Attacks and Defenses in the Data Plane of Networks

Danai Chasaki, *Student Member, IEEE,* and Tilman Wolf, *Senior Member, IEEE*

Abstract—Security issues in computer networks have focused on attacks on end-systems and the control plane. An entirely new class of emerging network attacks aims at the data plane of the network. Data plane forwarding in network routers has traditionally been implemented with custom-logic hardware, but recent router designs increasingly use software-programmable network processors for packet forwarding. These general-purpose processing devices exhibit software vulnerabilities and are susceptible to attacks. We demonstrate – to our knowledge the first – practical attack that exploits a vulnerability in packet processing software to launch a devastating denial-of-service attack from within the network infrastructure. This attack uses only a single attack packet to consume the full link bandwidth of the router's outgoing link. We also present a hardware-based defense mechanism that can detect situations where malicious packets try to change the operation of the network processor. Using a hardware monitor, our NetFPGA-based prototype system checks every instruction executed by the network processor and can detect deviations from correct processing within four clock cycles. A recovery system can restore the network processor to a safe state within six cycles. This high-speed detection and recovery system can ensure that network processors can be protected effectively and efficiently from this new class of attacks.

Index Terms—network security, network attack, programmable router, network processor, processing monitor, embedded system security.

1 Introduction

Network security is an important concern in the Internet. Most network security efforts have focused on vulnerable end-systems that are exploited by remote attacks through the network, on denial-of-service attacks that use the network to disable end-systems, and on general information security. Until recently, the network infrastructure itself has not been a major concern for network security since it presented no practical attack target. However, the technology used to implement network routers has changed in recent years and new vulnerabilities are emerging. In our work, we focus on a specific example of a novel type of attack that exploits these vulnerabilities and thereby attacks the network infrastructure itself.

In the past, high-performance network routers have used application-specific integrated circuits (ASICs) to implement packet forwarding functions. While these ASICs were costly to develop, they represented the only technology that was able to achieve the performance that was necessary for multi-Gigiabit per second traffic forwarding. Over the last few years, however, the performance of general-purpose multi-core processors (e.g., network processors or high-end server processors) has reached a level where high traffic forwarding rates can be achieved. Since the functionality of an ASIC cannot be changed once it has been designed, the use of these new general-purpose network processors provides router vendors with much more flexibility to adjust a router's functionality after production [1]. Therefore, there is an ongoing shift in the industry toward implementing routers based on programmable packet processing engines rather than based on ASICs.

A side-effect of this shift from ASIC-based routers to routers with programmable packet processors is that it gives rise to a new class of vulnerabilities and corresponding attacks. Routers based on ASICs represented no practical attack target since their functionality could not be changed except by replacing actual hardware. In contrast, routers based on general-purpose processors that run software to implement packet processing functions exhibit the same kind of vulnerabilities that have been observed and exploited in conventional end-systems and embedded systems: attackers can attempt to crash the system, change its operation, extract information, etc.

Vulnerabilities in the network infrastructure itself are particularly problematic. First, routers are shared infrastructure and outages can affect a large number of users. Second, some routers at the core of the network are connected to links with extremely high data rates (e.g., 40 Gigabits per second). If an attacker can modify the behavior of a router to send out malicious traffic, devastating denial-of-service attacks can be launched using only one or a handful of vulnerable systems. An important open question is if this type of attack is practically feasible in a network, especially since the attacker is limited to only sending data packets.

In our work, we show a practical example of such an attack. Specifically, we demonstrate how benign protocol processing code (in our case, the insertion of a protocol

D. Chasaki and T. Wolf are with the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA, 01003 USA e-mail: {dchasaki, wolf}@ecs.umass.edu

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

Manuscript received Nov 18, 2011; revised May 2, 2012; accepted June 20, 2012.

header) that contains an implementation vulnerability can be exploited by a single data packet and trigger a denial-of-service that consumes the entire outgoing link bandwidth of a router. The attack is based on a stack smashing attack that is triggered by a data packet with a malform User Datagram Protocol (UDP) header. The effect of the attack is the execution of arbitrary code (e.g., an infinite retransmission loop) or a system crash. We show a successful attack for two specific systems, a custom packet processor [2] based on the NetFPGA platform [3] and the Click modular router [4], as representatives for the broad class of routers with programmable packet processors.

The existence of such a data-plane attack leads to the immediate question of how to defend against it. On end-system and server processors, malware detection mechanisms (e.g., virus scanner) have been used to successfully defend against attacks on the processor system. On network processors, where processor cores are much simpler and typically do not support full-blown operating systems, these solutions cannot be used. In our work, we present a hardware-based solution that uses a processor monitor to identify attacks and a recovery system to restore the processor after an attack. We show in our prototype implementation that is based on the NetFPGA platform that detection and recovery can be performed in about a dozen processor cycles. This ability to detect and recover from attacks is critical to ensure that the protection system itself is not susceptible to denial-of-service attacks.

Overall, our paper presents a comprehensive study of this new class of data plane attacks and potential defense mechanisms. Our specific contributions are:

- Discussion of data plane attacks and their emergence due to the use of programmable packet processors in network routers.
- Demonstration (and thus proof of existence) of data plane attacks triggered by a single data packet on two different prototype systems.
- Design of a defense mechanism based on a hardware monitoring and recovery system and evaluation of its performance on a prototype system.

The remainder of the paper is organized as follows. Section 2 discusses related work. We describe the problems arising from programmability in the data plane of networks in Section 3. A specific attack example and results from its implementation are presented in Section 4. Section 5 presents the design and prototype implementation of a defense mechanism against this type of attacks. In Section 6, we evaluate the performance of our prototype system. Finally, Section 7 summarizes and concludes this paper.

2 Related Work

Programmability in the data plane of routers is widely used and many modern routers use programmable packet processors to implement protocol processing. Routers that use software for packet processing include workstation-based routers [4], [5], programmable routers [6], and virtualized router platforms [7]. High-performance router systems use multi-core packet processors (so-called "network processors") [8], [9]. Commercial examples of network processors are the Intel IXP2400 [10], the EZchip NP-3 [11], the LSI APP [12], the Cavium Octeon [13], and the Cisco QuantumFlow [14]. The number of processor cores in these chips ranges from as little as eight in the IXP2400 to over a hundred in the Cisco Silicon Packet Processor (SPP).

Addressing the problem of vulnerabilities in routers is also important in the context of research on the design of the future Internet [15], [16], [17]. The use of programmable packet processors is at the core of many future Internet designs (e.g., network virtualization [7], [18]). Thus, developing defense mechanisms to protect the packet processors in router systems is critical for the continued success of the Internet.

The vast majority of security issues in networking are related to end-systems and protocols. One example is large-scale distributed denial-of-service attacks, which are generated by botnets [19]. Widely deployed intrusion prevention systems including firewalls [20] and deep packet inspection [21], are trying to control end-system intrusion and thus to limit the access to platforms from which attacks can be launched. Secure protocols (e.g., IPsec [22]) are used to provide basic information security, including authentication and privacy.

Very little work has addressed security issues in the network infrastructure itself. A recent study [23] surveyed network devices that are considered vulnerable due to exposed administrative interfaces, which are part of the control plane of the network and can be protected by better management methods. In our work, we consider the data plane of the network, which inherently needs to be exposed and thus needs novel protection techniques. One such protection is based on processor monitoring, originally proposed for embedded systems in general [24] and recently adapted for network systems in our prior work [25]. Other defenses may be based on techniques from embedded system security [26].

Software vulnerabilities have been studied extensively on a range of different systems. For programmable routers based on Click, the integer vulnerability exploited in this paper effectively leads to a buffer overflow attack on the host operating system. Although there have been many attempts to tackle this problem statically [27], [28] and dynamically [29], [30], state-of-the-art attack prevention mechanisms lack the ability to adjust the execution flow at runtime and lead to termination of the packet processing task.

Large scale DoS attacks have been previously studied in the context of worms [31]. Worms can spread quickly by infecting a large number of vulnerable end-systems and can absorb a large amount of network bandwidth. The key difference of the attack that we describe in this paper is that it has an even more devastating effect:

The attack is triggered with *a single packet*, absorbs *all bandwidth* of the outgoing link on the router, and can *propagate* to all vulnerable downstream routers. Some initial ideas of our work were published in our prior work [25], [32].

3 Vulnerabilities and Attacks in Network Infrastructure

Before discussing the details of a specific attack and defense mechanisms in the following sections, we provide a brief overview of the vulnerabilities and potential attacks in the network infrastructure.

3.1 Attack Classification

The main functionality of the Internet (and any other data communication network) is to allow end-systems to communicate. As such, the Internet has served as a vehicle for many attacks where malicious users have gained unauthorized access to end-systems for the purpose of hacking, espionage, etc. In addition to such attacks that target access to data on end-systems, there are also denial-of-service attacks that aim to make end-systems temporarily inaccessible. While attacks on end-systems are often highly visible due to news media attention, there are are also several other types of attacks on other components of the network. These attack types are shown in Figure 1 together with a few examples and common defense mechanisms. This figure by no means contains a comprehensive list of attacks and defenses, but merely a selection that helps in illustrating major differences in attack types. The control plane of the network, where routing information and other control information is exchanged, is a target of attacks that aim to disrupt the correct operation of the network (e.g., by stealthily redirecting traffic to malicious end-systems). In the data plane of the network, where the actual network traffic is transmitted between end-systems and routers, attackers may aim to eavesdrop on or intercept communications. It is here where a new type of attack that can lead to denial-of-service is emerging.

The attack on the data plane of the network that aims at denial of service is the main focus of this paper. As explained in the introduction, this attack is rooted in the way modern routers are implemented. Until a few years ago, practically all high-performance routers used ASICs to implement packet forwarding operations. The function of an ASIC cannot be changed after it has been created and thus there was no way to change the forwarding operation of a router for the purpose of a network attack. However, the recent development of high-performance MPSoCs that are specialized for packet processing (i.e., network processors) has shifted router designs from ASIC-based packet forwarding to software-based forwarding on general-purpose processing systems [1], [13], [14], [33]. As with any software-based system, the flexibility provided by programmability also presents a security challenge as attackers can change the operation of the system for malicious purposes.

3.2 Commercial Routers with Programmable Features

Since attacks on the data plane of networks hinge on the presence of programmable packet processing systems in routers, we provide a brief overview of commercial router products. We show that programmable packet processors are indeed widely used in the Internet. Thus, these routers present the potential for the types of attacks we discuss in this paper. However, we do not want to imply that any specific products are vulnerable to any specific attack. We merely want to show that the technology that is the basis for these attacks (i.e., programmable packet processors) is commonly used.

Modern Internet routers for the network core and the network edge typically employ programmable packet processors. This trend can be seen by examining the router products from two of the leading router equipment manufactures:

- Cisco: The Cisco CRS-1 core router uses a Cisco Silicon Packet Processor (SPP). The SPP is a 188-core processor that can be programmed to execute multiple advanced networking services. Cisco CRS-3 is based on the 40-core Cisco Quantum Flow processor which also supports high levels of parallelism and flexibility in terms of implementable services. Edge routers like the Cisco ASR 1000 and 9000 series are built on Quantum Flow as well. They are commonly used for applications that require parallel processing, security provisioning, QoS mechanisms and virtualization. The processors support any combination of layer 2 and layer 3 services and features (which can be programmed in C, similar to how we program our prototype system).
- Juniper: Juniper J and SRX series as well as Juniper MX series are used by both service providers and enterprise networks. They are based on network processors (e.g., Intel IXP models, which can efficiently implement services like load balancing, SSL offload, web acceleration, application security, access control etc.).

While there are many more network equipment vendors, it is clear that programmability in the data plane is commonly used.

3.3 Security Model for Network Infrastructure

In our work, we use a straightforward security model that reflects the operation of current Internet. Basically, we assume that the packet processing code on a router is benign (i.e., not intentionally malicious) and an attacker aims to exploit vulnerabilities in this code to change the operation of a router.

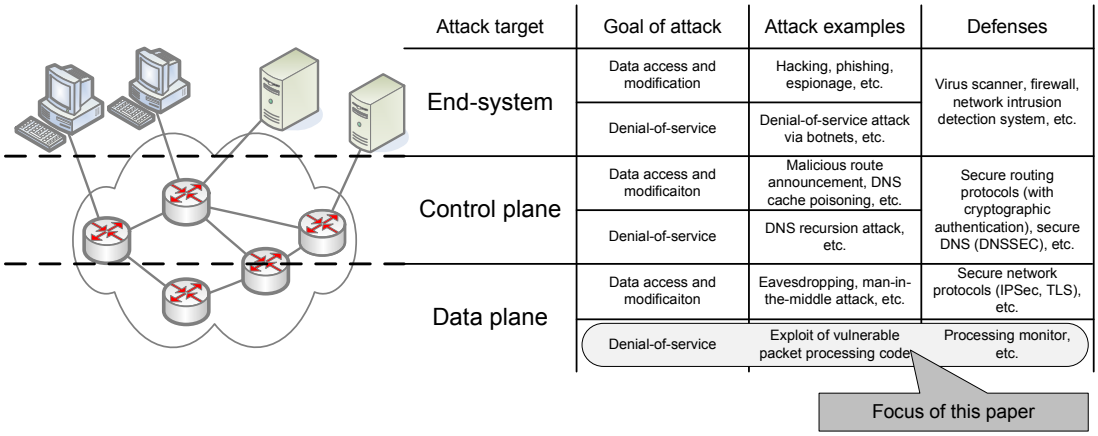


Fig. 1. Examples of network attacks and defenses.

3.3.1 Security Requirements

The basic security requirements for a router are twofold: (1) Normal (i.e., benign) network traffic should be processed according to network protocol specifications; (2) Malicious network traffic should not have a negative effect on the router (and can be discarded). These requirements imply that regular traffic should be processed by the router even in the presence of attack traffic. That is, attack traffic should not have a negative impact on the router system.

Note that the infrastructure attacks we discuss here (see Figure 1) rely on the ability of an attacker to change the behavior of the packet processor (i.e., change in control flow or instruction memory) or its data (i.e., change in data memory). If it can be ensured that the operation of the router does not change from what it originally was programmed to do, then an attack is not possible.

We show in Section 4 how an attacker can violate security requirement (1) by changing the operation of the router. In Section 5, we then show how processing monitoring can enforce requirement (2) and thus avoid a change in router operation thereby circumventing the problems caused by attack traffic.

3.3.2 Attacker Capabilities

In our work, we consider the following attacker capabilities. An attacker can:

- 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.

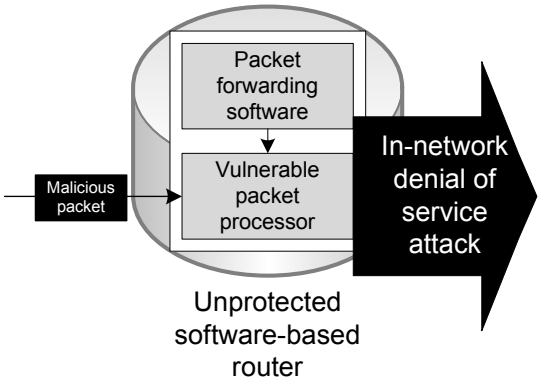


Fig. 2. Example of in-network attack. Vulnerable packet processing systems on routers can be used to launch large-scale denial-of-service attacks with a single packet.

The attacks mentioned in this section is 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.

Based on this security model, we present a specific example of an attack.

4 Data Plane Attack Example

The main idea of the attack is illustrated in Figure 2. A cleverly crafted packet may be able to exploit software vulnerabilities (e.g., stack smashing attack) and change the operation of the packet processor. A simple change in the software could lead to an infinite loop where the same packet is transmitted repeatedly. Such an approach is particularly effective and damaging since the attack originates from within the network, where the compromised system may have access to links with tens of Gigabits per second bandwidth.

To describe the attack in detail, we briefly discuss the code vulnerability that we exploit, as specific example code that exploits this vulnerability in the context of protocol processing, and an example data packet that triggers an exploit of the vulnerability.

4.1 Vulnerability

Our attack exploits a vulnerability in the program that is executed on the packet processor of the routers. There are known C/C++ code exploits such as pointer subterfuge, use of `strcpy` and `memcpy` for buffer overflows, and integer vulnerabilities. A large number of them is present in commercial software designs and implementations. These vulnerabilities, under certain conditions, can be exploited by attackers, especially if programmers are not writing security-aware code.

The premise of our attack is that the packet processing code is benign and does not contain intentionally malicious code. The attacker sends a carefully crafted packet to one of the router's network interface cards. The processing of this packet turns the 'good' code/protocol routine that runs on the network processor into 'bad' code. There is nothing inherently wrong with the packet or the application code, but the combination of the two can lead to the processor's malfunctioning. In our case, the incoming packet changes the control flow of the routing and redirects it to malicious code that resides inside the payload of the attack packet. For all other packets, the correct processing is performed by the router.

The specific exploit we use in our attack is an integer vulnerability. Certain integer arithmetic operations, depending on the conditions, can result to unexpected outcome. Sign errors, truncation errors, integer overflows or underflows can occur, which, if not taken into account before the program execution, can lead to programs with unexpected behavior and security flaws [34].

Our attack is based on a vulnerability caused by an integer overflow. As we know, integers can represent values within a given range. For example, the integer type 'unsigned short' ranges from 0 to 65535. When a variable declared as short integer exceeds the upper limit, the assigned value wraps around zero in order to stay within the allowed limits. If the programmer does not anticipate this behavior, and the remaining of the program reuses that value at some point, potentially harmful things can happen. The following example code contains an integer overflow vulnerability:

```
unsigned short sum;
unsigned short one = 65532;
unsigned short two = 8;
sum = one + two;
```

The value assigned to the variable `sum` is not 65540, as one would expect, but 4 due to the limited amount of memory space that is assigned to it.

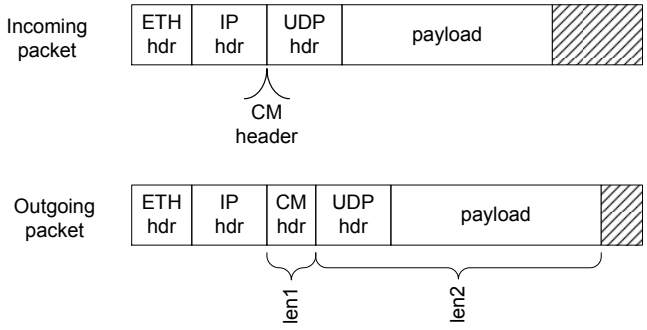


Fig. 3. Protocol Header Insertion.

4.2 Vulnerable Protocol Processing Code

Routers perform a variety of protocol processing operations, ranging from simple IP forwarding to more advanced functions that include IPsec termination, intrusion detection, tunneling, etc. For our attack example, we assume that the protocol processing operation consists of adding a header to a packet. We are describing this operation in the context of the congestion management protocol described in [35] to be concrete, but it is important to note that the vulnerability can apply to a much broader range of protocol operations that add packet headers.

The congestion management (CM) protocol uses a custom protocol header that is inserted between the IP header and the UDP header. This process is illustrated in Figure 3. For the discussion of our attack, the detailed operation of the CM protocol and its header format is irrelevant. The important aspect is that CM adds a header in a packet.

The processing steps associated with the header insertion by the CM protocol are:

1) Parse headers to identify header boundary between IP and UDP.
2) Shift the UDP header (and higher layer headers and payload) to the right to make room for the CM header.
3) Insert CM header in packet.

Figure 4 shows pseudocode for the part of the program that inserts the new CM header in the original packet, which is the part of the code that contains a vulnerability. While writing the CM header generation part of the protocol, a security aware programmer would perform a check on the packet's total size before shifting the UDP datagram and inserting the new header into the original packet. This check is making sure that the outgoing packet – after the 12-byte CM header is appended to it – does not exceed the maximum datagram size. Only if the check `(CM_hdr_size + UDP_length) < MAX_PKT` passes, the original UDP datagram gets shifted by 12 bytes, and the CM header followed by the original UDP datagram are copied into the new packet buffer. The following line is the one that performs the shift and copy operation: `memcpy((new_pkt_buf+len1), original_pkt, len2);` where `len1` is the CM

```
#define MAX_PKT 1484

int generate_CM_header(int orig_pkt[], unsigned
short len1, unsigned short len2)
{

        int new_pkt_buf[MAX_PKT];

        unsigned short sum;
        sum= len1+ len2;
        if(sum > MAX_PKT) { return -1;}
        else {
        memcpy((new_pkt_buf+len1), orig_pkt, len2);


        ...

        return 0;
}

int main(int argc, char **argv)
{
        int orig_pkt[];

        ...

        generate_CM_header(orig_pkt, CM_hdr_size,
UDP_length);

        ...

}
```

Fig. 4. Example Application Code.

header size (12 bytes) and `len2` is the UDP datagram's total length. Since the total length field of the UDP header is a 16-bit field and the CM header is only 12 bytes long, the programmer could choose to assign `len1` and `len2` to 'unsigned short' integer types, so that the embedded processor's limited resources are not wasted.

This code, while correct for CM protocol processing, contains a vulnerability that is based on an integer overflow in the length check. A carefully crafted attack packet can exploit this vulnerability.

4.3 Attack Packet

The vulnerability does not exhibit problematic behavior for most "normal" packets that are short enough to accommodate the 12-byte CM header within the maximum IP packet length. An attacker, on the other hand, can send a long UDP packet that triggers an overflow. If an attacker chooses to send a regular, oversized packet (larger than `MAX_PKT`), the size check will fail. However, if an attacker sends a packet with a malformed UDP length field (for example with the 16-bit value 0xfffc (65532 in decimal)), then the code performs incorrectly:

1) `CM_hdr_size + UDP_length = 12 + 65532 = 8` (incorrect due to integer overflow)
2) `CM_hdr_size + UDP_length < MAX_PKT` (even though it is not)
3) 65532 bytes are copied into the `new_pkt_buf`, which can only accommodate 1484 bytes

Due to the malformed UDP total length field of the incoming packet, the processing of the protocol code

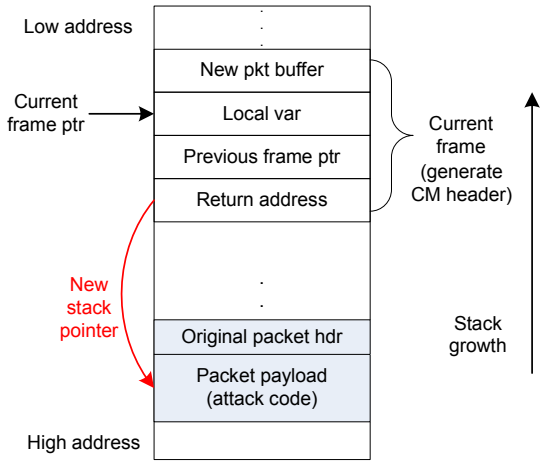


Fig. 5. Stack Smashing.

leads to unexpected program behavior. A large amount of data ends up being copied into a buffer that was not designed to handle more than the maximum datagram size. The result is the notorious buffer overflow attack, which will overwrite the processor's stack.

Figure 5 shows the stack of the processor when the function `generate_CM_header` is running. We can see the original incoming packet residing in the bottom of the stack, as part of the main function. The last few bytes of the original packet correspond to the payload and contain the attack code, which the attacker has devised. Once the function `generate_CM_header` starts processing the incoming packet with the malformed UDP length field, the new packet buffer will overflow and start rewriting the local variables of the current frame, continue with the stack pointer and finally overwrite the return address of the current frame as well. Originally, the program should have jumped back to the calling function after finishing with the CM header generation, but when the return address is overwritten, the program will jump to whichever address the attacker has chosen! Of course, the attacker chooses to overwrite the return address with the stack memory address where the attack code begins. Thereby, the attacker can make the program jump to malicious code that is carried inside the packet payload.

In our attack, we insert a few instructions of assembly code into the payload, which repeatedly broadcast the same attack packet in an infinite loop. As we show in Section 4.4, a single attack packet of this type triggers a denial-of-service attack that jams the routers outgoing link at full data rate. While our attack is used for launching a denial-of-service service attack, it should be noted that an attacker could choose to run attack code with other purposes.

With this example, we demonstrate that vulnerabilities in software-based routers are not only hypothetical, but can occur in common protocol processing code. We also show that these vulnerabilities can be exploited to

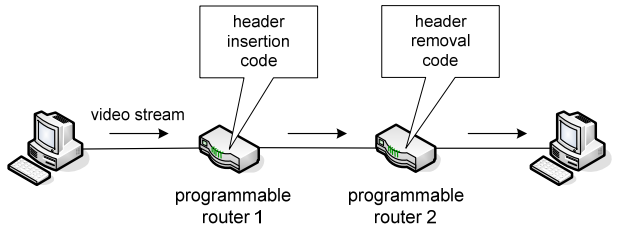


Fig. 6. Experimental Setup.

execute arbitrary attack code.

4.4 Results

To demonstrate the feasibility and effects of the attack described in Section 4, we show a prototype implementation in a real network. We have implemented the attack on the Click modular router [4] and on a custom packet processor [2] based on the NetFPGA platform [3]. The custom packet processor uses a Plasma soft core [36], which is a 32-bit MIPS architecture processor.

Our experimental setup is shown in Figure 6. We send video traffic from one end-system to the other over a network consisting of two routers. The first router implements the CM header insertion processing described above. The second router removes the CM header. The header insertion routine on the first router is implemented as discussed in Section 4 and exhibits the integer overflow vulnerability.

We measured the incoming and outgoing traffic on the first router for different scenarios. Figure 7 shows the results for benign traffic and attack traffic. There is a 30-second video transmission as baseline traffic (shown in green). In the first scenario (Figure 7(a)), only benign traffic is sent and the router forwards it as expected. In the second scenario (Figure 7(b)), a single attack packet is injected into the incoming traffic of our custom network processor. Since the attack packet triggers an infinite loop of retransmitting itself, all output traffic consists of attack traffic (shown in red). There are two important observations: (1) No benign traffic is forwarded. Thus, the attack not only absorbs all unused bandwidth, but *all* bandwidth. (2) The amount of outgoing attack traffic is around 850 Mbps, which is close to the total link rate of the system. The performance of the system is limited to 850 Mbps due to the maximum clock frequency in our prototype. In a commercial high-performance router, attack traffic would be sent at the full link rate.

We also demonstrate a denial-of-service attack with Click. Due to the integer vulnerability, the memory copy will exceed the pre-determined buffer boundary and overwrite adjacent memory content (e.g. variables, function pointers). In our experimental setup (kernel version 2.6.19.2), Click runs as a user process and cross-boundary write causes a runtime exception that leads to termination of the process. (We demonstrate a different attack scenario from that on the customized packet processor. The denial-of-service in this case consists of shutting down all packet forwarding in the router.) Figure 7(c) shows the attack scenario. An attack packet is sent to the router and effectively interrupts all packet processing services provided by Click.

These results very clearly show that the attack we describe in this paper is indeed possible in practice and that it has devastating effects on the network by generating attack traffic at full link rates in the core of the network.

4.5 Harvard Architecture and Executable Space Protection

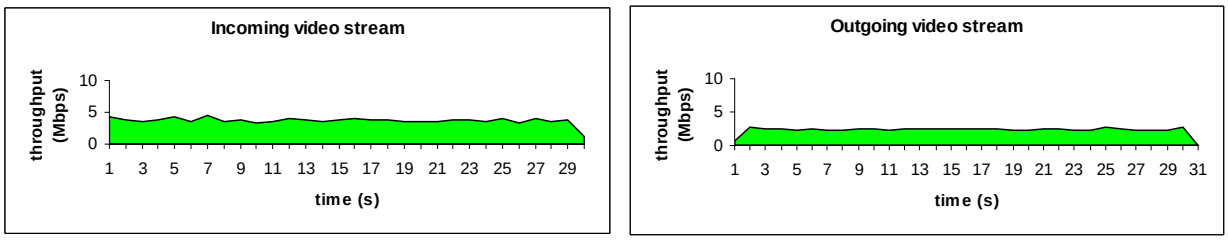
The above attack example is based on a processor with a von Neumann architecture, where program code and data can reside in the same memory. This memory architecture allows program code in the packet data to be executed. While von Neumann architectures are commonly used in general purpose processors, many existing network processors are based on the Harvard architecture, which uses separate data and instruction memories. While the above attack cannot immediately be applied to a Harvard architecture processor, we want to note that similar attacks are possible in such architectures.

The separation of instruction memory and data memory can be achieved physically by using separate memories or logically by reserving some pages of a combined memory for instruction code (e.g., using a no-execute (NX) or execute disable (XD) bit or a write-xor-execute (W$\oplus$E) approach [37]). In network processors, separate memories are typically used since the latter approach requires advanced memory management and operating system support. Separation of instruction and data memories, however, does not protect a system from attacks. Processor stacks are still vulnerable to smashing attacks, which can be used to redirect control flow into program code that is already installed in the system (e.g., return-into-libc attacks [38]). While these attacks are limited to utilizing code that already exists on the system, they have been shown to be Turing complete [39]. Clearly, network processors are simpler than systems with operating system libraries, but the potential for these types of attacks still exists.

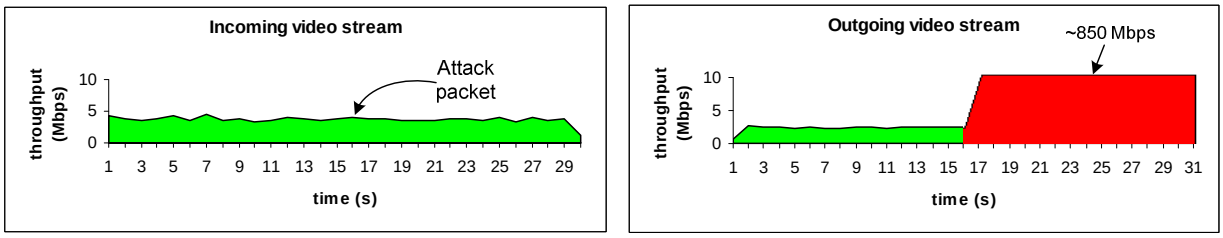
Thus, network processors with separate instruction and data memories are not fundamentally less vulnerable than conventional processors. Both types of systems can be attacked through the data plane and can benefit from the protection mechanisms developed in our work.

5 Defense Mechanisms against Data Plane Attacks

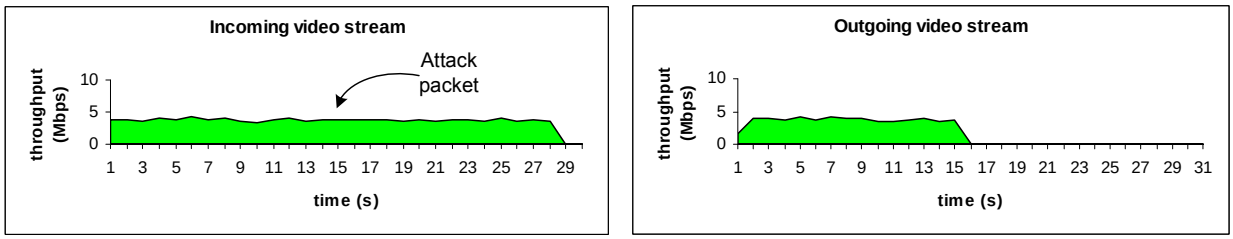
As we have shown in the previous section, attack on packet processing systems through the data plane of the network are possible. Typical defense mechanisms, such as separation of instruction and data memory or



(a) Benign network traffic



(b) Benign traffic and single attack packet on custom network processor



(c) Benign traffic and single attack packet on Click modular router

Fig. 7. Traffic Rates at Input Port and Output Port of Vulnerable Router. Benign video traffic is shown in green, attack traffic is shown in red.

executable space protection can be circumvented. Other techniques, such as memory layout randomization or stack cookies, require operating system support. Packet processors, which are designed for high throughput and therefore use many relatively simple processor cores, typically cannot support operating systems and thus these techniques cannot be easily employed.

In our work, we design a protection mechanism that can be implemented in hardware and co-located with a packet processor [25] and thus is suitable for embedded processing systems [26]. Using hardware ensures that the performance impact on packet processing is minimal. Also, using a hardware defense that is not (or not easily) accessible to an attacker can ensure that the protection mechanism cannot be not tampered with.

5.1 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

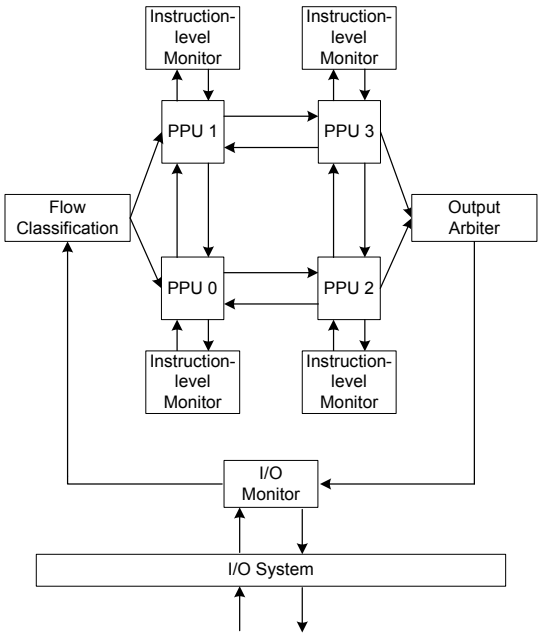


Fig. 8. Security Monitoring on the Packet Processor.

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.

5.2 System Design

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 packet processor prototype described in [2].

Figure 8 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 [2] next generation routers are expected to have multiple packet processing units (PPU), in order to 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, 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.

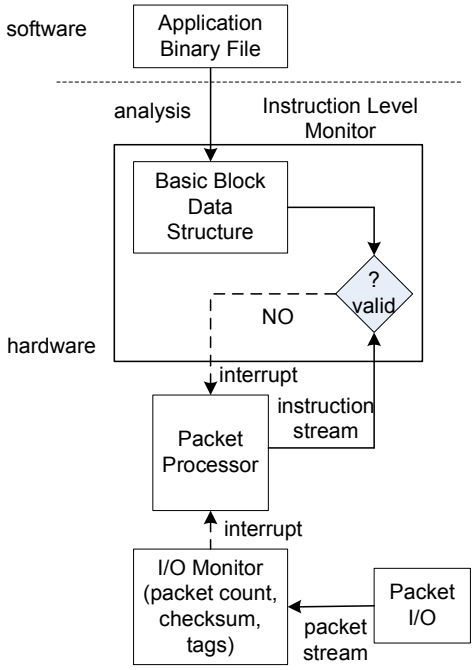


Fig. 9. Monitoring System Overview.

5.2.1 Monitoring

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 monitor's 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 9.

- 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 attack.

5.2.2 Recovery

In case an attack is detected by the monitor, the processing of the current packet is terminated. After dropping the packet, it is necessary to reset the processor to a state where it can continue processing other packets. Since attacks are typically based on changes in program execution, it is necessary to reset the processing stack and instruction store. The recovery process includes the following steps:

- The packet buffer where the current packet is stored is cleared to receive a new packet.
- The processor core that is processing the offending packet is reset: The processor stack and registers are reset to recover from any tampering with the stack pointer.
- The instruction memory is reset, so that potentially harmful code is overwritten and does not affect future packets to be processed on this core.

For the last step, the correct protocol processing code is reloaded from on-chip memory, which contains the instruction memory initialization values. That storage place is assumed to be secure and not accessible by the attacker.

Once the recovery process has been completed, the processor core continues to process the next packet. One aspect of this recovery process that is crucial is that it can be performed quickly. If recovery takes the processor core offline for a long time, a denial of service attack can be launched by sending multiple attack packets that keep the router system unavailable due to continued recovery. The evaluation results from our prototype system show that this is not the case (see Section 6.2).

5.3 Prototype Implementation

We have implemented a prototype system to demonstrate the functionality of our design. The proof-of-concept implementation of a packet processing system uses the two security techniques we have described above. Our prototype is implemented on a NetFPGA [3], 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.

5.3.1 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 9 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 10. Each pipeline stage takes one clock cycle to complete.

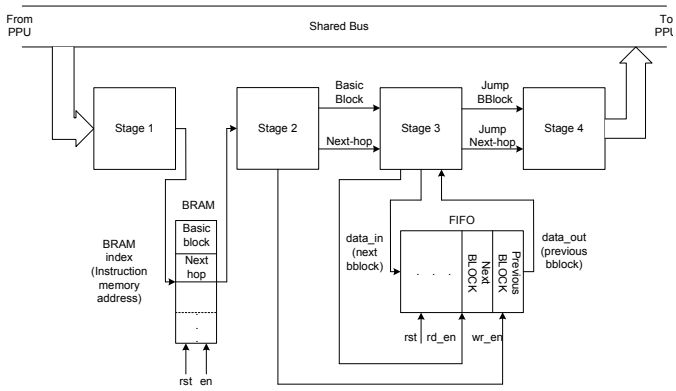


Fig. 10. Instruction Level Monitor Design.

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 stage, the monitor sends the packet drop signal to the packet memory unit, which stops the processing of the current packet. The corresponding packet buffer drops this packet and is ready to receive a new one. At the same time the instruction memory is reset, so that the harmful code is overwritten and does not affect the next packet to be processed by the packet processor. While resetting the instruction memory, we can first switch to a backup piece of 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.

5.3.2 I/O Monitor

The I/O monitor in this prototype implementation verifies the correct operation of the packet processor from the perspective of the entire system. Instead of monitoring individual processor instructions, it keeps track of the packets that have entered and left the packet processing system. At this level of granularity, incorrect system operation can be detected even if processing operations may be correct. For example, if there is no multicast traffic, but more packets leave the system than have entered it, then there is a problem with the packet processor. In our system, the I/O monitor maintains packet counts and triggers if more packets are sent than are received. More complex metrics can be used (e.g., packet processing delay [40]), but these are beyond the scope of this paper.

6 Evaluation

We demonstrate a successful data plane attack in Section 4.4. In this section, we provide evaluation results from our prototype system to demonstrate the correct functionality of the security features that can detect and recover from this attack. We also discuss resource utilization and performance results, specifically the delay in detecting attacks and the system's throughput.

6.1 Security Monitor Operation

In Figure 12, 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 normal 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 [2].

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 10 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

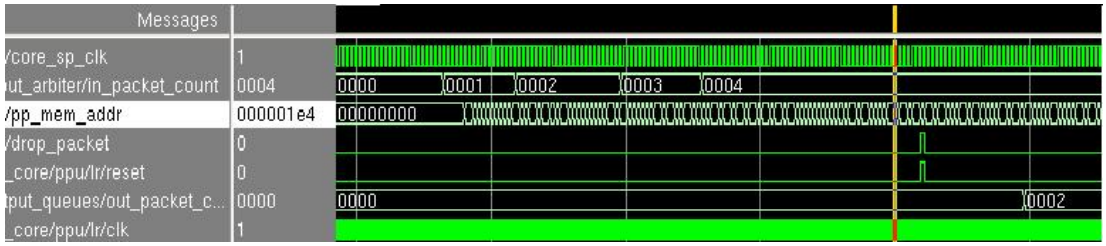


Fig. 11. Simulation Results.

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 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 11 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.

6.2 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

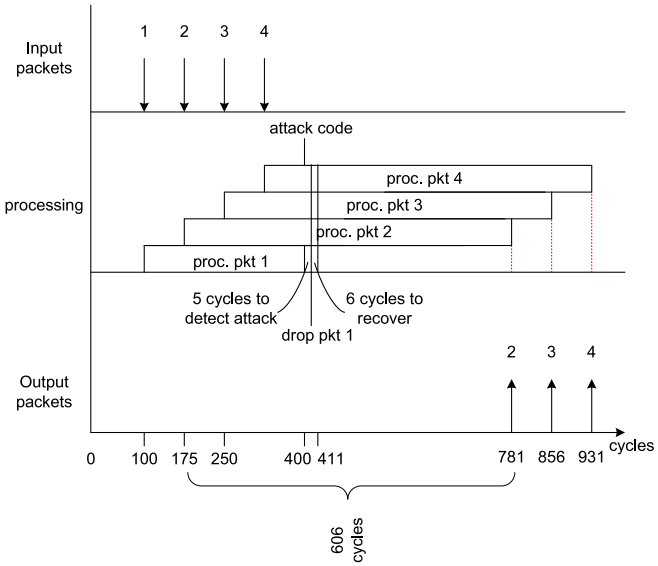


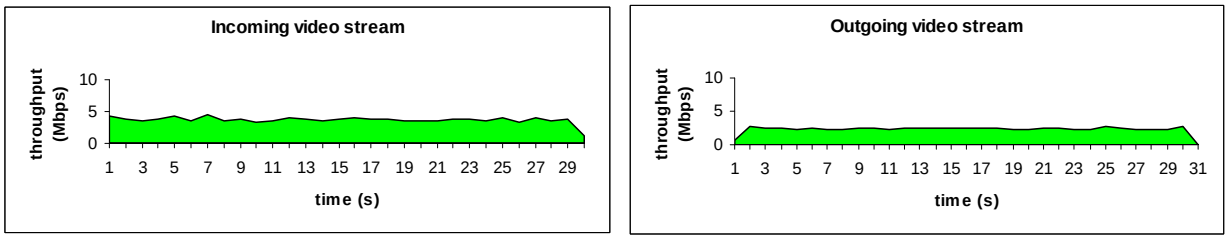
Fig. 12. Security Monitor Operation.

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.

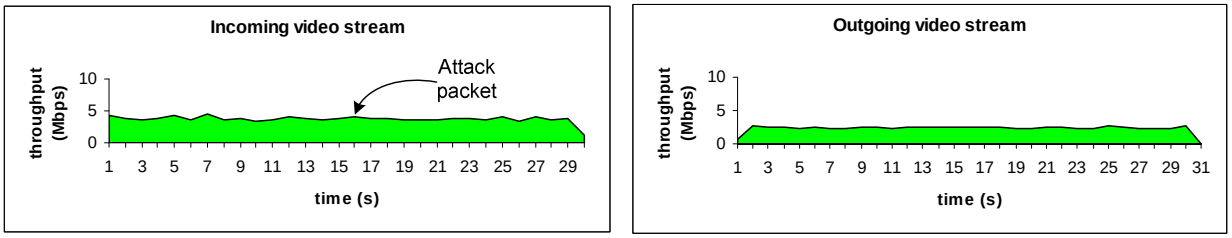
Here, we performed an experiment to measure the throughput of our packet processing system when the two security monitors are on: We have three working Ethernet ports on our prototype packet processing system. We setup a testbed by connecting the one of the NetFPGA ports to another machine, on which we measure throughput. First, we experiment by sending valid packets only, and then by sending a combination of valid and invalid packets. We did not notice any changes in the processor's throughput. We can still achieve high data rates (maximum of 100 Mbps). In Table 1, we report the average throughput numbers. 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 100Mbps because the experiment was performed by forwarding small packets.

TABLE 1
Resource consumption and performance.

	Single core	Single core w/ main monitor	Single core w/ 2 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(Mbps)	67.2	64.1	63.9



(a) Benign network traffic



(b) Benign traffic and single attack packet

Fig. 13. Traffic Rates at Input Port and Output Port of Router with Processing Monitor.

In the case of large packets the packet processor can achieve throughput greater than 2 Gbps [2].

As for 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.

6.3 Demonstration of Real Attack Detection

The final result shows how our system responds under a real attack scenario, like the one described in Section 4. We have experimented on the same custom processor used for the setup shown in Figure 6. The prototype successfully detects the example attack (and any other attack that changes the control flow), halts the processor, drops the packet, and restores 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).

Figure 13 shows the operation of the secure packet processor under attack. As can be seen, not only does the processor not fall victim to the attack, but it also continues to forward regular traffic without interruption.

While the results from our secure packet processor are encouraging by demonstrating that there are defenses against the types of attacks that we describe in this paper, it is important to note that such defenses are not currently deployed in the Internet. Existing software-based routers are still vulnerable and more research and development is necessary to design and deploy defenses against this novel type of attack.

7 Summary and Conclusion

In this paper, we describe and demonstrate a novel type of network attack. The attack exploits vulnerabilities in the packet processing systems of modern routers. We show how integer vulnerabilities in the implementation of a common protocol processing operation can be used to execute arbitrary attack code. Our attack can be used to launch devastating denial-of-service attacks in the core of the network. We show that defense mechanisms do exist, but they are not currently deployed in 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.

REFERENCES

[1] W. Eatherton, "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.

[2] Q. Wu, D. Chasaki, and T. Wolf, "Implementation of a simplified network processor," in *Proc. of IEEE International Conference on High Performance Switching and Routing (HPSR)*, Richardson, TX, Jun. 2010, pp. 7–13.

[3] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo, "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, Jun. 2007, pp. 160–161.

[4] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, Aug. 2000.

[5] N. C. Hutchinson and L. L. Peterson, "The x-kernel: An architecture for implementing network protocols," *IEEE Transactions on Software Engineering*, vol. 17, no. 1, pp. 64–76, Jan. 1991.

[6] L. Ruf, K. Farkas, H. Hug, and B. Plattner, "Network services on service extensible routers," in *Proc. of Seventh Annual International Working Conference on Active Networking (IWAN 2005)*, Sophia Antipolis, France, Nov. 2005.

[7] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the Internet impasse through virtualization," *Computer*, vol. 38, no. 4, pp. 34–41, Apr. 2005.

[8] J. S. Turner, P. Crowley, J. DeHart, A. Freestone, B. Heller, F. Kuhns, S. Kumar, J. Lockwood, J. Lu, M. Wilson, C. Wiseman, and D. Zar, "Supercharging PlanetLab: a high performance, multi-application, overlay network platform," in *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, Kyoto, Japan, Aug. 2007, pp. 85–96.

[9] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In VINI veritas: realistic and controlled network experimentation," in *SIGCOMM '06: Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Pisa, Italy, Aug. 2006, pp. 3–14.

[10] *Intel Second Generation Network Processor*, Intel Corporation, 2005, http://www.intel.com/design/network/products/npfamily/.

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

[12] *APP3300 Family of Advanced Communication Processors*, LSI Corporation, Aug. 2007, http://www.lsi.com/.

[13] *OCTEON Plus CN58XX 4 to 16-Core MIPS64-Based SoCs*, Cavium Networks, Mountain View, CA, 2008.

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

[15] A. Feldmann, "Internet clean-slate design: what and why?" *SIGCOMM Computer Communication Review*, vol. 37, no. 3, pp. 59–64, Jul. 2007.

[16] *Future INternet Design*, National Science Foundation, http://www.nets-find.net/.

[17] *Global Environment for Network Innovation*, National Science Foundation, http://www.geni.net/.

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

[19] D. Geer, "Malicious bots threaten network security," *Computer*, vol. 38, no. 1, pp. 18–20, 2005.

[20] J. C. Mogul, "Simple and flexible datagram access controls for UNIX-based gateways," in *USENIX Conference Proceedings*, Baltimore, MD, Jun. 1989, pp. 203–221.

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

[22] S. Kent and R. Atkinson, "Security architecture for the Internet protocol," Network Working Group, RFC 2401, Nov. 1998.

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

[24] S. Mao and T. Wolf, "Hardware support for secure processing in embedded systems," *IEEE Transactions on Computers*, vol. 59, no. 6, pp. 847–854, Jun. 2010.

[25] D. Chasaki and T. Wolf, "Design of a secure packet processor," in *Proc. of ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS)*, San Diego, CA, Oct. 2010.

[26] S. Parameswaran and T. Wolf, "Embedded systems security – an overview," *Design Automation for Embedded Systems*, vol. 12, no. 3, pp. 173–183, Sep. 2008.

[27] E. Haugh and M. Bishop, "Testing C programs for buffer overflow vulnerabilities," in *Proc. of the Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2003.

[28] T.-C. Chiueh and F.-H. Hsu, "Rad: a compile-time solution to buffer overflow attacks," in *Proc. of 21st International Conference on Distributed Computing Systems (ICDSC)*, Apr. 2001, pp. 409–417.

[29] K.-s. Lhee and S. J. Chapin, "Type-assisted dynamic buffer overflow detection," in *Proceedings of the 11th USENIX Security Symposium*, San Francisco, CA, Aug. 2002, pp. 81–88.

[30] J. Wilander and M. Kamkar, "A comparison of publicly available tools for dynamic buffer overflow prevention," in *Proc. of the Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2003.

[31] D. Moore, C. Shannon, and J. Brown, "Code-Red: a case study on the spread and victims of an Internet worm," in *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement*, Marseille, France, Nov. 2002, pp. 273–284.

[32] D. Chasaki, Q. Wu, and T. Wolf, "Attacks on network infrastructure," in *Proc. of Twentieth IEEE International Conference on Computer Communications and Networks (ICCCN)*, Maui, HI, Aug. 2011.

[33] T. Wolf, "Challenges and applications for network-processor-based programmable routers," in *Proc. of IEEE Sarnoff Symposium*, Princeton, NJ, Mar. 2006.

[34] R. C. Seacord, *Secure Coding in C and C++*, 1st ed. Addison-Wesley Professional, 2005.

[35] H. Balakrishnan, H. S. Rahul, and S. Seshan, "An integrated congestion management architecture for internet hosts," in *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication (SIGCOMM)*, Cambridge, MA, Sep. 1999, pp. 175–187.

[36] S. Rhoads, *Plasma – most MIPS I(TM) Opcodes*, 2001, http://www.opencores.org/project,plasma.

[37] *The OpenBSD 3.3 Release*, OpenBSD, http://www.openbsd.org/33.html, May 2003.

[38] H. Shacham, "The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86)," in *Proc. of the 14th ACM Conference on Computer and Communications Security (CCS)*, Alexandria, VA, Oct. 2007, pp. 552–561.

[39] M. Tran, M. Etheridge, T. Bletsch, X. Jiang, V. Freeh, and P. Ning, "On the expressiveness of return-into-libc attacks," in *Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science, Sep. 2011, vol. 6961, pp. 121–141.

[40] D. Chasaki, Q. Wu, and T. Wolf, "Inferring packet processing behavior using input/output monitors," in *Proc. of ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS)*, Brooklyn, NY, Oct. 2011, pp. 91–92.

Danai Chasaki (S'11) is an doctoral student in the Department of Electrical and Computer Engineering at the University of Massachusetts Amherst. Her research interests are in the area of networked embedded systems.

Tilman Wolf (M'02-SM'07) is an associate professor in the Department of Electrical and Computer Engineering at the University of Massachusetts Amherst. He received a D.Sc. in computer science in 2002, all from Washington University in St. Louis. His research interests include network processors, their application in next-generation Internet architectures, and embedded system security.