High-Performance Hardware Monitors to Protect Network Processors from Data Plane Attacks

Kekai Hu, Harikrishnan Chandrikakutty, Deepak Unnikrishnan, Tilman Wolf, and Russell Tessier

Department of Electrical and Computer Engineering
University of Massachusetts Amherst
Outline

- The problem: software attacks on network routers
  - Routers now include programmable processors
- Our solution: include a monitor for the processor
- Software to generate the monitor information
- Implementation on DE4 NetFPGA board
- Experimental results
Computer Networks

- Networks provide **connectivity** between end-systems
- Success of the Internet: **hourglass architecture**
- Success is also a problem: **diverse apps, diverse systems**
- **Changing requirements** for the network layer

Layered protocol stack

Example protocols

...
Network Architecture

- **Extensions** to current Internet
- **Customization** of data plane
- Requires router systems with **ability to adapt**
Programmable Router

- Network processor
  - General-purpose processing capability in data path
  - Packet processing in software

- High-performance processing hardware
  - Scalability through high levels of parallelism
  - Example: 40-core network processors

- Key challenge:
  - Security due to software processing
Exploit of Vulnerable Network Processor

- Vulnerability in network processor can be exploited
- “In-network” denial of service attack (new type of attack)
- Key questions
  - Can such vulnerabilities occur in packet processing code? (Yes, we show one example.)
  - Can vulnerabilities be exploited? (Yes, for von-Neumann and for Harvard architectures)
Header Insertion Exploit

- Stack smashing during header insertion
  - Control flow can be changed to attack code

- Attack code: infinite transmission loop
  - Devastating denial-of-service attack

- Prototype implementation
  - Custom network processor on NetFPGA

- Harvard arch.: possible, but less exciting

![Diagram of stack smashing and packet payload](image)

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

- Congestion management protocol
  - Inserts a custom header

- Integer overflow vulnerability
  - $len1 = 12$
  - $len2 = 65334$
  - $sum = 10$

```c
int mybuf[60];
unsigned short sum;
Pack (in, out);

sum = len1 + len2;
if (sum > MAX_PKT_SIZE) {
    return -1;
} else {
    memset(mybuf, buf1, len1);
    memset((mybuf+len1), buf1, len2);
    return 0;
}
```
Defense Mechanism: Hardware Monitor

- Hardware monitor co-located with each processor core
  - **Core reports hash** of each executed instruction
- **Monitoring graph** represents correct behavior
  - Obtained from offline analysis of binary
  - Deviations trigger reset
- Change of software easy
  - Just need matching monitoring graph
Offline Analysis of Processing Binary

- Executed instruction reported by core as 4-bit hash
  - Hash combines address, opcode, registers
  - Hash allows for compact representation of information

- Monitoring graph
  - Each instruction represented as a state
  - Edges correspond to execution of instruction
  - Control-flow operations lead to multiple possible next states
NFA-to-DFA conversion

- Problem: non-deterministic finite automaton (NFA)
  - State may have two next states with same edge value due to hash
  - Implementation would need to keep track of multiple states

- Solution: NFA-to-DFA conversion (powerset construction)
  - Well-known algorithm [Hopcroft and Ullman, 1976]
    - Deterministic finite automaton (DFA) requires only one state
Implementation of DFA Monitor

- Each state may have up to 16 next states
  - Most states only have one or two next states

- Requirements
  - Compact representation
  - Fast processing of each hash value (single memory access)

- Idea: grouping of next states in contiguous memory
  - Trick: determine offset into memory based on order of hash value
Implementation of DFA Monitor

- DFA monitor system
  - Memory keeps all valid hash values in one **bit vector**
  - Next state based on **offset** of group and position of hash value

![Diagram of DFA Monitor System]
Altera DE4 NetFPGA Infrastructure

- Open source port of NetFPGA 1G [1]
- Networking research platform
  - Altera DE4 board
  - 5x resources compared to NetFPGA 1G
  - PCI Express, 8GB DDR2
- Configurable designs
  - Reference router
  - Packet generator
Altera DE4 NetFPGA reference router

- Complete IPV4 router
  - Forward packets on all four 1Gbps interfaces
- Design components
  - Input queues
  - Input arbiter
  - Output port lookup
  - Output queues
- Prototype system replaces output port lookup module
Single-core system architecture

- Single-core network processor system integrated along with Altera DE4 NetFPGA reference router pipeline
Harvard memory architecture

- Separate physical memory space for instruction and data
  - General memory error techniques for code-injection attacks will not be successful
  - Attacks need to be generated with existing library functions
Experimental Setup

- Altera DE4 reference router with prototype system
- Altera DE4 packet generator
  - Generate packets
  - Capture forwarded packets
- Evaluation metrics
  - Attack detection
  - Throughput
  - Resource utilization
Offline analysis

- MIPS-GCC compiler generates binary and branch information
- Powerset construction to perform NFA-to-DFA transformation.
- Memory initialization file is loaded into memory
Evaluation

- Monitoring **speed**
  - Single memory access
  - Lookup into fixed-size register file

- Memory **size** of monitor
  - More states due to NFA-to-DFA conversion
  - More states due to multiple entries in memory for certain states
  - In practice, **overhead is below 10%**

- Very **fast and compact** hardware monitor

<table>
<thead>
<tr>
<th>Network application</th>
<th>No. of instr.</th>
<th>NFA states</th>
<th>Max. mem. access</th>
<th>DFA states</th>
<th>Mem. entries</th>
<th>Mem. overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>frag</td>
<td>573</td>
<td>573</td>
<td>3</td>
<td>592</td>
<td>627</td>
<td>9.4%</td>
</tr>
<tr>
<td>mtc</td>
<td>2427</td>
<td>2427</td>
<td>3</td>
<td>2460</td>
<td>2584</td>
<td>6.4%</td>
</tr>
<tr>
<td>red</td>
<td>802</td>
<td>802</td>
<td>2</td>
<td>808</td>
<td>857</td>
<td>6.8%</td>
</tr>
<tr>
<td>wfq</td>
<td>905</td>
<td>905</td>
<td>2</td>
<td>921</td>
<td>978</td>
<td>8.0%</td>
</tr>
</tbody>
</table>
Benchmarks

• NpBench [1]
  – Modern network applications
  – Three specific functional groups
    • Traffic management and quality of service group
    • Security and media processing group
    • Packet processing group

Prototype Implementation on FPGA

- **Small overhead** compared to processor core (4096 states):

<table>
<thead>
<tr>
<th>Resources</th>
<th>Secure monitor</th>
<th>Network proc.</th>
<th>DE4 interface</th>
<th>Available in FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUTs</td>
<td>140</td>
<td>3,792</td>
<td>37,803</td>
<td>182,400</td>
</tr>
<tr>
<td>FFs</td>
<td>26</td>
<td>2,120</td>
<td>38,444</td>
<td>182,400</td>
</tr>
<tr>
<td>Mem. bits</td>
<td>131,072</td>
<td>201,216</td>
<td>2,550,800</td>
<td>14,625,792</td>
</tr>
</tbody>
</table>

- **Correct operation**: attack packet detected and dropped
Attack with Defense in Place

- Attack packet dropped, **router continues to operate**

(a) Benign network traffic

(b) Benign traffic and single attack packet
Throughput

- Throughput performance of the network processor with security monitor
  - CM protocol and IPV4 application
Multicore Monitor

- **Dynamic workloads** pose problem for hardware monitor
  - Processing may differ between packets
  - Monitors need to match processing

- **Mapping between processors and monitors**
  - 1-to-1 mapping requires frequent reload of monitor
  - Any-to-any mapping costly to implement
  - Clusters with **n-to-m mapping** provide balance

- **Interconnect is configured dynamically** depending on workload
  - Mapping between core and monitor
System Architecture of Clustered System

- Multiple cores can access multiple monitors
  - Dynamic configuration of crossbar
- Secure loading of monitors through external interface
Cluster Design

- Simple implementation of **clustered monitor**
  - Dynamic configuration through programming of demultiplexers
Dual-Ported Monitor Implementation

- Memory of **monitor can be shared** between two monitors
  - Effective use of dual-ported memory
  - Two monitoring graphs can be used in parallel
Prototype Implementation on FPGA

- Resources cost for a multi-core system (4 cores, 6 monitors):

<table>
<thead>
<tr>
<th></th>
<th>Available in FPGA</th>
<th>DE4 interface</th>
<th>Network processors</th>
<th>SHMG monitors</th>
<th>SHMG interconn.</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUTs</td>
<td>182,400</td>
<td>33,427</td>
<td>15,025</td>
<td>816</td>
<td>96</td>
</tr>
<tr>
<td>FFs</td>
<td>182,400</td>
<td>36467</td>
<td>8,367</td>
<td>147</td>
<td>0</td>
</tr>
<tr>
<td>Bits</td>
<td>14,625,792</td>
<td>2,263,888</td>
<td>1,048,567</td>
<td>786,432</td>
<td>0</td>
</tr>
</tbody>
</table>
Conclusions

- **Current and future Internet** needs to meet new demands
- Programmable routers provide packet processing platform
  - Systems problem: *security vulnerabilities*
  - **Attacks can be launched within data plane** (i.e., not control access)
  - **Monitor-based hardware defense** mechanism is effective
- Hardware monitor design and prototype
  - Uses **compact DFA** (less than 10% more states than NFA)
  - Verification with **single memory access** per instruction
  - Defense shown for **Harvard architecture** attack
- Technique extended to multicore network processors
- **Promising defense** against attacks on network infrastructure