Attacks on Network Infrastructure

Danai Chasaki*, Qiang Wu† and Tilman Wolf*
*Department of Electrical and Computer Engineering
University of Massachusetts, Amherst, MA, USA
Email: {dchasaki,wolf}@ecs.umass.edu
†Juniper Networks, Inc.
Sunnyvale, CA, USA

***Abstract*—We present the first practical example of an entirely new class of network attacks – attacks that target the network infrastructure. Modern routers in computer networks use general-purpose programmable packet processors. The software used for packet processing on these systems is potentially vulnerable to remote exploits. In this paper, we demonstrate a specific attack that can launch a devastating denial-of-service attack by sending just a single packet. We show that vulnerable packet processing code can be exploited on a Click modular router as well as on a custom packet processor on the NetFPGA platform. We also show that defense techniques based on processor monitoring that we have proposed in prior work can help in detecting and avoiding such attacks.**

***Index Terms*—network security, network attack, programmable router, network processor, processing monitor**

I. 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 demonstrate a specific example of a novel type of attack that exploits these vulnerabilities and thereby attacks the network infrastructure itself.

In the past, most high-performance network routers used application-specific integrated circuits (ASICs) to implement packet forwarding functions. While ASICs are costly to develop, they represented the only technology that could 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 general-purpose processor provides a router vendor with much more flexibility to adjust a router's functionality after production [1]. Therefore, there is an ongoing shift in the industry toward developing 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.

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 header) can be exploited by a single data packet and trigger a denial-of-service that consumes the entire outgoing link bandwidth of a router. We show this vulnerability for two specific systems, a Click modular router [2] and a custom packet processor [3] based on the NetFPGA platform [4], as representatives for the broad class of routers with programmable packet processors. We also show that processor monitoring techniques developed in prior and related work [5] can help in identifying and mitigating these attacks.

The specific contributions of our paper are:

- To our knowledge, the first practical example of a novel type of attack on routers with programmable packet processors,
- A prototype implementation and evaluation of the attack on a Click modular router system and on a custom network processor based on the NetFPGA platform, and
- A discussion of mitigation techniques, including previously proposed processor monitors.

The remainder of the paper is organized as follows. Section II discusses related work. We describe the problems arising from programmability in the data plane of networks in Section III. A specific attack example is presented in Section IV.

Results from two implementations of the attack in a real network setup are shown in Section V. Section VI presents a discussion of defense mechanisms against this type of attacks. Section VII summarizes and concludes this paper.

II. 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 [2], [6], programmable routers [7], and virtualized router platforms [8]. High-performance router systems use multi-core packet processors (so-called "network processors") [9], [10]. Commercial examples of network processors are the Intel IXP2400 [11], the EZchip NP-3 [12], the LSI APP [13], the Cavium Octeon [14], and the Cisco QuantumFlow [15]. 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 [16]–[18]. The use of programmable packet processors is at the core of many future Internet designs (e.g., network virtualization [8], [19]). 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 [20]. Widely deployed intrusion prevention systems including firewalls [21] and deep packet inspection [22], 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 [23]) 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 [24] 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 [25] and recently adapted for network systems in our prior work [5]. 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.

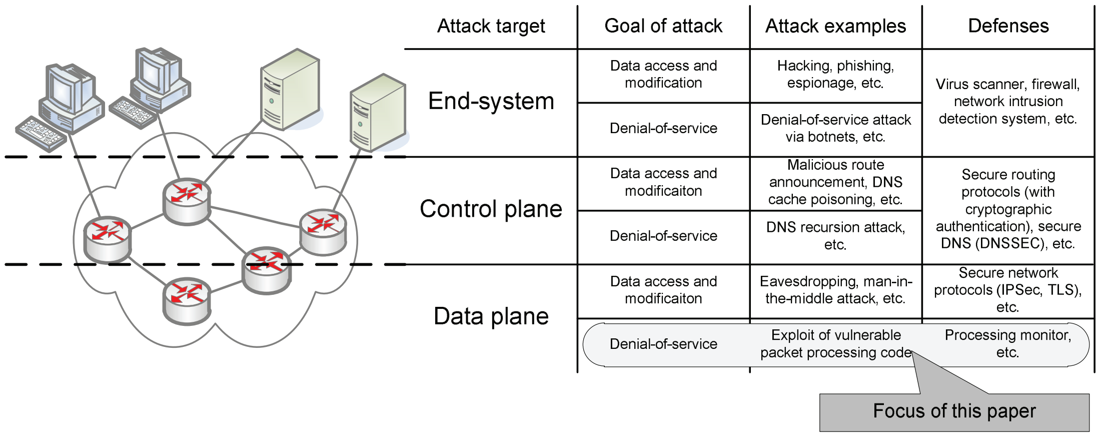
III. Vulnerabilities and Attacks in Network Infrastructure

Before discussing the details of our specific attack in Section IV, we provide a brief overview of the vulnerabilities and potential attacks in the network infrastructure.

A. 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], [14], [15], [32]. 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.



Attack target	Goal of attack	Attack examples	Defenses
End-system	Data access and modification	Hacking, phishing, espionage, etc.	Virus scanner, firewall, network intrusion detection system, etc.
	Denial-of-service	Denial-of-service attack via botnets, etc.	
Control plane	Data access and modificaiton	Malicious route announcement, DNS cache poisoning, etc.	Secure routing protocols (with cryptographic authentication), secure DNS (DNSSEC), etc.
	Denial-of-service	DNS recursion attack, etc.	
Data plane	Data access and modificaiton	Eavesdropping, man-in-the-middle attack, etc.	Secure network protocols (IPSec, TLS), etc.
	Denial-of-service	Exploit of vulnerable packet processing code	Processing monitor, etc.

Fig. 1. Examples of network attacks and defenses.

B. 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.

1) Security Requirements: The basic security requirement in our model is that the operation of the router does not change under attack. 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). It is important to note that in most attack scenarios a modification of behavior is necessary even when modification of or access to data is the ultimate goal of the attack. This leads to two main security requirements that ensure that the router continues to perform correct protocol processing: (1) Benign packets should be processed according to protocol specifications without interference from possible attacks; (2) Malicious traffic should be identified and be discarded.

We show in Section IV how an attacker can violate security requirement (1) and in Section VI how processing monitoring can enforce requirement (2) and thus circumvent the problems caused by attack traffic.

2) Attacker Capabilities: The capabilities of an attacker that define the potential attack space include the following: (1) An attacker can send arbitrary data and control packets; (2) An attacker can modify instruction and data memory through exploits; (3) An attacker cannot modify the source code or binary of the protocol implementation before it is installed on the router; (4) An attacker cannot physically access the router. These capabilities reflect what most practical attackers can do: they can try to hack a router remotely (i.e., (1) and (2)), but the basic functionality of the router is benign (i.e., (3) and (4)).

Based on this security model, we present a concrete attack.

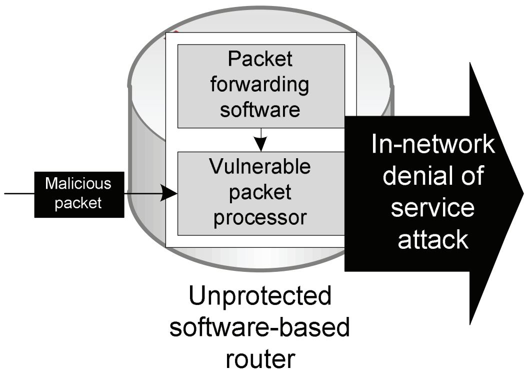

Packet forwarding software
Malicious packet
Vulnerable packet processor
In-network denial of service attack
Unprotected software-based router

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.

IV. Network Infrastructure Attack

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.

A. 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 [33].

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.

B. 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 [34] 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.

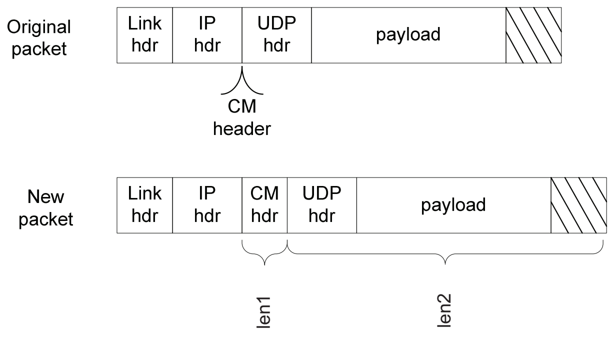


Fig. 3. Protocol Header Insertion.

```
#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.

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 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.

C. 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 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

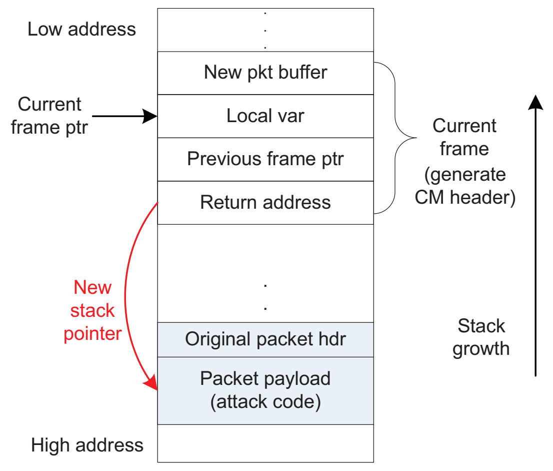


Fig. 5. Stack Smashing.

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 V, 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.

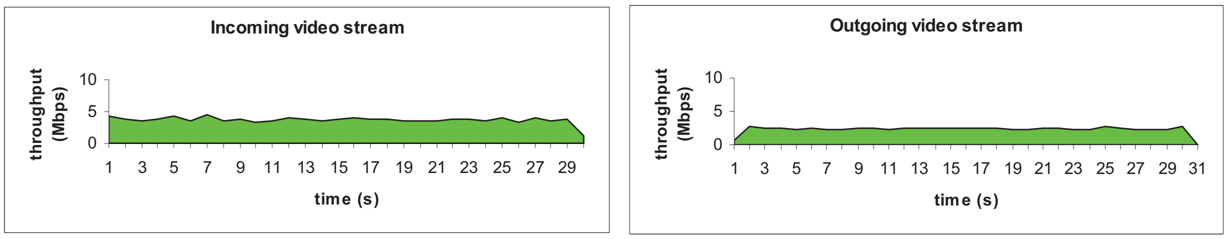
With this example, we demonstrate that vulnerabilities in software-based routers are not only hypothetical, but can occur in common protocol processing code. We also show that these vulnerabilities can be exploited to execute arbitrary attack code.

V. Results

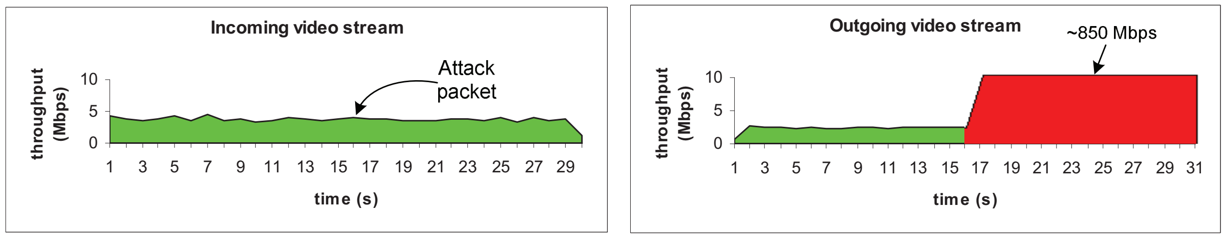
To demonstrate the feasibility and effects of the attack described in Section IV, we show a prototype implementation in a real network. We have implemented the attack on the Click modular router [2] and on a custom packet processor [3] based on the NetFPGA platform [4]. The custom packet processor uses a Plasma soft core [35], 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 IV and exhibits the integer overflow vulnerability.

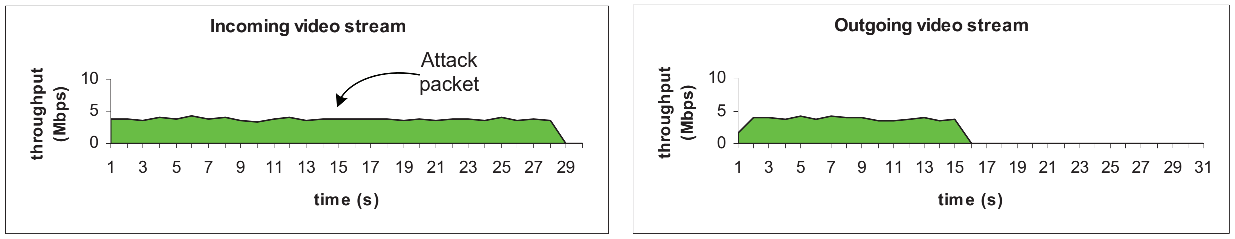
We measured the incoming and outgoing traffic on the first router for different scenarios. Figure 7 shows the results for



(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.

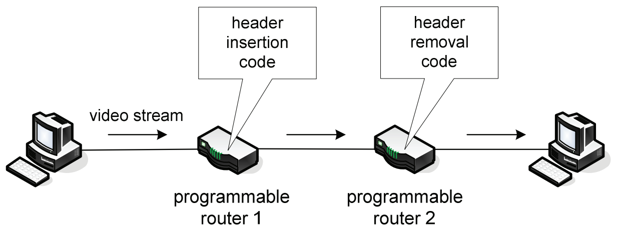


Fig. 6. Experimental Setup.

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.

VI. Defense Mechanisms

To defend against this type of attack on the packet processing systems of routers, a variety of security mechanisms could be used. These can range from using No eXecute (NX) bits to mark non-instruction memory to other security techniques used in embedded systems [26]. In our prior work,

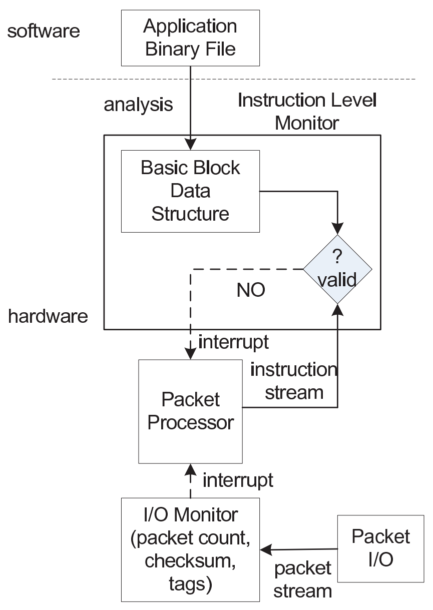


Fig. 8. Monitoring System Overview (from [5]).

we proposed a secure packet processor design [5]. We briefly describe the operation here and demonstrate that it can defend against the attack we describe.

Defense mechanisms for a packet processor have the following requirements:

- The system should be able to correctly identify date plane attacks.
- When an attack occurs, malicious traffic should be detected and eliminated quickly to reduce its potential impact.
- The detection mechanism should be lightweight, since embedded processors have limited resources.

Our secure packet processor design accomplishes these goals. It uses a fine-grained hardware monitor to track the instruction-level operations of the packet processor. These operations are compared to a reference model of operation that has been obtained through offline analysis of the processor's binary file. Under normal conditions, the operations reported by the processor match the offline model. This process is illustrated in Figure 8. If an attack occurs, the behavior of the processor changes in an unexpected way (e.g., executes malicious code instead of executing the functions that it was programmed for) and the executed operations no longer match the reference model. The secure packet processor can detect this condition, drop the offending packet, and initiate a recovery process that resets the processor core and allows the normal operation to resume.

We have implemented this type of monitor on the same custom processor used for the experiments shown in Figure V. 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 9 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.

VII. 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. To our knowledge, this work represents the first time a practical attack on the data plane of the actual network infrastructure has been shown and thus provides an important step toward understanding and correcting vulnerabilities in the network infrastructure.

Acknowledgements

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

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.

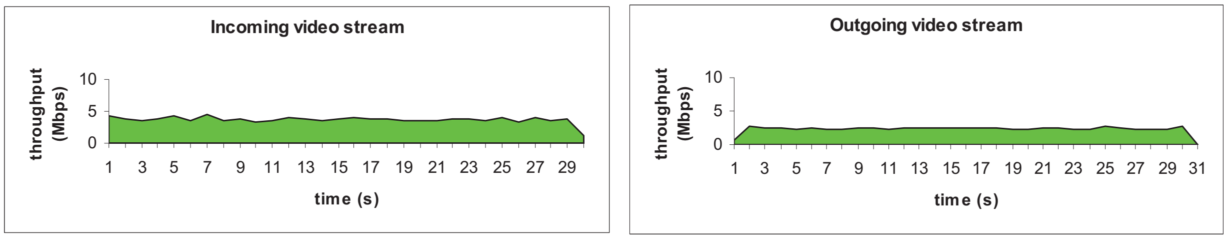
[2] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, Aug. 2000.

[3] 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.

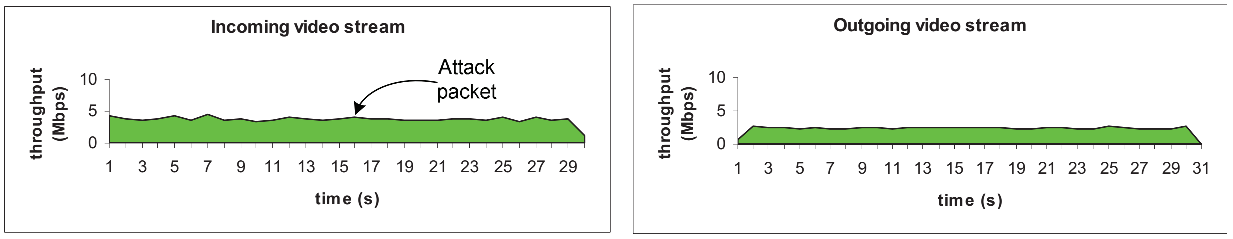
[4] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo, "NetFPGA–an open platform for gigabit-rate network switching and routing," in *MSE '07: Proceedings of the 2007 IEEE International Conference on Microelectronic Systems Education*, San Diego, CA, Jun.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

[5] 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.

[6] 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.



(a) Benign network traffic



(b) Benign traffic and single attack packet

Fig. 9. Traffic Rates at Input Port and Output Port of Router with Processing Monitor.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

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

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

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

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

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

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

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

[19] 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.

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

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

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

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

[24] 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.

[25] 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.

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

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

[34] 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.

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