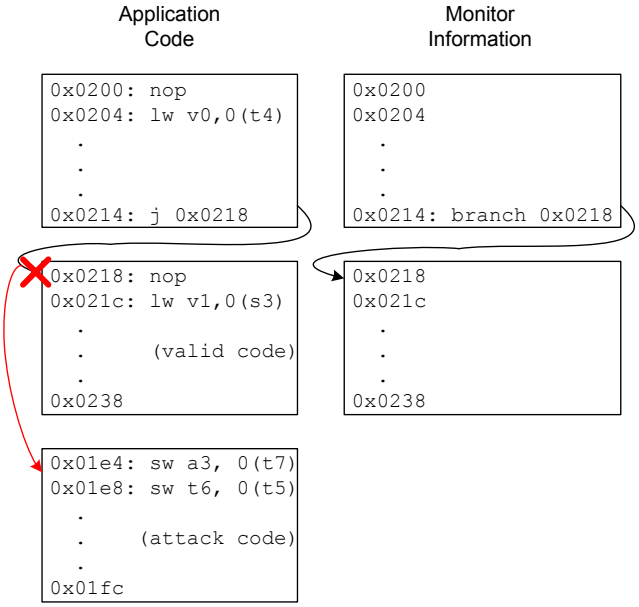

Figure 5: Attack scenario.

forwarded normally. Because of the 6 cycles, in which the system stays idle (recovery phase), the total processing time of packets 2,3 and 4 increases from 600 to 606 cycles.

In Figure 6 we can observe the same sequence of events in our simulator tool (Modelsim). Signal *in_packet_count* counts the incoming packets, *out_packet_count* the outgoing, and 'drop packet' is the signal that notifies the processor to drop the current packet. Packet 1 gets dropped when the signal *pp_mem_address* is equal to 0x01e4, which is the initial instruction address of the 'attack' block of instructions. By setting the signal $ppu/lr/rst$, we rewrite the instruction memory of the IP-forwarding application. Due to space constraints, we only show packet 2 coming out of the output queue.

Parallel to the instruction level monitor, our I/O monitor counts incoming and outgoing packets. It reports 4 incoming packets and 3 outgoing. Since we experimented on the unicast IP-forwarding routine, our I/O monitor considers the protocol level behavior legitimate. As expected, it does not detect the instruction level attack.

A theoretical scenario where we would have the opposite outcome (the I/O monitor detecting an attack while the instruction level monitor does not detect the attack) would be if for the same unicast IP-forwarding protocol, we duplicate some packets and send them to all ports jamming them. Then the I/O monitor would count more outgoing packets than incoming, which is not usual behavior for a unicast protocol. On the other hand, there is no reason for the instruction level monitor to detect an attack, since the processing routine (forwarding) is executed correctly for all packets.

5.3 Performance Results

As mentioned in the previous section, once an attack is detected, the recovery phase of the instruction level monitor lasts only a few cycles. This time is necessary for the instruction set to be correctly reloaded on the NetFPGA. Compared to the number of cycles it takes for the processor to forward even a small packet (around 600 cycles), the time our system stays idle before resuming correct processing functionality is not significant. This is an important feature because, otherwise, an attacker could just keep on sending packets that cause the system to misbehave, so that the processor is locked into a repeated effort of long recoveries, without doing any useful processing at the same time. That would become a vulnerability of our recovery mechanism that leads by itself to DoS attacks.

To evaluate the overhead of the monitoring system, we performed an experiment to measure the throughput of our packet processing system when the two security monitors are on: We have three working Ethernet ports on our prototype packet processing system. We setup a testbed by connecting the one of the NetFPGA ports to another machine, on which we measure throughput. First, we experiment by sending valid packets only, and then by sending a combination of valid and invalid packets. In Table 1, we report the average throughput numbers for different configurations. The important thing to note is that the throughput of the single processor with embedded security monitors is almost the same with the throughput that the single core achieves by itself. The data rates are in the order of 60 Mbps because the experiment was performed by forwarding small packets. In the case of large packets the packet processor can achieve throughput greater than 2 Gbps [29].

Table 1 also shows the resource consumption of our monitoring systems, the packet processing system with both security monitors uses 0.8% more slice LUTs compared to the single core processor alone. Memory-wise we observe an increase of 5.6% in the consumption of block RAMs. These results show that only small amounts of resources are required for our monitoring system.

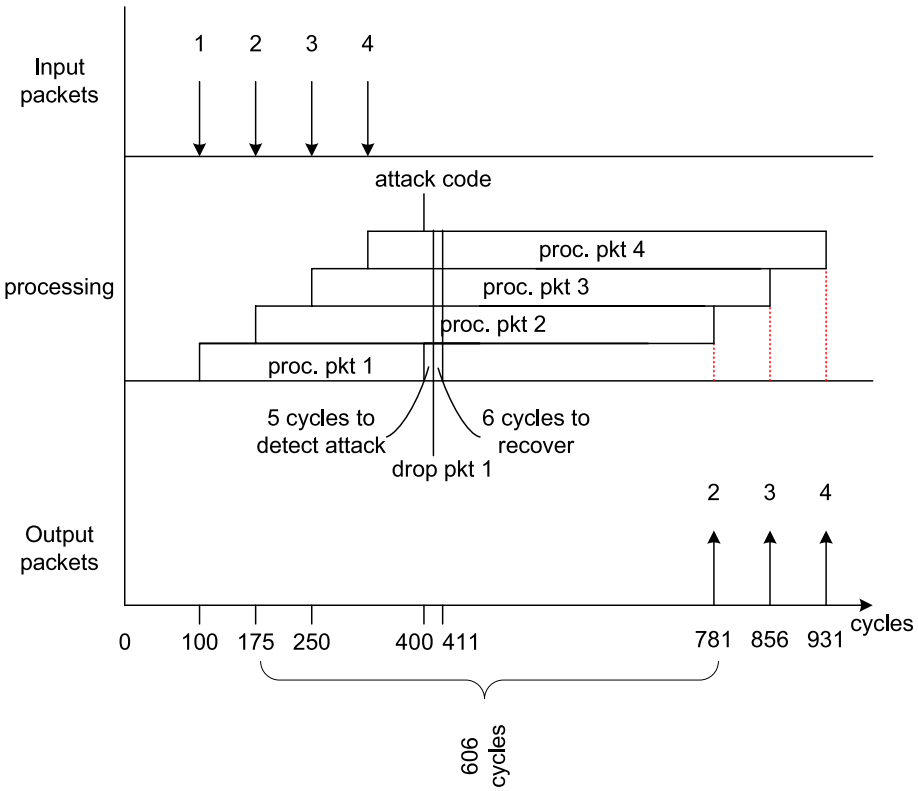

Figure 7: Security Monitor Operation.

Table 1: Resource consumption and performance.

	Single Processor	Single core w/ Instr. level monitor	Single core w/ both monitors
Slice LUTs	15,025	15,112	15,134
BRAM (RAMB16s)	124	130	131
detection time	NA	5 cycles	mon. window
speed (MHz)	62.5	62.5	62.5
throughput(avg in Mbps)	67.2	64.1	63.9

6. SUMMARY AND CONCLUSIONS

The use of software-based packet processing in routers with general-purpose processing engines is becoming more prevalent in the Internet. The use of software in the data plane of the network presents a target for novel intrusion and denial-of-service attacks that can have significant impact on the overall security of the network. In our work, we present the design and prototype implementation of a secure packet processor that is equipped with a monitoring system that can detect such attacks. The monitoring system compares the operation of the processor cores to the expected behavior that is obtained from analyzing the packet processing binary. A processing monitor continuously checks the validity of processor operations and triggers a recovery mechanism when deviations from expected behavior are detected. The prototype implementation of our system can detect intrusion attacks within six processor cycles and recover the system in that time. The result from the prototype system indicate that our design is an effective approach to protecting networking infrastructure in the future Internet.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. CCF-0952524.

7. REFERENCES

[1] Anderson, T., Peterson, L., Shenker, S., and Turner, J. Overcoming the Internet impasse through virtualization. *Computer 38*, 4 (Apr. 2005), 34–41.

[2] Arora, D., Ravi, S., Raghunathan, A., and Jha, N. K. Secure embedded processing through hardware-assisted run-time monitoring. In *Proc. of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05)* (Munich, Germany, Mar. 2005), pp. 178–183.

[3] Ballani, H., Chawathe, Y., Ratnasamy, S., Roscoe, T., and Shenker, S. Off by default! In *Proc. of Fourth Workshop on Hot Topics in Networks (HotNets-IV)* (College Park, MD, Nov. 2005).

[4] Cavium Networks. *OCTEON Plus CN58XX 4 to 16-Core MIPS64-Based SoCs*. Mountain View, CA,

2008.

[5] Cisco Systems, Inc. *The Cisco QuantumFlow Processor: CiscoŠs Next Generation Network Processor*. San Jose, CA, Feb. 2008.

[6] Clark, D. D. The design philosophy of the DARPA Internet protocols. In *Proc. of ACM SIGCOMM 88* (Stanford, CA, Aug. 1988), pp. 106–114.

[7] Cui, A., Song, Y., Prabhu, P. V., and Stolfo, S. J. Brave new world: Pervasive insecurity of embedded network devices. In *Proc. of 12th International Symposium on Recent Advances in Intrusion Detection (RAID)* (Saint-Malo, France, Sept. 2009), vol. 5758 of *Lecture Notes in Computer Science*, pp. 378–380.

[8] Eatherton, W. The push of network processing to the top of the pyramid. In *Keynote Presentation at ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS)* (Princeton, NJ, Oct. 2005).

[9] Estevez-Tapiador, J. M., Garcia-Teodoro, P., and Diaz-Verdejo, J. E. Anomaly detection methods in wired networks: a survey and taxonomy. *Computer Communications 27*, 16 (Oct. 2004), 1569–1584.

[10] EZchip Technologies Ltd. *NP-3 – 30-Gigabit Network Processor with Integrated Traffic Management*. Yokneam, Israel, May 2007. `http://www.ezchip.com/`.

[11] Geer, D. Malicious bots threaten network security. *Computer 38*, 1 (2005), 18–20.

[12] Lesk, M. E. The new front line: Estonia under cyberassault. *IEEE Security & Privacy 5*, 4 (July 2007), 76–79.

[13] Liao, Y., Yin, D., and Gao, L. PdP: parallelizing data plane in virtual network substrate. In *Proc. of the First ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures (VISA)* (Barcelona, Spain, Aug. 2009), pp. 9–18.

[14] Lockwood, J. W., McKeown, N., Watson, G., Gibb, G., Hartke, P., Naous, J., Raghuraman, R., and Luo, J. NetFPGA–an open platform for gigabit-rate network switching and routing. In *MSE '07: Proceedings of the 2007 IEEE International Conference on Microelectronic Systems Education* (San Diego, CA, June 2007), pp. 160–161.

[15] Mao, S., and Wolf, T. Hardware support for secure processing in embedded systems. In *Proc. of 44th Design Automation Conference (DAC)* (San Diego, CA, June 2007), pp. 483–488.

[16] Mogul, J. C. Simple and flexible datagram access controls for UNIX-based gateways. In *USENIX Conference Proceedings* (Baltimore, MD, June 1989), pp. 203–221.

[17] Nakka, N., Kalbarczyk, Z., Iyer, R. K., and Xu, J. An architectural framework for providing reliability and security support. In *Proc. of the 2004 International Conference on Dependable Systems and Networks (DSN)* (Florence, Italy, June 2004), pp. 585–594.

[18] Parno, B., Wendlandt, D., Shi, E., Perrig, A., Maggs, B., and Hu, Y.-C. Portcullis: protecting connection setup from denial-of-capability attacks. In *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications* (Kyoto, Japan, Aug. 2007), pp. 289–300.

[19] Ragel, R. G., and Parameswaran, S. IMPRES: integrated monitoring for processor reliability and security. In *Proc. of the 43rd Annual Conference on Design Automation (DAC)* (San Francisco, CA, USA, July 2006), pp. 502–505.

[20] Ragel, R. G., Parameswaran, S., and Kia, S. M. Micro embedded monitoring for security in application specific instruction-set processors. In *Proc. of the 2005 international conference on Compilers, architectures and synthesis for embedded systems (CASES)* (San Francisco, CA, Sept. 2005), pp. 304–314.

[21] Savage, S., Wetherall, D., Karlin, A., and Anderson, T. Network support for IP traceback. *IEEE/ACM Transactions on Networking 9*, 3 (June 2001), 226–237.

[22] Snort. *The Open Source Network Intrusion Detection System*, 2004. `http://www.snort.org`.

[23] Stallings, W. *Cryptography and Network Security*, 4th ed. Prentice Hall, 2006.

[24] Turner, J. S., and Taylor, D. E. Diversifying the Internet. In *Proc. of IEEE Global Communications Conference (GLOBECOM)* (Saint Louis, MO, Nov. 2005), vol. 2.

[25] Wiseman, C., Turner, J., Becchi, M., Crowley, P., DeHart, J., Haitjema, M., James, S., Kuhns, F., Lu, J., Parwatikar, J., Patney, R., Wilson, M., Wong, K., and Zar, D. A remotely accessible network processor-based router for network experimentation. In *Proc. of ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS)* (San Jose, CA, Nov. 2008), pp. 20–29.

[26] Wolf, T. Data path credentials for high-performance capabilities-based networks. In *Proc. of ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS)* (San Jose, CA, Nov. 2008), pp. 129–130.

[27] Wolf, T., and Tessier, R. Design of a secure router system for next-generation networks. In *Proc. of Third International Conference on Network and System Security (NSS)* (Gold Coast, Australia, Oct. 2009).

[28] Wolf, T., Tessier, R., and Prabhu, G. Securing the data path of next-generation router systems. *Computer Communications* (2010).

[29] Wu, Q., Chasaki, D., and Wolf, T. Implementation of a simplified network processor. In *Proc. of IEEE International Conference on High Performance Switching and Routing (HPSR)* (Richardson, TX, June 2010).

[30] Zambreno, J., Choudhary, A., Simha, R., Narahari, B., and Memon, N. SAFE-OPS: An approach to embedded software security. *Transactions on Embedded Computing Sys. 4*, 1 (Feb. 2005), 189–210.