

# Collaborative Monitors for Embedded System Security

Tilman Wolf\*, Shufu Mao\*, Dhruv Kumar\*, Basab Datta\*, Wayne Burleson\* and Guy Gogniat†  
\*Department of Electrical and Computer Engineering † Laboratory LESTER  
University of Massachusetts, Amherst, USA University of South Brittany, Lorient, France  
{wolf,mao,kumar,bdatta,burleson}@ecs.umass.edu guy.gogniat@univ-ubs.fr

## ABSTRACT

This paper presents a hardware based approach to embedded systems security. Usage of on-chip monitors is proposed for identifying attacks on embedded systems by tracking the operation of the system. Two types of monitors: a processing monitor and thermal monitor are presented with their detailed operation and results. The effectiveness of this security system can be enhanced by correlating information from both monitors through collaborative decision making.

## 1. INTRODUCTION

The expansion of the Internet to a wide range of networked embedded devices introduces a myriad of security and reliability concerns. Already, the proliferation of network connectivity to mobile wireless "thin clients" has expanded the scope of potentially vulnerable systems to include critical infrastructure for finance, health, transportation, power, communications, and defense. Networked computer systems are known to be susceptible to a range of hacking and denial-of-service attacks and recent research efforts have focused mainly on conventional client and server systems and the Internet that interconnects them. The new generation of embedded systems will be characterized by remote and unmanned nodes and specialized computations, but also limited processing, communication, and power resources, which makes them inherently vulnerable to a new class of attacks.

In our recent work, we have proposed a novel systems architecture to support dynamic security on embedded systems by continuous on-chip monitoring of the operation of the system [10], [11]. The key aspect of this system is that verification and protection are provided in a dedicated hardware system. Thus security is provided by a subsystem that is fundamentally separate from the system that is under attack. Using a separate subsystem for attack detection not only helps provide a level of security that is not available with software solutions, but also reduces the impact on processing performance and energy consumption of the protected system.

In this paper, we address the important question of how to design monitors that can detect abnormal system behavior and protect the system from known and unknown attacks. We also discuss how

to use the information from multiple monitors to make joint decisions that lead to a more accurate detection of threads. Our specific contributions are threefold:

- **Design of a processing monitor.** This monitor tracks the processing steps the system's embedded processor core and compares them to the expected behavior of the application.
- **Design of a thermal monitor.** This monitor uses simple ring oscillators to track the temperature of a particular region of the chip to detect thermal attacks.
- **Collaborative monitoring.** We discuss an approach to integrating the information from multiple monitors to detecting attacks.

The remainder of the paper is organized as follows. Section 2 discusses related work in the area of embedded systems security. Section 3 briefly introduces the general architecture of our embedded systems monitoring system. Sections 4 and 5 introduce the processing and thermal monitors. Section 6 discusses collaborative attack detection and Section 7 summarizes and concludes this paper.

## 2. RELATED WORK

A system level approach to security is essential since it defines the security architecture that designers have to be aware of when building a new system [2], [7]. This can be done through security primitives and protocols that are used to guarantee privacy and integrity of users, these security methods are mainly defined through standards and are part of the system [28]. In [18] the security architecture to build a mobile embedded system is discussed and a new solution to deal with security issues is proposed. In their work they focus on security primitives and security protocols but they do not address the attack issue. Proximity-attacks are of major concern in embedded systems as presented in [3], [12], [21], and [26] and remote-attacks [9], [27] are even more dangerous since they can be performed at a large-scale through the network. In [7], [9], and [25] different types of remote-attacks are presented for embedded systems that are based on software or protocol weaknesses. In our work, we focus on protecting the system through monitoring its behavior, detecting abnormal activity, and to reacting when deviation to expected behavior exceeds some threshold.

In [20] and [16] a formalization of attacks on software is provided. Their approach is based on a graph definition that describes the potential paths to get access to sensitive information or codes for an existing application. This formalization gives designers a guide to build their system as no path must be found within their applications in order to provide a safe system. Our work builds on these approaches and shows a framework for efficient and high

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*First Workshop on Embedded System Security in conjunction with EMSOFT '06*, October 26, 2006, Seoul, South Korea.

Copyright 2006 ACM ...\$5.00.

performance detection and mitigation of many classes of attacks. In our approach the formalization is based on profiling to define expected behavior of the system and considering any significant deviation from normal operation an attack.

In the context of monitoring processing on embedded systems, using a secure co-processor to monitor operating system kernel data structures and invariants has been proposed by Zhang [29]. This approach requires a hardware co-processor that is implemented as a separate PCI card and is targeted at workstation computers rather than embedded systems. Arora et al. have proposed a system [4] that is conceptually similar to our work. The main difference is that their finest granularity of monitoring is the basic-block level due to the use of per-block hash values. In our work, deviations from the binary can be determined within a single (or a few) instructions. Suh et al. use the concept of “information flow” to track if data is considered authentic or spurious (i.e., potentially malicious) [22]. This system requires a much more complex design that needs to be integrated with the processor. Our solution operates in parallel to the embedded processor and has a very simple interface. Abadi et al. have proposed a control-flow integrity solution where binaries are modified to monitor processing [1]. In our approach to monitoring processing, we do not need change binaries. In stead of monitoring system security on the embedded processor itself, we use separate system resources (i.e., dedicated hardware monitors) to reduce the vulnerability of the system.

Another type of monitor that fits into our general architecture is bus protection as proposed in [30]. This bus protection aims to remove all data leakage as data is transferred between processor and memory. Their approach is based on control flow graph analysis which provides a unique signature of the application, which is conceptually similar to our approach of monitoring processing.

Davar et al. [8] described possible thermal attacks and thermal monitors are proposed in [6]. In [19] a dynamic critical path predictor enables reduced processor power consumption. These monitors and predictors track system behavior in the same spirit as our security-driven thermal monitor. In [23] temperature monitors are used at various on-chip locations to sense thermal fluctuations. Individual monitors are coordinated by a central controller. In both systems, communication with the main controller is not secure and hence is vulnerable to attacks. Our work uses monitors that can adapt threshold values dynamically depending on the current application running. This information is obtained through integration with the processing monitor.

### 3. SECURE EMBEDDED SYSTEMS

Attacks on embedded systems can be motivated by a number of different goals. The following list illustrates this point (but is not meant as a complete enumeration of all possible scenarios):

- Extraction of secret information (e.g., reading of cryptographic key material from a smart card).
- Modification of stored or sensed data (e.g., tampering with utility meter readings).
- Denial of service attack (e.g., reducing the functionality of a sensor network).
- Hijacking of hardware platform (e.g., reprogramming of TV set-top box).
- Damaging or destruction of device (e.g., overheating of chip in thermal attack).

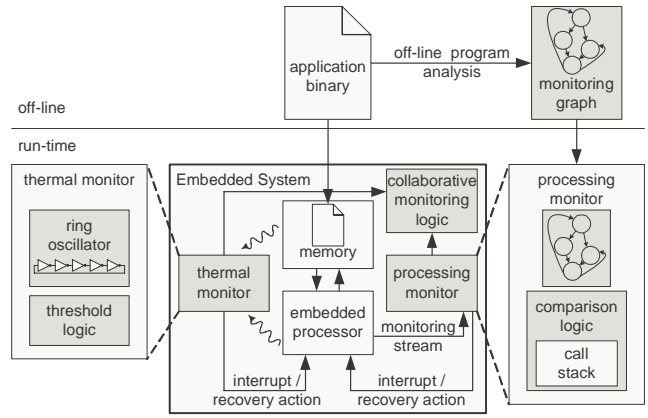


Figure 1: Monitoring Architecture on Embedded System.

In each of these cases, the attack relies on the ability to get access to the embedded system and change its behavior (i.e., change in 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. It is also important to observe that many of the attacks are exploiting vulnerabilities that are common to many different embedded system architectures.

The general concept of our hardware security approach is to augment conventional system-on-a-chip hardware with additional component for monitoring and defense mechanisms. The monitoring and verification capability of the system needs to be provided in hardware for several reasons. First, hardware can monitor the operations of the system at a much finer granularity than software (e.g., bus transactions). Second, hardware can easily correlate multiple events that occur in parallel on the system (e.g., I/O operation at the same time a private key is read from memory). Third, certain system functions can only be monitored by specialized hardware components (e.g., power drain, temperature, tampering, and soft errors from fault-induced attacks.). Software monitoring and verification is limited to the domain in which it operates, i.e. the CPU of the system, but hardware monitoring can expand the perimeter of security monitoring to the entire embedded system.

In our prior work, we have described the design of the monitoring subsystem infrastructure [10], [11]. We have proposed to use an on-chip network that connects all monitoring components. The details of this design go beyond the scope of this paper. Instead, we focus on two specific types of monitors: a processing monitor and a thermal monitor. Their operation and interaction with the embedded system is illustrated in Figure 1. The system architecture consists of four major components operating in parallel:

- **Conventional Embedded Processing Subsystem.** This part of the architecture consists of a general-purpose processor, memory, I/O, and any other components that are necessary to execute the embedded system application. The only addition to this part of the system is an extension to the processor core that continuously sends a stream of information to the processing monitor subsystem. There is also a feedback component from the each monitor system (and also from the collaborative monitoring logic – not shown) to the processor. In the case an attack is detected, the monitor can halt the processor and possibly initiate a recovery attempt.
- **Processing Monitor Subsystem.** This monitor compares the stream of information sent from the processor with the ex-

pected behavior derived from the off-line analysis of the binary. A “monitoring graph” represents the sequence of possible control flows between basic blocks. More detailed information about the processing steps within each basic blocks is also maintained. In order to be able to keep track of all permissible control flows, a call stack is necessary. If the comparison logic determines that there is a discrepancy between the stream of information from the processor and the monitoring graph, it determines that an attack occurred and initiates an interrupt to the processor. As indicated in the figure, the monitoring graph is generated in an off-line process, where the binary of the application is simulated and analyzed. The simulation is necessary to resolve some branch targets that cannot be determined statically. It is important to note that this process indeed only requires the binary and not the source code of the application.

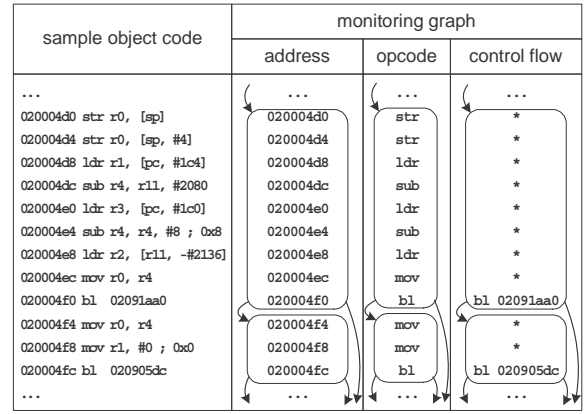
- **Thermal Monitor Subsystem.** This monitor collects temperature information at one or many points of the chip and uses it to determine if unusual or dangerous patterns warrant slowing the system clock or halting the processor. We propose to use a ring oscillator circuit that can measure temperature changes while being entirely implemented in CMOS. A simple threshold-based decision logic is used to detect critical conditions. The information from the thermal sensor is used in the collaborative monitoring logic.
- **Collaborative Monitoring Logic.** Each monitor is specialized to detect particular conditions and events. In order to more effectively avoid false-positives and false-negatives in the attack detection, the information of multiple monitors can be used to make a collaborative decision. In the case of processing and thermal monitors, for example, the thermal profile is much easier to track when information on the control flow within the program is available. This makes it feasible to correlate higher processor temperatures with processing-intensive tasks in the program without raising a false alarm.

The following sections describe each of the monitoring components in more detail and present results on their operation.

## 4. PROCESSING MONITOR

To achieve secure processing, our monitoring system verifies that the processor indeed performs the operations that it was intended to. In order for an attacker to abuse an embedded system, it is necessary to modify its operation in some way: either by adding a new piece of instruction code that performs malicious operations or by modifying the existing application to execute malicious code. Thus, in an off-line process, the binary code of an embedded system application is analyzed and an augmented control flow graph is obtained. Due to the simplicity of embedded systems workloads, it is possible to extract this information efficiently. During run-time, the embedded processor reports on the progress of application processing by sending a stream of information to the monitoring system. The monitoring system compares the stream to the expected behavior of the program as derived from the executable code. If the processor deviates from the set of possible execution paths, then it is assumed that an attacker has altered the instruction store or program counter to alter the behavior of the system.

There are several different approaches to what information the processor should provide to the monitor. We evaluate streams of instruction addresses, opcodes, and control flow operations. Figure 2 illustrates the information that is used in each case for each of the monitoring graphs:



**Figure 2: Examples of Monitoring Graphs for Different Information Streams.**

- The idea behind using the instruction address as an indicator for monitoring processing is that each instruction address is unique. Assuming instruction memory cannot be corrupted, a program must follow exactly the same sequence of instructions as it had been programmed to do. Using addresses, however, is vulnerable for the same reason. If an attacker can replace parts of the application code with a sequence of instructions that has the same basic block structure as the original, this change goes undetected. This vulnerability is due to the pattern using no information on what instructions are actually executed on the processor.
- In contrast to the address pattern, the opcode pattern focuses solely on the operations that are performed on the processor. The intuition behind using this information for monitoring processing is that the sequence of operations parallels the underlying functionality of the program. An attacker would need replace instructions with malicious instructions that use the same opcodes (but possibly different operands) in the same sequence. This type of attack is likely to be more challenging than in the case of the address pattern.
- Another intuitive pattern is the control flow pattern. In this pattern, all control flow operations are stored (e.g., branches, calls, returns) including their branch targets if applicable. This allows the monitor to track any change in the program counter, but exhibits a similar vulnerability as the address patterns since there is no information exchange on the actual operation of the processor. In related work, similar information is used to monitor processing [4]. In some cases the control flow information is limited to system calls. We consider control flow at the level of basic blocks.

Given the monitoring graph that matches one of the patterns from above, the comparison logic can verify that the processing on the embedded system follows a possible path of execution:

- **Monitoring within a Basic Block:** Within a basic block, the comparison logic simply follows the sequence of patterns that is stored in the monitoring graph. For example, in the case of the opcode pattern, the monitor compares the opcodes reported by the processor to those in the current basic block of the monitoring graph. If wildcards are used (e.g., for the control flow pattern), any instructions reported by the processor can match the wildcard, except those that are part

of the pattern (loads and stores in this case). The necessary logic is straightforward to implement since it comes down to a simple comparison between what the processor reports and what is stored in the monitoring graph. Note that each wildcard replaces exactly one single instruction and not an arbitrary number of instructions. Thus, there is no ambiguity in this comparison process.

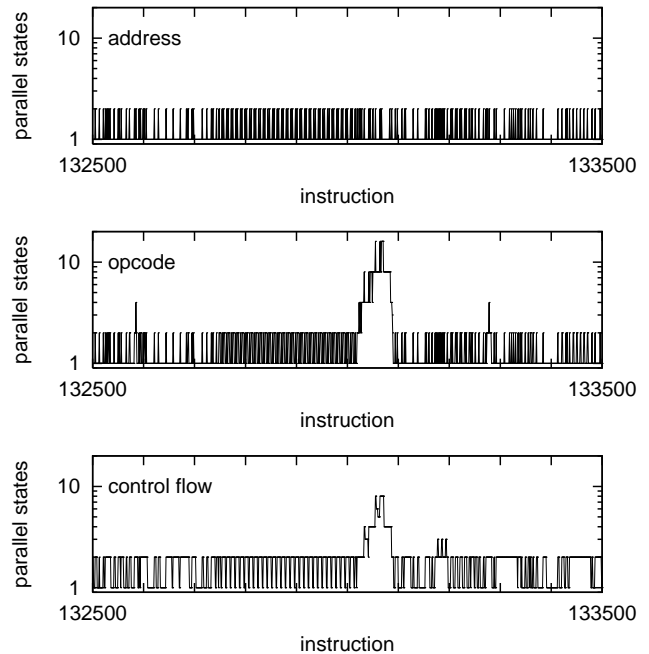
- Monitoring between Basic Blocks:** When the end of a basic block is reached, control flow branches to one of up to two targets. The monitoring logic does not replicate the data path of the processor and thus cannot determine which branch is taken. Also, information from the processor should not be used to make this decision to avoid being influenced by an attacker. Instead, the comparison logic allows for multiple parallel execution paths. That is, the monitor allows the current state of execution to be in multiple locations in the monitoring graph at the same time. As monitoring progresses, some of these states turn out to be invalid and thus are pruned from the set of concurrent states. If all states lead to invalid comparisons, then an attack is detected.

To illustrate the monitoring process between basic blocks, consider an opcode monitor at the end of a basic block where the current instruction is a conditional branch. In the next instruction, the processor either jumps to the branch target (e.g., an *add* instruction) or continues with the following instruction (e.g., a *sub* instruction). After validating the branch, the opcode monitors allows both following instructions to be valid states. If either an *add* or a *sub* is reported by the processor, the monitor accepts it as correct. At the same time, the path that does not match gets pruned. Depending on the code of the application, the duration for which the monitor is in an ambiguous state varies. As a result, detection of possible attacks can be drawn out until all ambiguity is removed and the monitor is certain that a reported processing sequence is invalid.

To quantify the performance, we have simulated the processing monitor on an ARM instruction set processor. We use the MiBench benchmark suite [13] to generate realistic workloads for an embedded system scenario. The SimpleScalar simulator is used to extract the relevant monitoring information that is passed on to the monitor. We have tested the monitoring for the majority of applications in the MiBench suite, but only present results from the *patricia* application due to space constraints.

There are two metrics that are important to consider: size of the monitoring graph and speed of detection. The size of the monitoring graph determines the overhead of the monitor compares. The comparison logic in the processing monitor is simple and can be implemented with few system resources, the memory to store the monitoring graph can be present a considerable overhead. In the *patricia* application, the size of the monitoring graph is 150kB for the address pattern and 120kB for the opcode pattern and for the control flow pattern. In comparison, the complete binary for the application is 1.1MB. This means that the secure monitor has an overhead of approximately 11–13% in terms of memory requirements. For other applications, similar overheads of around 10% were observed.

In terms of how well a deviation from normal program behavior can be detected, there are two metrics that can be considered. The number of ambiguous states that are present in the monitor due to branches is shown for a sample of 1000 instructions in Figure 3. This shows that there are phases of no ambiguity interleaved with phases of heavy ambiguity. A different metric is the measure of how long an ambiguous state lasts. This reflects how long the monitor cannot be entirely sure that the program execution is correct.



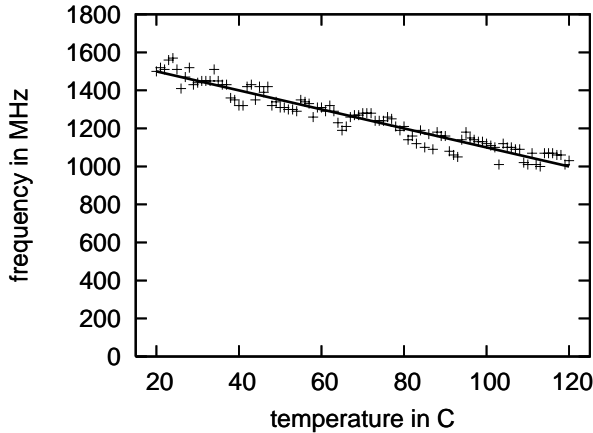
**Figure 3: Snapshot of Monitoring of 1000 Instructions in *patricia* Application.**

The address pattern show only one step of ambiguity at the point of a conditional branch. For the opcode pattern, 93% of ambiguous phases are shorter than 10 instruction and 96% are shorter than 100 instruction. For the control flow pattern, only 62% of ambiguous phases are shorter than 10 instructions and 99% are shorter than 100 instructions.

## 5. THERMAL MONITOR

The notion of monitoring certain characteristics of embedded systems for secure processing can also include temperature sensing. This information can be used to gather more information to detect anomalies and attacks. With the increasing sophistication of embedded systems and the higher power density values in chips, temperature-related effects and attacks are becoming increasingly important. For example, disk drives in embedded systems (e.g., iPod) are severely susceptible to erroneous operation at high temperatures – if ambient temperature increases by 5 degrees Celsius over the designed value, disk drives are 15% more likely to crashing [24]. Thermal attacks in processors were elucidated by Dadvar and Skadron [8]. Since there is often no place for forced air cooling systems in embedded systems due to shrinking form factors and portability requirements, thermal problems continue to be present despite low-power component designs. Malfunctioning of devices due to thermal conditions can facilitate intrusion and hence it is necessary to incorporate schemes to detect and prevent such attacks. Sensors that can detect abnormal thermal conditions can help prevent attacks as well as non-malicious misuse of the processor (e.g., process variations, design flaws etc.).

Modern Dynamic Thermal Management (DTM) systems use thermal transducers to obtain accurate temperatures of the chip and then dynamically scale the frequency and voltage. Some of the sensors commonly used are thermal diodes and ring oscillators. Ring oscillator based sensors provide a digital output and can be used to design sensors for embedded microprocessors. The ring oscil-



**Figure 4: Dependence of Frequency on Temperature for a Ring Oscillator.** The oscillator uses 11 stages and assumes 65nm CMOS technology.

lator is basically a circuit consisting of a series of odd number of inverters with the output of the last inverter serving as the input to the first. This system is inherently unstable and oscillates at a frequency dependent on the delay across each inverter, which again is dependent on the temperature. Figure 4 shows how the frequency changes with the junction temperature, thus qualifying the oscillator as an effective thermal transducer. The power supply noise sensitivity of the sensor can be mitigated by increasing the number of inverters in the oscillator. The resolution of ring oscillators—close to 2 degrees Celsius—is excellent due to the high linearity over the temperatures of interest.

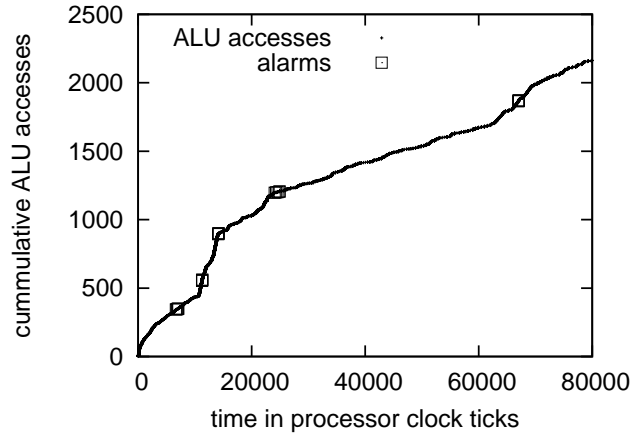
An alternative approach to monitoring temperature is to use hardware access counters [15] as a proxy for temperature. Architectural level power simulators like Wattch [5] use access count as a way to compute the total power. Both approaches to monitoring temperature are useful in the context of merging temperature information with processing information to achieve collaborative monitoring.

The overhead for implementing thermal monitors is low. When using access counters, access rates need to be maintained for different resources. But the time constants involved in hotspot generation are large, and tracking can be done through sampling (e.g., every 1000 cycles [14]). To improve the space efficiency of the access monitors, a running average of access rate values could be maintained instead of storing all of the values. The hardware infrastructure required to monitor access rate behavior comprises of one counter, a few registers (depending on the level of monitoring granularity), and some peripheral arithmetic logic.

A VLSI implementation of a ring-oscillator-based thermal sensor requires very little area and power overhead (e.g., 0.5% of die area and 0.5% of the power consumption of a processor [17]). Such implementations provide temperature information over a range of 20–80 degrees Celsius with an accuracy of 3 degrees. It is possible to calibrate such circuits against different power supply voltages, circuit die temperature, and manufacturing variations to further improve accuracy.

## 6. COLLABORATIVE DETECTION

The goal of attack detection through collaborative monitoring is to reduce the false-positive and false-negative rates that would be achieved by using independent monitors. In our scenario of processing and thermal monitors, temperature information can be



**Figure 5: Points where ALU Access are Dangerously High and Cause an Alarm.**

correlated with the monitoring graphs in the processing monitor. Specifically, the monitoring graphs can identify program regions that use more power (and thus dissipate more heat) in certain components than others. If thermal information was used without the processing context, the static alarm threshold would either be too conservative (i.e., causing false alarms in regions of intense processing) or too optimistic (i.e., opening avenues for potential attacks that cannot be detected).

Figure 5 shows a scenario with the *anagram* application, where ALU accesses are monitored. Using access counters (or ring oscillators in a actual implementation), the temperature of the ALU can be estimated. If a static threshold for monitoring was used, then processing regions with frequent accesses trigger an alarm. These regions are marked in Figure 5. Clearly it is undesirable to get this large number of false alarms. By raising the threshold, the number of alarms can be reduced. The drawback is that in most programs that are short periods of intense ALU access. If the threshold was raised so high as to not cause false alarms in these regions, the sensitivity of the thermal monitor would no longer be useful.

In a collaborative monitoring approach, the run-time information on valid program operation obtained from the processing monitor can be used for determining the a suitable threshold. For example, the opcode information stream (see Figure 2) can be used to identify ALU operations. By maintaining a suitable counter of recent history of ALU accesses, the temperature can be estimated and compared to the actual value obtained from the thermal monitor. This approach allows a change in temperature in different program regions and thus does not display the problems that are encountered when using a static threshold.

## 7. SUMMARY

In this paper, we have presented two types of hardware-based on-chip monitors for embedded systems that can track processing operations and thermal signatures. We have shown how each system operates and is able to identify a certain set of attacks. By using collaborative monitoring, where information from both types of monitors is correlated, we can improve the effectiveness of the monitors and increase the security of the embedded system.

## 8. REFERENCES

- [1] M. Abadi, M. Budiu, Ú. Erlingsson, and J. Ligatti. Control-flow integrity principles, implementations, and

- applications. In *ACM Conference on Computer and Communications Security*, pages 340–353, Alexandria, VA, Nov. 2005.
- [2] R. J. Anderson. *Security Engineering, A Guide to Building Dependable Distributed Systems*. Wiley, Jan. 2001.
- [3] R. J. Anderson and M. G. Kuhn. Low cost attacks on tamper resistant devices. In *Proceedings of the 5th International Workshop on Security Protocols*, volume 1361 of *Lecture Notes In Computer Science*, pages 125–136. Springer-Verlag, 1998.
- [4] D. Arora, S. Ravi, A. Raghunathan, and N. K. Jha. Secure embedded processing through hardware-assisted run-time monitoring. In *Proc. of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05)*, pages 178–183, Munich, Germany, Mar. 2005.
- [5] D. Brooks, V. Tiwari, and M. Martonosi. Watch: A framework for architectural-level power analysis and optimizations. In *Proc. of ACM ISCA-27*, pages 83–94, Vancouver, BC, June 2000.
- [6] E. Chi, A. M. Salem, R. I. Bahar, and R. Weiss. Combining software and hardware monitoring for improved power and performance tuning. In *Proceedings of the Seventh Workshop on Interaction between Compilers and Computer Architectures (INTERACT '03)*, page 57, Anaheim, CA, Feb. 2003.
- [7] B. Cole. Balancing cost vs. security for embedded design. Technical report, EE Times, Sept. 2005. <http://www.eetimes.com/showArticle.jhtml?articleID=170102341>.
- [8] P. Dadvar and K. Skadron. Potential thermal security risks. In *Proc. of Twenty First Annual IEEE Semiconductor Thermal Measurement and Management Symposium*, pages 229–234, Mar. 2005.
- [9] T. Dagon, T. Martin, and T. Starner. Mobile phones as computing devices: the viruses are coming! *IEEE Pervasive Computing*, 3(4):11–15, Oct. 2004.
- [10] G. Gogniat, T. Wolf, and W. Burleson. Reconfigurable security primitive for embedded systems. In *Proc. of International Symposium on System-on-Chip (SOC)*, Tampere, Finland, Nov. 2005.
- [11] G. Gogniat, T. Wolf, and W. Burleson. Reconfigurable security support for embedded systems. In *Proc. of 39th Hawaii International Conference on System Science (HICSS-39)*, Poipu, HI, Jan. 2006.
- [12] S. Guilley and R. Pacalet. SoC security: a war against side-channels. *Annals of Telecommunications*, 59(7–8), July 2004.
- [13] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *Proc. of IEEE 4th Annual Workshop on Workload Characterization*, Austin, TX, Dec. 2001.
- [14] J. Hasan, A. Jalote, T. N. Vijaykumar, and C. E. Brodley. Heat stroke: Power-density-based denial of service in SMT. In *Proc. of the 11th International Symposium on High-Performance Computer Architecture (HPCA)*, pages 166–177, San Francisco, CA, 2005.
- [15] K.-J. Lee and K. Skadron. Using performance counters for runtime temperature sensing in high-performance processors. In *Proc. of International Symposium on Parallel and Distributed Systems (IPDPS)*, 2005.
- [16] P. Manadhata and J. M. Wing. An attack surface metric. Technical Report CMU-CS-05-155, Department of Computer Science, Carnegie Mellon University, July 2005.
- [17] R. McGowen, R. A. Poirier, C. Bostak, J. Ignowski, M. Millican, W. H. Parks, and S. Naffziger. Power and temperature control on a 90-nm Itanium family processor. *IEEE Journal of Solid-State Circuits*, 41(1):229–237, Jan. 2006.
- [18] P. Schaumont and I. Verbauwhede. Domain-specific codesign for embedded security. *IEEE Computer*, 36(4):68–74, Apr. 2003.
- [19] J. S. Seng, E. S. Tune, and D. M. Tullsen. Reducing power with dynamic critical path information. In *MICRO 34: Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitecture*, pages 114–123, Austin, Texas, Dec. 2001.
- [20] O. Sheyner and J. M. Wing. Tools for generating and analyzing attack graphs. In *Proc. of Second International Symposium on Formal Methods for Components and Objects*, volume 3188 of *Lecture Notes in Computer Science*, pages 344–372, Leiden, The Netherlands, Nov. 2003.
- [21] F.-X. Standaert, L. van Oldeneel tot Oldenzeel, D. Samyde, and J.-J. Quisquater. Power analysis of FPGAs: how practical is the attack? In *Proc. of 13th Conference on Field Programmable Logic and Application*, volume 2778 of *Lecture Notes In Computer Science*, pages 701–711. Springer-Verlag, Sept. 2003.
- [22] G. E. Suh, J. W. Lee, D. Zhang, and S. Devadas. Secure program execution via dynamic information flow tracking. In *ASPLOS-XI: Proceedings of the 11th international conference on Architectural support for programming languages and operating systems*, pages 85–96, Boston, MA, Oct. 2004.
- [23] S. Velusamy, W. Huang, J. Lach, M. R. Stan, and K. Skadron. Monitoring temperature in FPGA based SoCs. In *Proc. of 23rd International Conference on Computer Design (ICCD 2005)*, pages 634–640, San Jose, CA, Oct. 2005.
- [24] W. Webb. Take the heat: Cool that hot embedded design. Technical report, EDN, May 2004. <http://www.edn.com/article/CA415105.html>.
- [25] J. M. Wing. A call to action: look beyond the horizon. *IEEE Security and Privacy Magazine*, 1(6):62–67, Nov. 2003.
- [26] T. Wollinger, J. Guajardo, and C. Paar. Security on FPGAs: state-of-the-art implementations and attacks. *Transactions on Embedded Computing Systems*, 3(3):534–574, Aug. 2004.
- [27] A. Wood and J. A. Stankovic. Denial of service in sensor networks. *IEEE Computer*, 35(10):54–62, Oct. 2002.
- [28] C. Xenakis and L. F. Merakos. Security in third generation mobile networks. *Computer Communications*, 27(7):638–650, May 2004.
- [29] X. Zhang, L. van Doorn, T. Jaeger, R. Perez, and R. Sailer. Secure coprocessor-based intrusion detection. In *Proc. of Tenth ACM SIGOPS European Workshop*, Saint-Emilion, France, Sept. 2002.
- [30] X. Zhuang, T. Zhang, and S. Pande. HIDE: an infrastructure for efficiently protecting information leakage on the address bus. In *Proc. of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XI)*, pages 72–84, Boston, MA, Oct. 2004.