# System-Level Security for Network Processors with Hardware Monitors

Kekai Hu, Tilman Wolf, Thiago Teixeira and Russell Tessier
Department of Electrical and Computer Engineering
University of Massachusetts, Amherst, MA, USA
{khu,wolf,tteixeira,tessier}@ecs.umass.edu

## ABSTRACT

New attacks are emerging that target the Internet infrastructure. Modern routers use programmable network processors that may be exploited by merely sending suitably crafted data packets into a network. Hardware monitors that are co-located with processor cores can detect attacks that change processor behavior with high probability. In this paper, we present a solution to the problem of secure, dynamic installation of hardware monitoring graphs on these devices. We also address the problem of how to overcome the homogeneity of a network with many identical devices, where a successful attack, albeit possible only with small probability, may have devastating effects.

## 1. INTRODUCTION

The Internet is a critical communication infrastructure for many aspects of our society. As such, it is a vehicle for and target of many malicious attacks. Network security has long focused on scenarios where the network is used to provide connectivity between an attacker and the attacked end-system. Results of such attacks compromise the attacked end-system [4], which can be used to steal data, execute malicious code (e.g., botnets [6]), etc. There are numerous existing mechanisms aiming to defend against such attacks at the edge of the network (e.g., by filtering malicious traffic with firewalls [10] or content-inspection systems [12]) or on the attacked devices (e.g., virus scanner software).

Recently, a new class of attacks has emerged targeting the network infrastructure itself. This type of attack aims to disrupt or modify the operation of routers to achieve denial of service attacks or to use routers to actively launch denial of service attacks. Attacks that target the control interface of routers [5] can be prevented using standard security mechanisms for end-systems. However, there are attacks on network processors, the general-purpose processor cores used for packet forwarding in routers, that can be launched through the data plane by simply sending malformed data packets [3]. These types of attacks are particularly problematic since they target critical infrastructure and cannot be easily defended against with conventional mechanisms.

Network processors are used in routers of all major vendors and are implemented using multiprocessor system-on-a-chip (MPSoC) devices. The processor cores on these MPSoCs are simple embedded cores that are typically incapable of running operating systems or other software defense mechanisms. Protection mechanisms for these types of processors have been proposed in the form of hardware monitors that track the operation of each core, detect deviations from programmed behavior due to attacks, and perform reset and recovery. These hardware monitors have been studied in the context of network processors (e.g., [7]), as well as in the context of embedded systems in general (e.g., [1, 11]).

While the design and operation of single processor cores with hardware monitors running one or a small number of preconfigured applications is well-understood, there is also a need to look at the *system-level perspective* of the problem. There are two key challenges for a practical hardware monitoring system for network processors: (1) *Dynamics:* multiple processor cores and their monitors need to be managed and reprogrammed at runtime as network traffic and network functionality change. (2) *Homogeneity:* To simplify management, practical networks use large numbers of identical router devices, which can lead to Internet-scale failures in case an attack can be developed circumventing one specific monitoring system. This paper addresses these system-level issues and presents the design of a monitoring system that can securely install binaries and monitors on network processors and parameterize these configurations such that potentially successful attacks cannot propagate.

The specific contributions of our paper are:

- Design of a Secure Dynamic Multicore Hardware Monitoring System (SDMMon), which enables secure installation of binaries and monitors on network processor systems based on cryptographic principles and suitable key management.
- Design of a novel, high-performance, parameterizable hash function for use in hardware monitors enabling the deployment of diverse monitoring systems that are not susceptible to the same potential attack.
- Evaluation of a prototype system implementation showing SDMMon functionality and performance.

We first discuss the operation of hardware monitors and our security model. Then we present the design of our system-level architecture followed by results from our prototype and related work.


*DAC'14,* June 01–05 2014, San Francisco, CA, USA.

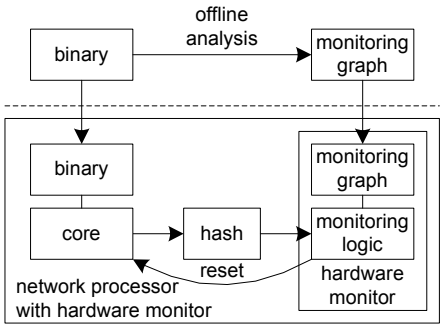


**Figure 1: Hardware monitor operation. Application binaries are analyzed offline to obtain the monitoring binary. During runtime, a hashed value of the processor operation is compared to the information in the monitoring graph. An attack causes a deviation from the monitoring graph, triggering a reset of the network processor core.**

# 2. HARDWARE MONITORS AND SECURITY MODEL

To provide the necessary context for the system architecture presented in Section 3, we briefly discuss the operation of hardware monitors and the security model for our work.

## 2.1 Secure Processing with Hardware Monitors

The hardware monitors used in our work are based on [8] and illustrated in Figure 1. The monitors track the operation of the processor core for every instruction and compare the actual behavior to expected behavior. To describe expected, "normal" packet processing, the processing binary is analyzed to extract a "monitoring graph." The graph contains all possible control flow operations between basic blocks and a short (e.g., 4-bit) hash of each binary instruction. The use of a hashed version of the binary instruction (rather than the full binary instruction) is necessary to reduce the size of the monitoring graph to a fraction of the processing binary.

During processing, the processor core reports the hash value of its current operation. The monitor compares that hash value to the one stored in the hardware monitor. If the hash matches, the comparison continues for the next instruction. In case of control flow changes, the monitor considers both next operations as valid (since the monitor does not have a data path to compute which choice is valid). In case the processor is attacked and starts executing code that is different from the original binary (e.g., after a stack smashing attack), the hash values reported by the processor core differ from the hash values in the monitor and the attack can be detected. In an Internet Protocol (IP) based network, recovery from an attack can be performed easily by dropping the attack packet, resetting the processing stack, and continuing with processing the next packet.

The security of the hardware monitoring system is based on the observations that any meaningful attack necessarily needs to change the operation of the processor core (otherwise the core continues processing unchanged). Changes in processor core operation generate different hash values that are reported by the processor core. While an attacker may create an attack where the hash values of particular instructions match the hashes of the valid instructions they replace, the probability of a matching sequence decreases geometrically with the length of the sequence. (For example, when using a 4-bit hash, there is a 1 in 16 chance that one instruction matches the monitor, a 1 in 256 chance for a match for two instructions, etc.) Crafting a set of instructions that represent a meaningful attack and matches a predetermined sequence of hash values is particularly difficult if the attacker does not know what hash function is used.

One key requirement for the security of the hardware monitoring system is that an attacker cannot modify the hardware monitoring graph. If the attacker could modify the monitoring graphs, an attack could be hidden by substituting the correct monitoring graph with one that would accept the attack as valid code. Prior and related work have not suitably addressed the question of how the monitoring graph is installed in a hardware monitoring system. For embedded systems that execute a single, unchanging binary (as was assumed in prior and related work), a one-time installation of the monitoring graph through a dedicated interface can be assumed. However, for network processors systems that need to dynamically download new processing code, there is no existing suitable solution.

## 2.2 Security Model

In the following, we focus on how to achieve the secure installation of valid hardware monitoring graphs. We do not consider security issues relating to the hardware monitor itself since these issues have been addressed in prior work [3, 8]. Instead, the focus is on the security issues relating to dynamically installing monitoring graphs onto network processor systems while considering that attackers may tamper with this process to be able to launch attacks that accept malicious code as valid.

To make the security model realistic in the context of practical network operation, we consider three entities that are part of the system environment:

- Network processor manufacturer: The manufacturer produces network processors and router systems and sells them to the network operator. In some cases, the network processor is manufactured by a different party and then integrated on the router system by the router manufacturer. For simplicity, we assume that the same entity produces the router and the network processor.
- Network operator: The operator purchases the router system with network processors from the manufacturer and programs its operation for use in the network.
- Network processor device: The network processor device is programmed by the network operator. That is, the network processor needs to obtain processing binaries and monitoring graphs from the network operator.

### *2.2.1 Security Requirements*

The specific system-level security requirements for a network processor system with hardware monitors are:

SR1 Only valid binaries and matching hardware monitor graphs should be installed on the network processor. Validity implies that the binary and monitor have been authenticated as being sourced from the network processor's network operator.

SR2 Hardware monitoring mechanisms should be sufficiently diverse – despite the operation of identical binaries –

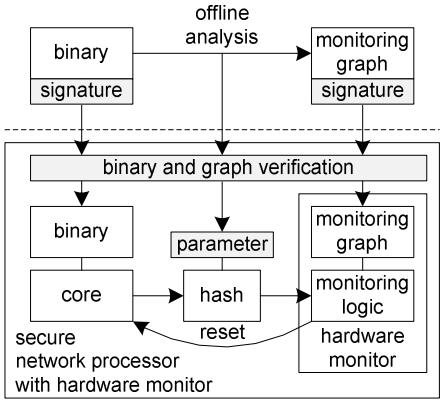


**Figure 2: Hardware monitor with system-level security. Application binaries and monitoring graphs are signed to ensure authenticity and integrity. In addition, the hash function for each network processor core is parameterized differently to achieve heterogeneity.**

to avoid catastrophic failures in a highly homogeneous network environment in case of a successful attack.

SR3 Binaries, monitoring graphs, and hash parameters should be confidential to prevent an attacker from obtaining a hash parameter and from "stealing" the intellectual property of a binary.

SR4 Binaries and monitor graphs should only be identified as valid on one specific network processor system. This security requirement helps in preventing an attacker from injecting an binary from a different device.

### 2.2.2 *Attacker Capabilities*

We assume that attackers can do the following:

AC1 An attacker can observe any traffic and inject any type of traffic. To limit the scope of this work, we do not consider a case where an attacker can block all traffic on a link. This problem can be addressed through other techniques (e.g., multipath transmissions).

AC2 An attacker can generate a monitoring graph that matches a binary that is vulnerable to a chosen attack.

To enable us to find a solution to this security problem, we also need to constrain the abilities of the attacker. Specifically, we assume the following limitations:

AC3 An attacker cannot obtain cryptographic keys stored by any of the three entities.

AC4 An attacker cannot break standard symmetric and asymmetric cryptographic algorithms.

These limitations also imply that we do not consider physical or side-channel attacks. While such attacks may exist, there is ongoing research to develop suitable protection mechanisms.

# 3. SYSTEM-LEVEL ARCHITECTURE

The system architecture for SDMMon is shown in Figure 2. The main difference to conventional hardware monitoring approaches, such as shown in Figure 1, is the use of

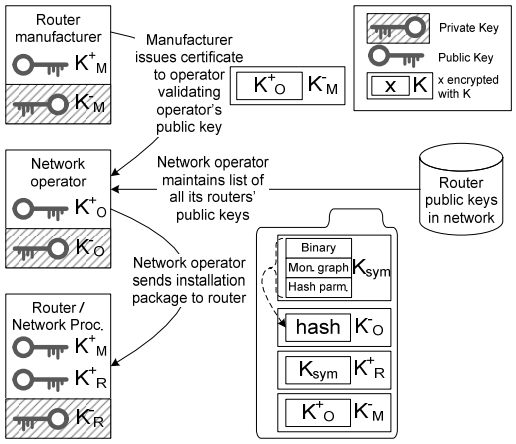


**Figure 3: Security operations in SDMMon.**

signed binaries, monitoring graphs, and hash function parameters. This approach protects the system from attacks as we discuss below.

A critical aspect for security and operational functionality is the set up of keys and cryptographic operations. Figure 3 shows the three entities that we consider for our work, the router manufacturer, the network operator, and the network processor device. In the following, we explain the interactions between these entities and how they achieve the required security properties.

## 3.1 Operation and Key Management

The following operations describe the interactions between the entities:

- At manufacturing time: During initial setup of the network processor, the manufacturer configures the device with a public key/private key pair (denoted as $K_R^+$ and $K_R^-$). The manufacturer also installs the manufacturer's public key ($K_M^+$) into the device so that a root of trust can be established. The keys can be stored in hardware logic or a trusted platform module.

- At installation time: When the network processor is installed in a network operator's network, the manufacturer provides a certificate that contains (at least) the network operator's public key signed with the manufacturer's private key. Using this certificate, the network processor can establish a chain of trust to the network operator. This certificate may be sent to the network processor once at boot time or with every reprogamming step.

- At programming time: To program the network processor, the network operator generates a monitoring graph obtained from the processing binary. The monitoring graph is then parameterized with a randomly chosen 32-bit hash parameter. The binary, the monitoring graph, and the hash parameter are then signed with the network operator's private key. In addition, the binary, monitoring graph, and hash parameter are encrypted with a random symmetric key ($K_{sym}$). The symmetric key is encrypted with the router's public key to ensure only the router can decrypt this information. The encrypted binary, monitoring graph, and

hash parameter, the signature, the encrypted key, and the certificate are then transmitted over the network to the network processor. The network processor decrypts the data with the provided symmetric key (after applying the router's private key) and verifies the authenticity and integrity of the data with the public key of the network operator.

- At runtime: When the network processor performs packet processing operations, the processor reports its 32-bit operation to the parameterizable hash function. The 4-bit hashed operation is then reported to the hardware monitor that compares it to the monitoring graph (as discussed in Section 2).

We do not consider the question of when each binary needs to be installed on the network processor. There has been extensive work on workload management on multicore network processors to guide that decision [7, 13]. We focus on the problem of installing the binary and monitor securely once the decision that this binary is required has been made.

## 3.2 Parameterizable Hashing

The hash function used in our system takes the instruction word executed by the processor core and maps it to a smaller (e.g., 4-bit) hash value. This value is then validated by the hardware monitor against the information in the monitoring graph. The monitoring graph has a hashed value of each instruction in the binary and thus can detect when the executed instruction does not match. Monitor graphs with small hash values can be represented very compactly and processed with a single memory access.

The drawback of using a hashed representation of the executed instruction word is that hash needs to be computed every processor clock cycle and that hashing is a many-to-one mapping. The latter can be exploited by a potential attacker by creating an attack that hijacks the processor with an instruction sequence that is identical to the hash values expected by the monitor. To provide security from such an attack, our SDMMon uses different hash function parameters on each router and these parameters are communicated securely between network operator and router. Thus, an attacker cannot know what hash function to expect and the only viable attack would be a brute force enumeration of different hash sequences. The use of different hash parameters on different routers also ensures that a potentially successful brute force attack on one system cannot be exploited on other systems.

The design challenge for the hash function used in our hardware monitoring system is to provide good hash characteristics while allowing a high-performance implementation. Cryptographic hash functions would be a great choice since they can be parameterized with a cryptographic key and achieve strong collision resistance, but they require too much processing complexity. Instead, we aim for a hash function with weak collision resistance that can be implemented efficiently in hardware.

We base our hash function design on a Merkle tree [9], as shown in Figure 4. The tree structure can be efficiently implemented in hardware and requires only a logarithmic number of dependent operations based on the instruction and parameter length. Each tree node computes an 8-to-4 bit compression function as part of the overall hash calculation. Leaf nodes take 4 bits from the hash function parameter and 4 bits from the processor instruction.

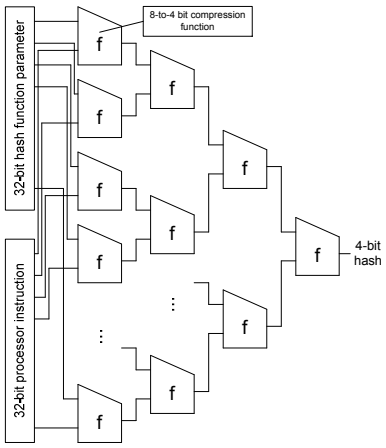


**Figure 4: Parameterizable hash function based on Merkle tree.**

Without knowledge of the parameter used in the calculation, an attacker cannot guess the mapping of a potential 32-bit attack instruction to its 4-bit hash. As discussed above, brute force probing is possible, but difficult to implement for longer attacks.

## 3.3 Security Properties

Based on our system-level design of secure transmission of binaries and monitoring graphs, as well as the use of a hash function with different parameters for different router systems, we can now illustrate how our security requirements can be achieved.

1. Security requirement SR1 (only valid binaries installed) is achieved because the packages of binaries and monitoring graphs are signed by the network operator. Because the attacker cannot obtain the network operator's private key (AC3 and AC4) and because only valid network operators receive certificates from the manufacturer, the packages have to be authentic.

2. Security requirement SR2 (hardware monitor diversity) is achieved because each monitor instance uses a different, randomly chosen, hash parameter. The attacker can try to generate an attack that matches (AC2), but would need to do that using a very inefficient brute-force approach.

3. Security requirement SR3 (confidentiality) is achieved because the components of the package are encrypted with a symmetric key that is only available to the network operator and the router (because of AC3 and AC4). The attacker can observe the package (AC1), but cannot interpret the content.

4. Security requirement SR4 (binaries and monitor graphs specific to single system) is achieved because the symmetric encryption key is encrypted with the router's public key. Therefore, only the router for which a package is intended can correctly decrypt the information in it (again because of AC3 and AC4).

Because we can ensure that our security requirements are

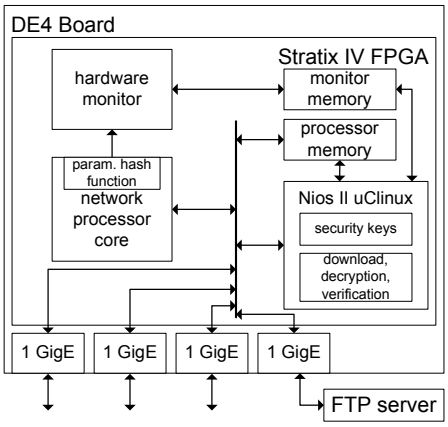


**Figure 5: Prototype network processor system configuration.**

**Table 1: Resource use on DE4 FPGA**

| | Available on FPGA | Nios II contr. proc. | NP core with hw monitor |
|---|---|---|---|
| LUTs | 182,400 | 13,477 | 41,735 |
| FFs | 182,400 | 16,899 | 40,590 |
| Memory bits | 14,625,792 | 497,976 | 2,883,088 |

maintained, we claim to achieve system-level security for network processors with hardware monitors.

# 4. PROTOTYPE IMPLEMENTATION

To illustrate how a secure system for network processors with hardware monitors can be implemented in practice, we present results from a prototype implementation.

## 4.1 System Setup

We have implemented a prototype SDMMon in an Altera Stratix IV FPGA on an Altera DE4 board. A reconfigurable network processor was implemented with a PLASMA processor and a reconfigurable hardware monitor was connected to this NP. For the security and dynamic control purpose, a Nios II soft processor was implemented as the control processor. The board contains four 1Gbps Ethernet ports, through which the control processor can reach the network operator's server. The system can download, decrypt, and verify the binaries and monitor graphs, load the binaries and graphs to the shared memory, and reconfigure the network processor and hardware monitor. We have installed a uClinux operating system in the Nios II core to provide essential service support such as TCP/IP, FTP, SSH, and OpenSSL.

The relative size of the control processor compared to a network processor core with hardware monitor is shown in Table 1. The control processor, which performs all the security operations, is only about one third the size of a network processor core with hardware monitor. In addition to the resources shown in the table, the control processor also requires 2895kB of memory for the operating system image.

## 4.2 Binary and Monitoring Graph Installation

We have implemented the decryption and verification steps on the embedded control processor of the network processor system. All cryptographic operations use the commercial-grade OpenSSL toolkit (version 1.01e).

**Table 2: Processing of security functions on Nios II**

| Step | Time (s) |
|---|---|
| Download data from FTP server | 1.90 |
| Check manufacturer certificate of network operator's public key $K_O^+$ | 3.33 |
| Decrypt AES key $K_{sym}$ using router's private key $K_R^-$ | 8.74 |
| Decrypt package with AES key $K_{sym}$ | 7.73 |
| Verify packet signature with network operator's public key $K_O^+$ | 3.92 |
| Total | 25.62 |
| Total (no networking or certificate check) | 20.39 |

**Table 3: Implementation cost of hash functions**

| | Bitcount hash | Merkle tree hash |
|---|---|---|
| LUTs | 103 | 95 |
| FFs | 61 | 61 |
| Memory bits | 0 | 32 |

We generate public/private key pairs for all three entities – network processor manufacturer, network operator, and network processor device – using the RSA algorithm with key length of 2048 bits. We sign the operator's public key with the manufacturer's private key to create a certificate to establish a chain of trust.

The package of binary (for IPv4+CM), monitoring graph, and hash parameter is signed with the operator's private key. Since the package size exceeds the RSA algorithm's capacity to encrypt it, we encrypted the package using a randomly chosen AES symmetric key. That symmetric key is then encrypted with the router's public key to allow secure exchange.

At the network processor device, we verify the certificate to make sure the operator is indeed the operator and not an attacker. We also decrypt the AES key using the router's private key, making it able to access the package upon decryption of the AES algorithm. We can then verify the signature using the operator's public key. Finally, we unpack the package and obtain the binary, monitoring graph, and hash parameter. These files are then installed in the memory network processor device.

The running times of the various steps taken on the control processor are shown in Table 2. The total time is about 25 seconds, which is acceptable since new processing applications for network processors are created at slower time scales. (Note that switching between applications already installed on the network processor can be done quickly to accommodate dynamic changes in workload by keeping multiple binaries and graphs in memory.) When skipping the certificate check (which has to be done only once) and ignoring network delay (which can be decreased based on server location), then the verification time is around 20 seconds.

## 4.3 Hash Function Evaluation

We have also implemented the parameterizable hash function on the prototype system. As compression function $f$, we use the 4-bit arithmetic sum of both 4-bit inputs. The resource consumption of the implementation of the hash function is shown in Table 3. For comparison, the resources for a

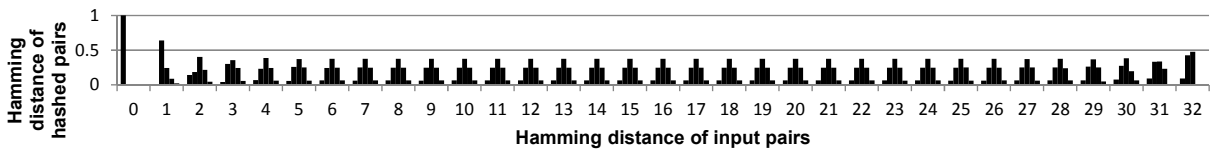


**Figure 6: Distribution of hash values using our Merkle-tree-based hashing.**

typical conventional hash function (counting set bits in the word to hash) are also shown. It can be seen that the resource requirements are comparable. Our Merkle tree hash requires less logic, but requires memory to store the parameter, whereas the bitcount hash does not require memory. Both hash functions are fast enough to compute the hash within the available cycle time on our system.

To show that the parameterizable hash function based on Merkle trees is effective for our goal, i.e., providing diversity across systems (security requirement SR2), we evaluate the distribution of generated hash values. We created $10^{11}$ 32-bit value pairs (i.e., two different processor instructions) and compared their 4-bit hashed value pair. As a comparison metric, we use the Hamming distance between each pair, which is an indication on the number of bits that is different between the values in the pair. Figure 6 shows the distribution of Hamming distances for hashed values for each possible Hamming distance of the original pair. This figure shows that changes in the input to the hash function (or its key due to symmetry of inputs in the Merkle tree) lead to changes in the output that have a Gaussian distribution of the Hamming distance and thus are indistinguishable from random changes (except for a Hamming distance of 1, where the distribution is slightly different). Therefore, we can argue that an attacker cannot guess or strategize what 32-bit instruction leads to a particular hash value other than by exhaustively calculating the hash.

## 5. RELATED WORK

Hardware monitoring for embedded systems has been developed at different levels of granularity (e.g., basic blocks [1, 11], processor instructions [8]). All hardware monitoring approaches identify attacks by comparing processor behavior to expected behavior based on offline analysis. Under dynamic workloads [13], dynamic installation of binaries and data describing expected behavior is necessary.

Dynamic and secure installation of data has been explored in the context of FPGAs [2], which is similar to our problem, but with different security goals (protection of IP of more concern than installation of a compromised bitfile). Also, this work assumes local area network connections to derive some security properties. In our work, we provide security guarantees through cryptographic approaches while allowing devices to be distributed anywhere in the Internet, which is more realistic for router systems.

## 6. SUMMARY AND CONCLUSIONS

We have presented a system-level security design that ensures that hardware monitors on network processors can be programmed dynamically, while ensuring that attackers cannot tamper with the monitoring system. Our design is based on the use of cryptographic principles to securely transfer all the necessary data to the network processor. We have also designed a parameterizable hash function that allows monitoring graphs to be customized to each individual router system in order to protect from a cascading attack. We have demonstrated to operation of our system in hardware on an FPGA-based platform. We believe that this work provides an important contribution toward moving from device-level security to system-level security in embedded hardware monitoring.

## 7. REFERENCES


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

[2] Bomel, P., Crenne, J., Ye, L., Diguet, J.-P., and Gogniat, G. Ultra-fast downloading of partial bitstreams through Ethernet. In *Proc. of the 22nd International Conference on Architecture of Computing Systems (ARCS)* (Delft, The Netherlands, Mar. 2009), pp. 72–83.

[3] Chasaki, D., and Wolf, T. Attacks and defenses in the data plane of networks. *IEEE Transactions on Dependable and Secure Computing 9*, 6 (Nov. 2012), 798–810.

[4] Clough, J. *Principles of Cybercrime*. Cambridge University Press, June 2010.

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

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

[7] Kumarapillai Chandrikakutty, H., Unnikrishnan, D., Tessier, R., and Wolf, T. High-performance hardware monitors to protect network processors from data plane attacks. In *Proc. of 50th Design Automation Conference (DAC)* (Austin, TX, June 2013).

[8] Mao, S., and Wolf, T. Hardware support for secure processing in embedded systems. *IEEE Transactions on Computers 59*, 6 (June 2010), 847–854.

[9] Merkle, R. C. *Secrecy, Authentication, and Public Key Systems*. PhD thesis, Stanford University, Stanford, CA, June 1979.

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

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

[12] Roesch, M. Snort - lightweight intrusion detection for networks. In *Proc. of the 13th USENIX Conference on System Administration (LISA)* (Seattle, WA, Nov. 1999), pp. 229–238.

[13] Wu, Q., and Wolf, T. Runtime task allocation in multi-core packet processing systems. *IEEE Transactions on Parallel and Distributed Systems 23*, 10 (oct 2012), 1934–1943.