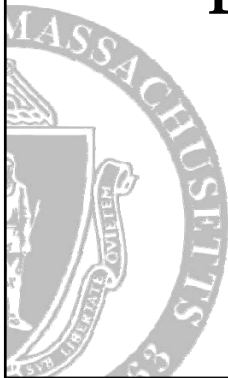


# Secure Processing in Embedded Systems



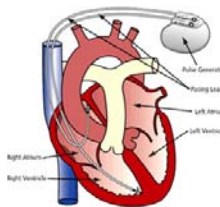
Tilman Wolf

Department of Electrical and Computer Engineering

## Embedded Systems

▪ Embedded systems are everywhere

- Smart cards
- Power meters
- Set-top boxes
- Cars
- Military applications
- Medical applications

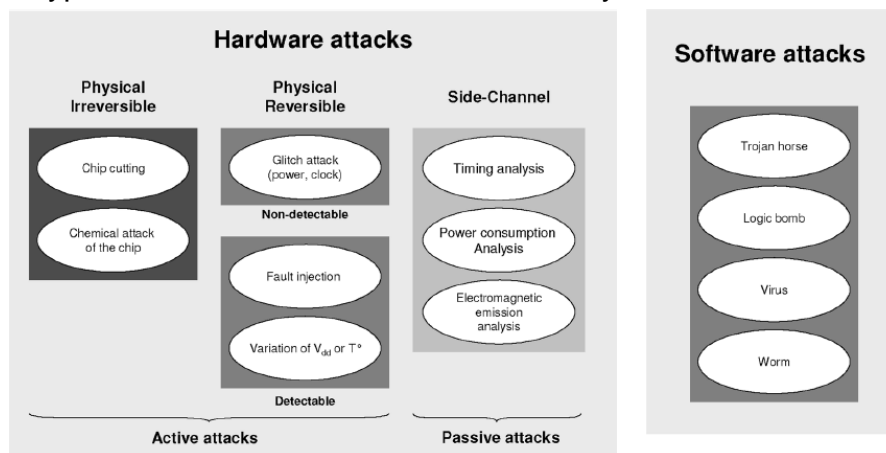


## Embedded System Characteristics

- Limited processing power
  - Cannot run typical defenses (e.g., virus scanner, intrusion detection system)
- Limited available power
  - Increased power consumption reduces system lifetime
  - Limited power resources to provide system security
- Physical exposure
  - Inherently vulnerable to attacks that exploit physical proximity of attacker
- Remoteness and unmanned operation
  - Inaccessible locations (e.g., harsh environment, remote field location)
  - Automated updates and patches provide potential targets for attacks
- Network connectivity
  - Vulnerabilities can be exploited remotely from anywhere

## Embedded System Attacks

- Typical attack scenarios for embedded systems



## Attack Goals

- Vulnerabilities open door for abuses
  - Different goals depending on attacker motivation
- Potential abuses of embedded systems
  - Energy drainage (exhaustion attack)
  - Physical intrusion (tampering)
  - Network intrusion (malware attack)
  - Information theft (privacy)
  - Introduction of forged information (authenticity)
  - Confusion/damaging of sensors of other peripherals
  - Thermal event (thermal virus or cooling system failure)
  - Reprogramming of system for other uses (stealing)

## Attack Examples

- Example 1: power meter
  - Tampering: change metering program
  - Authenticity of data: change meter reading
  - Confusion of sensor: reduce meter reading sensitivity
- Example 2: set-top box:
  - Tampering: change software to play protected content
  - Theft: extract decoder keys
  - Stealing of platform: run Linux
- Other examples
  - Networked intrusion: buffer overflow, virus, etc.



## Countermeasures

- Defenses against hardware attacks
  - Tamper-proofing
  - Fault tolerant design
  - Identification of attacks and recovery
  - Architectural defenses against side-channel leakage
- Defenses against software attacks
  - Safe languages
  - Code analysis
  - Sandboxing and damage containment
- Our approach: identification of attacks through monitoring
  - Main challenge: monitor design

## Outline

- Introduction
- **Monitoring architecture for embedded systems**
- **Processing monitor**
- Collaborative monitoring
- Summary

## Outline

- Introduction
- **Monitoring architecture for embedded systems**
- **Processing monitor**
- Collaborative monitoring
- Summary

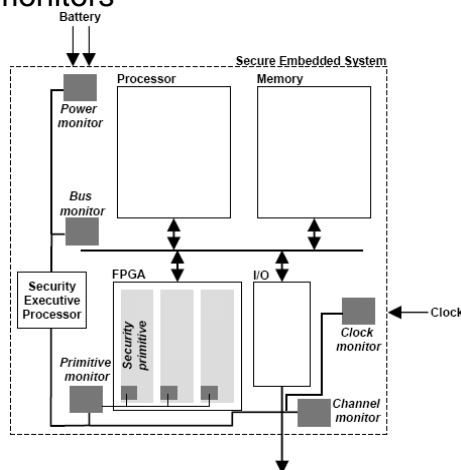
Tilman Wolf, Shufu Mao, Dhruv Kumar, Basab Datta, Wayne Burleson, and Guy Gogniat, "Collaborative monitors for embedded system security," in *Proc. of First International Workshop on Embedded Systems Security in conjunction with 6th Annual ACM International Conference on Embedded Software (EMSOFT)*, Seoul, Korea, Oct. 2006.

Guy Gogniat, Tilman Wolf, Wayne Burleson, Jean-Philippe Diguët, Lilian Bossuet, and Romain Vaslin, "Reconfigurable hardware for high-security/high-performance embedded systems: the SAFES perspective," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 2, pp. 144–155, Feb. 2008.

Shufu Mao and Tilman Wolf, "Hardware support for secure processing in embedded systems," in *Proc. of 44th Design Automation Conference (DAC)*, San Diego, CA, June 2007, pp. 483–488.

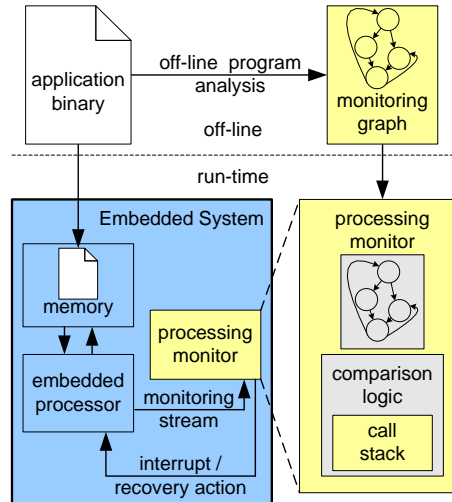
## Monitoring Architecture

- Distributed on-chip hardware monitors
  - Embedded system augmented by hardware monitors
  - Monitors communicate via network on chip
  - Control processor aggregates information
- Separate hardware
  - Less susceptible to attack
  - Less performance impact
- Our work
  - Processing monitor
  - Collaborative decision making



## Processing Monitor

- Idea: attack can be observed by deviation from expected program behavior
  - Expected behavior based on offline analysis
  - Actual behavior reported by processor
- Questions
  - What should be monitored?
  - How can expected behavior be represented?
  - What is the performance overhead?

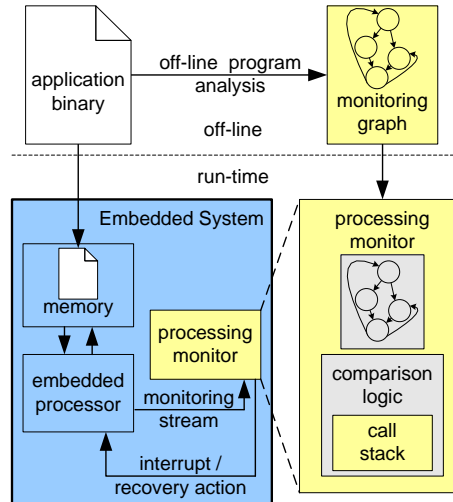


## Related Work

- Processing monitors
  - Zhang, van Doorn, Jaeger, Perez, Sailer (SIGOPS 2002)
    - Invariants on kernel data structures
  - Arora, Ravi, Raghunathan, Jha (DATE 2005) (basis of comparison)
    - Monitoring of basic blocks (hash over entire block)
  - Suh, Lee, Zhang, Devadas (ASPLOS 2004)
    - Tracking of information flow in system
  - Abadi, Budiu, Erlingsson, Ligatti (CCS 2005)
    - Control flow integrity with modified binaries
- Other monitors
  - Zhuang, Zhang, Pande (ASPLOS 2004)
    - Bus monitor to avoid data leakage
  - Chi, Salem, Bahar, Weiss (INTERACT 2003)
    - Thermal sensor for performance improvement
  - Velusamy, Huang, Lach, Stan, Skadron (ICCD 2005)
    - Thermal sensors on FPGA

# Processing Monitor

- Application analysis
  - Graph representation of possible transitions between basic blocks
- Runtime operation
  - Comparison of processor information with possible transitions on monitoring graph
  - Call stack to keep track of dynamic branches
  - Invalid state represents attack
- Recovery
  - Initiated when attack is detected



# Monitoring Information

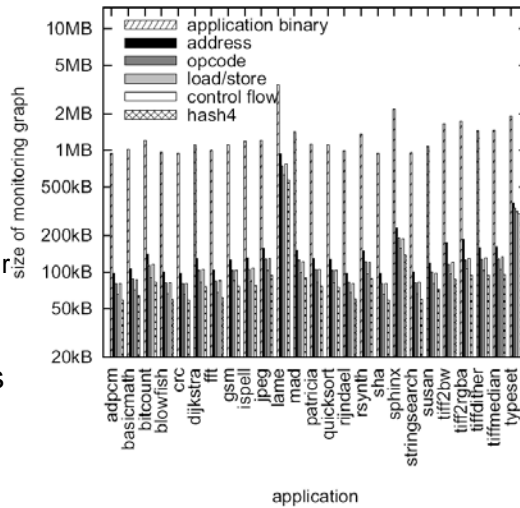
- Possible types of monitoring information:

sample object code	monitoring graph				
	address	opcode	load/store	control flow	hash4
...	...	...	...	...	...
020004d0 str r0, [sp]	020004d0	str	str r0	*	0011
020004d4 str r0, [sp, #4]	020004d4	str	str r0	*	0001
020004d8 ldr r1, [pc, #1c4]	020004d8	ldr	ldr r1	*	0001
020004dc sub r4, r11, #2080	020004dc	sub	*	*	0110
020004e0 ldr r3, [pc, #1c0]	020004e0	ldr	ldr r3	*	1001
020004e4 sub r4, r4, #8 ; 0x8	020004e4	sub	*	*	0010
020004e8 ldr r2, [r11, #-2136]	020004e8	ldr	ldr r2	*	1010
020004ec mov r0, r4	020004ec	mov	*	*	1111
020004f0 bl 02091aa0	020004f0	bl	*	bl 02091aa0	0011
020004f4 mov r0, r4	020004f4	mov	*	*	1010
020004f8 mov r1, #0 ; 0x0	020004f8	mov	*	*	0111
020004fc bl 020905dc	020004fc	bl	*	bl 020905dc	1100
...	...	...	...	...	...

- Which type of monitoring information is best?

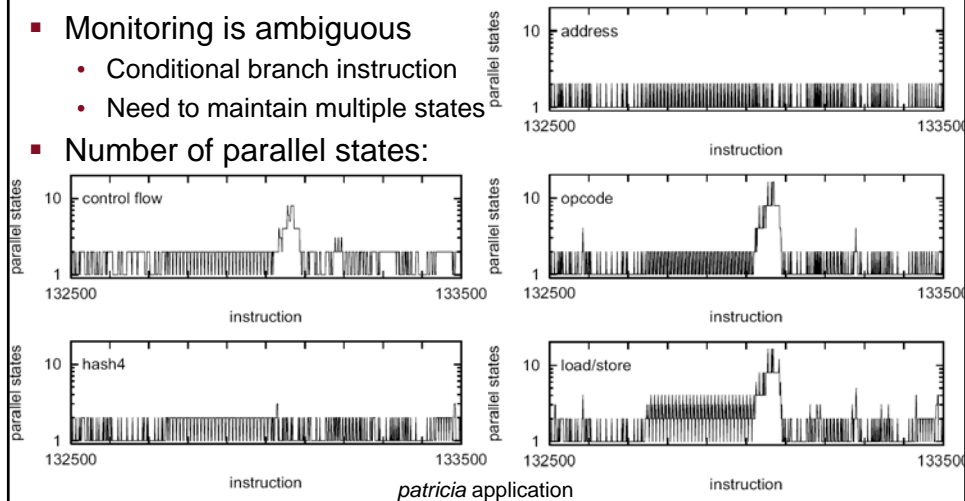
## Size of Monitoring Graph

- Evaluation:
  - MiBench on SimpleScalar
- Memory requirement of different monitors
  - Approximately 10% of application binary
  - Hash4 and load/store require least memory
  - Address requires most
- Monitoring graph requires only small fraction of application binary



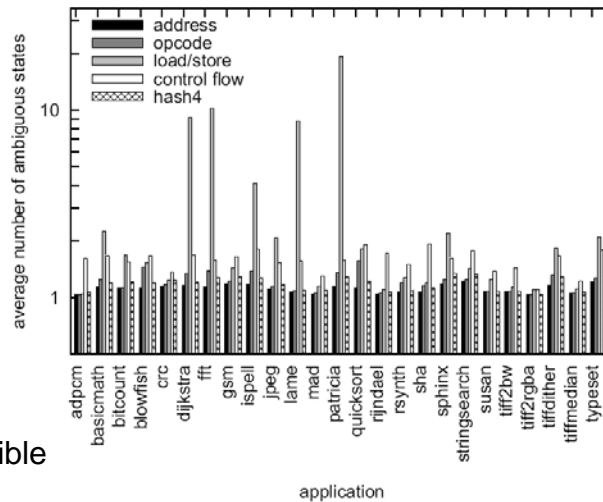
## Monitoring Ambiguity

- Monitoring is ambiguous
  - Conditional branch instruction
  - Need to maintain multiple states
- Number of parallel states:



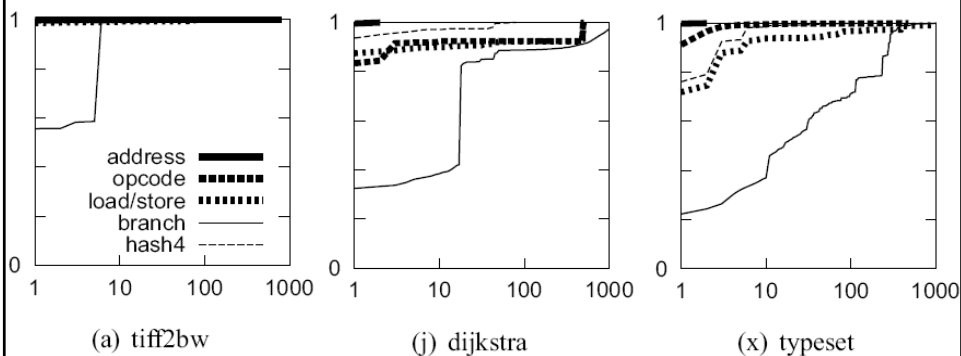
## Monitoring Ambiguity

- Average length of ambiguous execution path
  - Duration for which attack may not be detected
  - Load/store pattern shows worst performance
  - Other patterns only between 1 and 2 instructions
- Fast detection possible



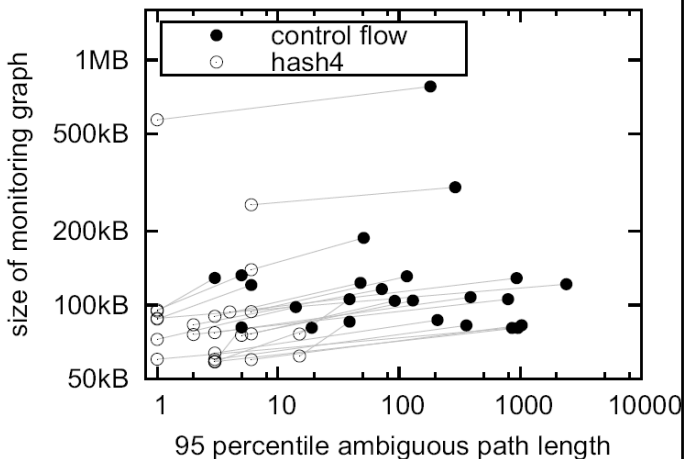
## Monitoring Ambiguity

- Cumulative distribution function of ambiguous path length
  - Application dependent
  - Hash4 beats control flow in most case



## Comparison to State of the Art

- Comparison to Aurora et al. (control flow monitor)
- Strictly better
  - Less memory
  - Faster detection



## Bit-Flip Attacks

- System performance during actual attack
  - Random bit flip in binary
  - Duration until detection:

Monitoring pattern	undetected bit flips out of 100 runs	avg. no. of instr. to detection
address	87	49.1
opcode	60	1.2
load/store	76	15.8
control flow	74	23.6
hash4	6	1
hash16	0 ( $1.5 \cdot 10^{-3}\%$ †)	1†
hash32	0 ( $2.3 \cdot 10^{-8}\%$ †)	1†
Aurora et al. [6]	0 ( $2.3 \cdot 10^{-8}\%$ †)	approx. 6†

† Results estimated and not simulated due to small probability of occurrence of event in experimental setup.

## Buffer Overflow Attacks

- Attack on application stack (buffer overflow)
  - Hash4 has fastest detection

Monitoring pattern	stack attack detection	no. of instr. to detection
address	detected	1
opcode	detected	2
load/store	detected	2
control flow	detected	10
hash4	detected	1
Arora et al. [6]	detected	10

## Processing Monitor Summary

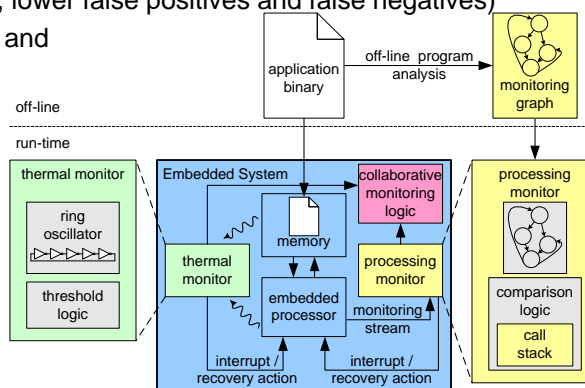
- Monitoring of processor behavior to identify attack
  - Expected behavior extracted from binary
  - Comparison to monitoring stream
- Evaluation results
  - Monitoring graph requires approximately 10% of memory of binary
  - Monitoring can be ambiguous
  - Detection fast when using hash4 (within 1 or few instructions)
- Effective method for ensuring correct processor behavior

## Outline

- Introduction
- Monitoring architecture for embedded systems
- Processing monitor
- **Collaborative monitoring**
- Summary

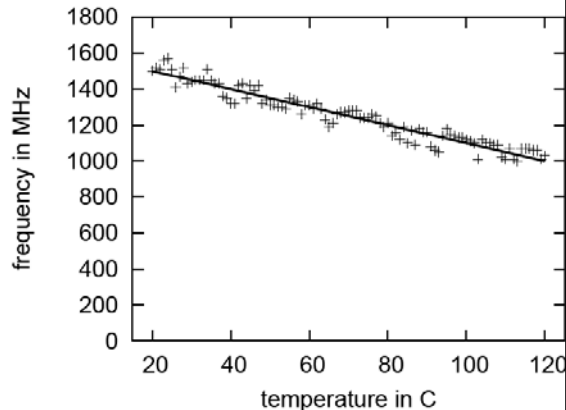
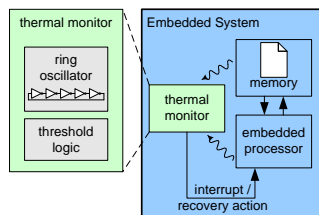
## Collaborative Monitoring

- Single monitor has only limited view of system
- Multiple monitors
  - Higher accuracy (e.g., lower false positives and false negatives)
  - Coupling of hardware and software monitors
  - Mutual calibration
- Control processor of monitoring system computes joint result
- Example
  - processing monitor and thermal monitor



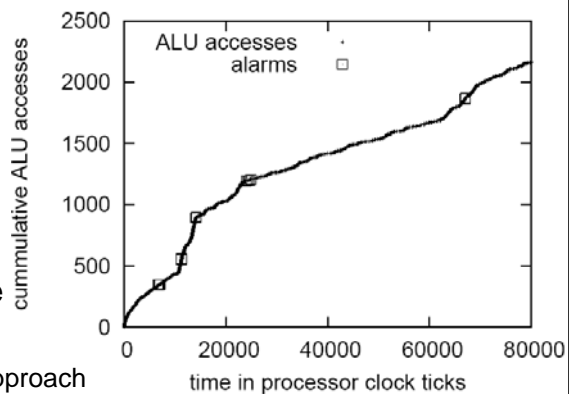
## Thermal Monitor

- Ring oscillator
  - Odd number of inverters in loop
  - Delay across inverter is temperature dependent
- Event counter
  - Infers heat dissipation



## Collaborative Decision

- Variation in processing changes heat dissipation
- Static temperature threshold problematic
  - Low: false positives
  - High: false negatives
- Processing monitor can identify when processing is "heat-intensive"
- Example: ALU access burst raise temperature
- Work in progress
  - Aiming for systematic approach



## Outline

- Introduction
- Monitoring architecture for embedded systems
- Processing monitor
- Collaborative monitoring
- **Summary**

## Summary

- Security considerations important in embedded systems
  - Characteristics of embedded systems lead to particular vulnerabilities
  - Novel attack space
- Monitoring architecture
  - Embedded monitors to identify abnormal behavior
  - Processing monitor
    - Hash pattern: small monitoring graph and fast detection
  - Collaborative monitors to correlate events
- Exciting new area
  - Many practical applications
  - Many interesting systems problems

## Questions?

- Contact information: [wolf@ecs.umass.edu](mailto:wolf@ecs.umass.edu)
- References:
  - Sri Parameswaran and Tilman Wolf, "Embedded systems security – an overview," *Design Automation for Embedded Systems*, vol. 12, no. 3, pp. 173–183, Sept. 2008.
  - Guy Gogniat, Tilman Wolf, Wayne Burleson, Jean-Philippe Diguët, Lilian Bossuet, and Romain Vaslin, "Reconfigurable hardware for highsecurity/ high-performance embedded systems: the SAFES perspective," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 2, pp. 144–155, Feb. 2008.
  - Shufu Mao and Tilman Wolf, "Hardware support for secure processing in embedded systems," in *Proc. of 44th Design Automation Conference (DAC)*, San Diego, CA, June 2007, pp. 483–488.
  - Tilman Wolf, Shufu Mao, Dhruv Kumar, Basab Datta, Wayne Burleson, and Guy Gogniat, "Collaborative monitors for embedded system security," in *Proc. of First International Workshop on Embedded Systems Security* in conjunction with *6th Annual ACM International Conference on Embedded Software (EMSOFT)*, Seoul, Korea, Oct. 2006.
- Available at: <http://www.ecs.umass.edu/ece/wolf/>