Multi-Core Embedded Controller Security Architecture with Instruction Stream Detection

Xiaosheng Wang, Yanchun Yang

School of Data Science and Computer, Shandong Women's University, China

Summary
The paper presents a design scheme for multi-core embedded controller security architecture and a control application runtime unexpected behavior detection method based on instruction stream. In the architecture design, the partition and isolation functions of the hypervisor are used to consolidate the security-critical system and the non-security-critical system into a single hardware to be handled by different cores. The two cores cooperate with hardware-based instruction stream tracking module dynamically monitors whether the behavior of the control application executed by the monitored core deviates from the expectation to implement secure and reliable seamless control. The compatibility of different instruction sets and the impact of interrupt on detection are considered synthetically in the control application security detection algorithm. Finally, the effectiveness of the proposed method is verified by simulation experiments on ball and plate system.

Key words:
Multi-core embedded controller, security Architecture, Hypervisor, Consolidating control unit, Instruction stream detection

1. Introduction

In the field of modern industrial security-critical control, such as aerospace, nuclear energy, automobile, Internet of things, etc., embedded system is usually used as the controller of the physical plants. The application of the embedded controllers also brings about security problems for industrial control system, such as information tampering, virus attack, etc., which leads to property loss, environmental damage and even casualties due to control failure. Recently, there have been many successful cases of security attacks for embedded control systems. For example, Iran's nuclear power plant was infected by W32.Stuxnet making the nuclear facility fail, malicious code was injected into the information communication unit of modern cars resulting in brake failure, unmanned aerial vehicles were hijacked, and smart home equipment in a family was controlled by hacker [1],[2],[3],[4]. Therefore, the security of embedded control system has attracted great attention from the industry [5],[6].

Recently, a trend is that high-performance embedded multi-core processor with low power consumption and small size (single board instead of multi-board) is used increasingly in the security-critical industrial control systems [7],[8]. However, in the multi-core architecture, some resources (e.g. Cache, Bus, Memory and other components) are shared among many cores, which may lead to security vulnerabilities. In addition, in order to adapt to the needs of high-performance embedded multi-core system, application software is complicated to design, and it is easy to result in software vulnerabilities, and attacker exploit the vulnerability to disrupt the application behavior by executing unplanned code or malicious code to endanger the system security. Therefore, a comprehensive solution is needed to enable embedded multi-core processor to work in a real-time control system in a secure and reliable manner. This paper proposes a design method to improve the overall security of the system by taking advantage of the hypervisor and considering the nature of multi-core processors (namely parallel cores and the convenience they provide).

The remaining sections are organized as follows: in section 2, the related work is described. A multi-core embedded controller secure architecture and its working principle are proposed for the security-critical control application in section 3. Section 4 presents the dynamic security detection algorithm based on instruction stream. The experiment and result analysis are given in section 5, and the conclusion and future work are proposed in section 6.

2. Related Work

At present, the architecture security design for multi-core embedded system mainly draws on the technology of trusted auxiliary hardware [9],[10] to monitor the execution behavior of a real-time control application running on an untrusted main system. Li and Nakajima propose a multicore embedded secure architecture based on Limited Local Memory (LLM) [11], which uses a privileged core with the LLM executing a integrity testing tool to monitor state of target Operating System (OS) running on other core, but they did not give specific detection methods for target OS and application integrity. Ragel et al present a dual-core embedded processor architecture (SecureD) [12], which can both detect application integrity through calculating checksum for each basic block and prevent the power analysis attack

Manuscript received May 5, 2019
Manuscript revised May 20, 2019

through using two processor cores running complementary encryption program to defense power analysis attack, however, its disadvantage is need to change the original binary code and instruction set, and not have versatility.

In security detection technology for application, most of the research is focused on control flow detection and the program execution time detection. Literature [10] and [13] use reconfigurable logic connected to the main CPU to detect the integrity of program code executed by the CPU, but a specific detection algorithm is not given, and this method is not universal. Abadi et al. [14] propose a control flow detection method based on the hardware without changing the processor core, which can check out the basic block granularity of malicious code by using the output processor instructions or information to determine whether a control flow graph legal [15],[16], but this approach is insensitive to unexpected code which does not change the control flow (e.g., non-jump instruction overrides). Literature [17] proposed a multi-security strategy combining information flow tracking and memory monitoring based on hardware, which can detect more malicious attacks, however, it needs to change the microarchitecture of the embedded system and not easily applicable to multi-core embedded architecture. Sanjeev et al [18] proposed a fine-grained control flow integrity detection method at the basic block level to prevent malicious attacks based on buffer overflow, but this strategy only considered the detection of function calls and jump instructions, but it did not include the detection on interrupts. Mohan et al [19] used the method based on the program execution time to detect malicious behavior, but this method requires application structure fixation and need to change the original binary program (insert additional check code) [20] and causes large detection latency. Instruction flow detection method can comprehensively solve the contradiction faced by control flow detection between detection coverage and software interference, which has gradually become an important research. However, the current literature does not consider the compatibility of security monitors to processors of different instruction sets and the impact of interrupts on detection.

This paper firstly proposes a multi-core embedded controller architecture based on hypervisor and instruction stream detection mechanism referring to the S3A [19], and then implements abnormal behavior detection method for the control application program, which comprehensively considers the impact of interrupt on algorithm and the universality of algorithm to processors with different instruction set. Therefore, our work is different from the above because we focus on the domain-specific characteristics of these systems, and in particular their interrupt real-time nature.

As the main contributions of this paper, we propose the secure multi-core embedded controller architecture where,

(i) In the architecture design, leveraging hypervisor features and multi-core conditions, the security-critical real-time system and non-real-time system are consolidated into a single hardware with minimum cost, and the consolidation allows for spatial (memory area) and temporal (time) isolation between applications (processor cores) to guarantee software executing in parallel.
(ii) Security detection with minimum granularity based on instruction stream.
(iii) Instruction set independence design to achieve compatibility of processors with different instruction set.
(iv) The security detection component is independent of the control application.

3. Secure Multicore Embedded Controller Architecture

A proposed architecture of secure multi-core embedded controller with hypervisor for control systems, as shown in Fig. 1, which are capable of detecting abnormal behavior in real time during the execution of the control application and ensuring seamless control for physical plants. The architecture consists of three main components: i) security-critical system, ii) non-security-critical system, iii) the hypervisor. The security-critical system is consists of core0 and core1, RTOS (e.g., µCOS), and the on-chip Instruction Stream Trace Module (ISTM). The non-security-critical system consists of core2 and embedded general-purpose OS (GPOS).

The hypervisor provides a virtualized hardware resource for the architecture through partition and consolidation, in which each core is divided into a virtual machine VM0, VM1, and VM2 respectively [21]. The isolation of space and time and resource utilization is supervised through the interface provided by hypervisor between the hardware and the application to work independently for each application. In the architecture, core0 and core1 are allocated to (used as) security-critical system and non-security-critical system respectively taking into account the requirements of real-time systems. To protect a security-critical system from malicious tampering by a compromised non-security-critical system, the hypervisor provides a clean isolated memory space by designing the MMU, and the hypervisor itself runs in its own protected memory space. So any attempt to access memory across partitions is blocked by the hypervisor. (Note, this paper assumes that the hypervisor is part of the trusted base and that there is no malicious code in it.)

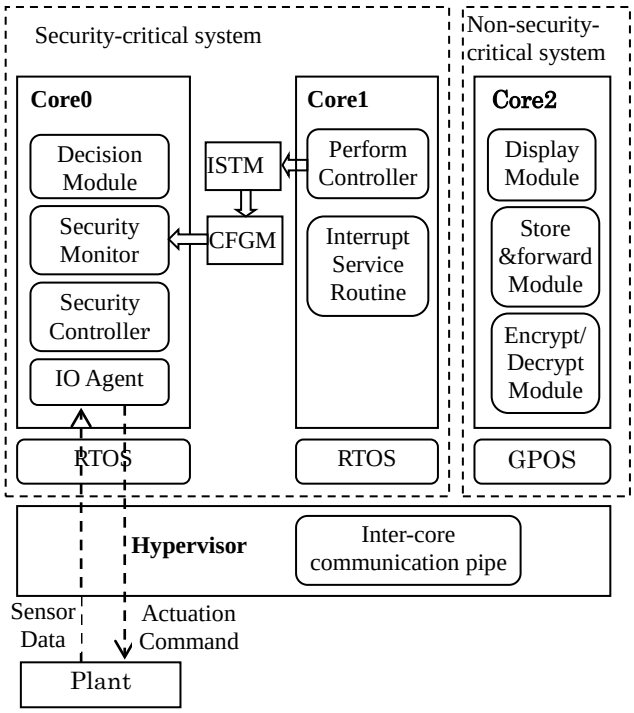

Fig. 1 Secure architecture multi-core embedded controller with hypervisor

3.1 Security-critical System-Core0 and Core1

In the security-critical real-time system, core0 (secure core) is privileged to run the embedded real-time operating system (RTOS), and it combines with the ISTM to monitor whether the execution behavior of the core1 (monitored core, responsible for data acquisition and execution control application) is against expectations. The core0 mainly executes four modules (processes): security controller, decision module, security monitor and IO Agent module. The core1 mainly includes perform controller (a process, essentially a controller that manages the physical plant) and interrupt service routines. Sensor data from the physical plant is fed to both the perform controller and the security controller by the trusted IO Agent process, and each controller using its own internal control logic to calculate the actuation command. The decision module then may forward the appropriate commands to the physical plant according to its pre-computed security envelope. Under normal case, the plant is actuated by command from the perform controller in core1. However, when an abnormal or unexpected action of the perform controller is detected, the control operation is transmitted to the security controller in core0 in order to keep the physical plant reliable seamless control. Through this mechanism, the reliability of the control action can be guaranteed by the decision module and the security controller (they can be formally verified) based on all entities in the security system trusted.

In the context of memory protection, I/O channel is designed between the processor and plant. The channel is managed by IO Agent module (IOA) which runs on the secure core0 to manage all I/O of the physical plant to prevent the I/O data confusion caused by malicious code on other cores.

3.2 Instruction Stream Detection Components

We define set BLOCK to represent all basic blocks in the control application, and set R is the transition relationship between all basic blocks in the control application, so the control application can be represented as Control Flow Graph G = {BLOCK, R}. The running of the application should conform to the set G, and instructions inside the base block are always executed sequentially. Thus, monitoring the instruction stream executed by the perform controller, if the executing instructions violate the G or do not conform to the order relationship of instructions in the basic block, it is an unplanned instruction to perform unexpected acts. Accordingly, the component structure designed hardware-independent unexpected acts detection is shown in Fig. 2. The relevant components are described below.

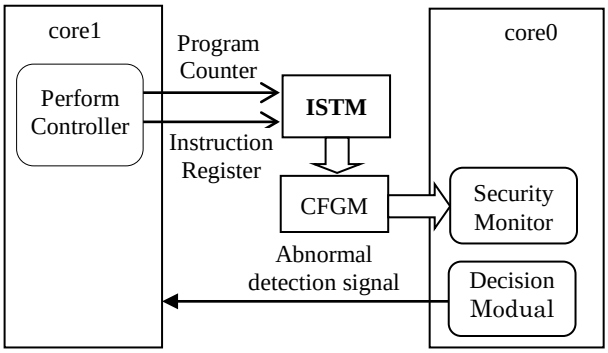

Fig. 2 Instruction stream tracking module

(i) ISTM. ISTM is a special on-chip hardware unit, which is directly attached to the monitored core1 to fetch the instruction stream from the perform controller at runtime by Instruction Register and Program Counter. The ISTM is used at two different stages in the system: (a) in the development/test (in the security environment), it is used to collect and organize (i.e., normalize) the original CFG (OCFG) profile of the application, and stored in the CFG memory (CFGM), and (b) when the system is actually deployed in the control field, the ISTM fetches the instruction stream outputted from the perform controller in real time and to normalize the instructions into real-time CFG (RCFG) profile that matches with the OCFG profile and stored into CFGM for comparison operations.

(ii) Instruction normalization. Essentially, the detection is to know whether the instruction sequence executed by the perform controller conforms to the expectation and insensitive to the specific content of the instruction, i.e., if it can distinguish the instruction sequence then it can meet the demand. Thus, substitution expressions of the instruction can be implemented by hashing. Assume that *ins* represents binary instructions, $y = hash(ins, x)$ represents that the hashing of x bit length is given to *ins*, and if x is given an appropriate value then different *ins* correspond to different y value, so it is a kind of equivalent substitution for distinguishing instruction sequence, and the length of y has nothing to do with the length or format of *ins*. Accordingly, the instruction normalization is a $hash(ins, x)$ operation to *ins*, i.e., OCFG profile is a result of using $hash(ins, x)$ value to substitute instruction *ins*. In this way, the comparison process is independent of the instruction set of the embedded processor to make the detection method has universal applicability.

(iii) CFGM. CFGM is a memory that stores the OCFG profile and the RCFG profile and mapped to the core0's address space area by the hypervisor.

(iv) Compare operations. In the application is running, the security monitor in the core0 fetches the instructions of the application through the ISTM and normalizes them into the RCFG profile and then compares with the OCFG profile, if they don't match, the *abnormal detection signal* will be output to the monitored core1, and the security core core0 will carry out subsequent processing such as takeover control, etc.. Specific detection method is described in section 4.

3.3 Non-security-critical System-Core2

In our architecture shown in Fig1., core2 is assigned to a non-security-critical system to be responsible for the execution of non-real-time parts (or subsystems) of a control system, such as the vehicle management subsystem in the real-time vehicle control system, or the functional parts such as information display, information storage and information forwarding, which runs embedded GPOS such as μClinux et al.

In addition, the pipe provided by hypervisor is used to complete communication between the non-security-critical system and the security-critical system. Since the pipe is finished full-duplex communication, two operating system domains can send messages to each other through it. At the same time, using the hypervisor completes isolation between operating system domains or the cores to ensure the controller security.

3.4 Startup and Execution Flow of the Secure Controller

In a multi-core embedded system, only one core will start run at startup (power on or reset), and other cores are then started after all the hardware checking and initialization is done. However, in a virtualized system based on hypervisor, the hypervisor should know the number of cores that it will manipulate, and the hypervisor plays the role of starting the cores. Therefore, the startup sequence including the hypervisor has to be modified. The startup and execution flow of the designed secure controller as shown in Fig. 3. Among them, the execution process of components of the security-critical system is described in section 3.1.

4. Instruction Stream Dynamic Detection Algorithm

The main ideas to illustrate whether the architecture is secure at run time is whether the instruction (normalized) fetched by the ISTM hit the path expected by the OCFG profile for security detection. Since the instructions in the basic block are always executed sequentially, when the instruction comparison is made in the basic block, the instruction output by the perform controller are compared with an exact instruction in the current position of OCFG profile, and the conclusion is clear. However, the CFG may contain multiple transfer branches at general transfer instruction and interrupt, so how to carry out instruction comparison is the key to the detection algorithm.

4.1 A General Transfer Detection Algorithm

When the instruction comparison process is made to last instruction in the basic block of OCFG profile, it determines actual execution path of the application by exclusive method: as the instruction compares with all possible subsequent branch pointed out by the OCFG profile, the matched branches are reserved and non-matched branches are cut off constantly until only one branch is left, which is the actual executive branch. Since subroutine involves a return problem, an address stack Sub_{stack} is kept to determine the subroutine return address. Algorithm 1 gives a branch matching process based on false branch pruning in a general transfer branch as an example, and the remaining branches are similar and run in parallel. In the algorithm, R_n is the nth instruction in the OCFG profile, M_n is the nth instruction current executed by the perform controller, b_i is a flag whether branch i is matched, and R_{bi_k} is the kth instruction of branch i.

Algorithm 1 Falsebranch_cutoff. -Algorithm of false branch pruning and branch match in general transfer branch.

```
//if Rn is last instruction in the basic block:
   Identify entry of branch_i from CFG profile
   b_i=1
   if branch_i is an entry returned by a call then
      push(Rn, Sub_stack)   // Push the Rn address
         goto L1
      else
L1:       k=0
         while b_i_k≠0 do
            n=n+1
            k=k+1
            if Mn≠R_bi_k   then
               b_i=0
               end match with the branch
            end if
         end while
    end if
return
```

4.2 Interrupt Detection Algorithm

Interrupt processing is often used in real-time control systems, so interrupt detection is very important. Interrupt is also a kind of transfer instruction, but because of its randomness, and each instruction is likely to be an interrupt instruction in the comparison process, the general transfer detection method cannot be adopted. In the paper, the fallback method can be adopted for the interrupt detection, and the basic idea of the method is that if the instruction executed by the perform controller does not match the OCFG profile with the normal process, we then check whether it conforms to the interrupt process, if so, the specific interrupt service program is determined, and the comparison will continue, if not, the unplanned instruction execution is determined. There is a case the embedded system has entered an interrupt before the comparison process finds an instruction mismatch. To solve the problem, the method to back q instructions is adopted, that is, at the executing nth instruction I_n, if the comparison process find I_n different from the expected instruction J_n of the OCFG profile, then the process of (I_n-q) to I_n is checked to see whether it has entered the interrupt service program. Since the system usually has more than one interrupt service program, the exclusive method also is adopted to determine the actual execution path of the application, i.e., the instruction comparison is operated for all possible interrupt service programs indicated by OCFG profile, and cuts off non-matched interrupt branch until only one interrupt branch is left. Algorithm 2 takes one interrupt branch as an example, and the other interrupt branches are processed similarly and executed in parallel. In this algorithm, *IF_i* is a flag whether the interrupt branch I is matched, R_{inti_k} represents the k-th instruction of interrupt branch I, *back* represents the number of backtracking instruction and its initial value is q (q=0,1,2,3...), and S represents a flag whether the interrupt branch is matched.

Algorithm 2 Int_JudgAndIntB_Cutoff. -Interrupt judgment and interrupt branch pruning algorithm.

```
if Mn≠Rn   then
   identify entry of INT_i from OCFG profile
    IF_i=1
    push(Rn, Int_stack)          //Push the Rn address
    k=1
    back=q
    x=0
    while M_n-x≠R_int_k   do
        if   x<q   then
            x=x+1
        else
            goto L1
        end if
    end while
    whie IF_i_k≠0 do
        n=n-x+1
        k=k+1
        if Mn≠R_inti_k   then
L1:        IF_i=0
           S=0
           end the match with the interrupt branch
        end if
    end while
    S=1
end if
return
```

4.3 Comprehensive Algorithm for Instruction Stream Detection

Based on the above considerations, the comprehensive algorithm for instruction stream detection designed is shown in Algorithm 3. The algorithm is executed when the controller runs the control application.

Algorithm 3 Comprehensive detection algorithm.

```
for the executing instruction Mn is not the end of the entire program do
    if Mn≠Rn then
       call Int_JudgAndIntB_Cutoff
       if S==1 then
          raise detection exception sign
       else
          n=n+1
       end if
    else
    if Rn is the first instruction in a subroutine that needs to be returned then
        push (R_n-1, Sub_stack)
        goto L1
    else
L1:  if Rn is last instruction from the basic block then
```

```
        if R_n is jump addresses frame then
          call Falsebranch_cutoff
          n=n+1
        else
          if R_n is the interrupt return then
            pop(Int_stack, n)
          else
            if R_n is the end instruction in a subroutine that needs to be returned then
              pop(Sub_stack, n)
            else
              n=n+1
            end if
          end if
        end if
      end if
    end for
```

5. Experiment and Results

5.1 The Controller Prototype Implementation

We have implemented a secure multi-core embedded controller prototype on Simics [22] platform, it is a full-system simulator that can runs real hardware platform such as firmware, device drivers, OS, and hypervisors, and allows the processor architecture modifications. We use Freescale QorIQ P4080 processor platform containing eight e500mc cores, only 3 cores are enabled, i.e., core0 and core1 are used in security-critical system as secure core and monitored core (includes perform controller) respectively, and core2 is used in non-security-critical system as processor core. The core0 and core1 run Linux kernel 2.6.34, and the core2 run general-purpose OS (μClinux). The implementation and experimental parameters are shown in Table 1.

Table 1: The experiment platform parameter configuration

Component		Description
Processor	Clock	1000MHz
	L1 Cache	D-Cache 16KB, I-Cache 16KB
	L2 Cache	128KB unified Cache
VM, OS	hypervisor	Xen hypervisor
	Security-critical system	Assign core0 and core1, and run the Linux kernel 2.6.34
	Non-security-critical system	Assign core2, runs μClinux
	Inter-core communication pipe	32B
ISTM	FPGA	Xilinx XC5VLX50T
CFGM	SRAM	64KB, mapped to core0's address space

Based on Xilinx FPGA chip XC5VLX50T, the ISTM is realized and hooked into the core0 (security core) on the security-critical system. Meanwhile, CFG of instruction stream can be monitored by connecting to Program Counter (PC) and Instruction Register (IR).

The hypervisor is configured to completely separate the memory space between the cores and to set the monitored core to a partition that is managed under the security core (core0 can reset core1 by one-way doorbell reset).

Finally, some of the processes such as security monitor, decision module, IO Agent, and security controller are all running in user space. The data sent/received through the inter-core communication pipe is done by the kernel module requested a hypervisor call.

5.2 The Control Application Model

As our physical control system, we use a ball and plate device as the control object, which has two mutually perpendicular rotating axial plates in order to allow a freely rolling ball to balance at a specific position on the plate or to roll along a certain trajectory [23]. The rotation of the plate around the X axis and Y axis is driven by two motors. The sensor obtains the position of the ball on the plate and then feeds back to the controller executed on Simics, and the controller uses a certain control strategy to calculate the actuation command to control rotation angle of the plate to achieve the control of the balance position and trajectory of the ball. In the simulation, PD control is adopted for ball position control and PID control for servo control. The control application binary code size is 53.42KB, which consists of the simple functions to read reference input and control loop. In each execution of the loop, the function responsible for reading the sensor data is called first, and the sensor data is read by interrupt mode.

5.3 Instruction Stream Detection Experiments

Experiment 1: Detection rate and OCFG profile size.

The length x (bit) of hashing is closely related to OCFG profile size and unexpected behavior detection rate. Taking different x values and introducing single-bit error and continuous instruction error in the control application, detection rate and OCFG profile size are tested. The results of 100 experiments are shown in table 2.

Table 2: The test results of detection rate and OCFG profile size

Length x for hashing (bit)	Size of OCFG profile (KB)	Times of single-bit error detected	Times of bias detected for continuous two ins.	Times of bias detected for continuous three ins.
4	31.15	96	98	99
8	35.80	99	100	100
16	41.67	100	100	100
32	54.13	100	100	100

The experimental results in table 2 shows when the bit length x for hashing is more than half of the length of binary instruction, the proposed method can detect single-bit bias, i.e., it can detect the finest granularity unplanned

code, but the detection method based on control flow cannot do that. In this way, the size of granularity and OCFG profile can be comprehensively coordinated to meet different control application requirements by adjusting the bit length for hashing.

Experiment 2: Detection latency.

Detection latency refers to the time from the beginning of the controller running unplanned codes to the system detecting out it. We mainly consider the detection latency from interrupt due to the backtracking instruction number *q* will affect the detection latency in a large extent. We take the hashing bit length $x = 4$ and introduce continuous three instruction errors in the random position of the application binary source code, and let *q* take different values, and carry out 100 tests of the detection latency respectively. Timing starts when the unplanned instruction is entered into the ISTM, and ends when the *abnormal detection signal* is output by security core (i.e., core0). The experimental results are shown in Fig. 4.

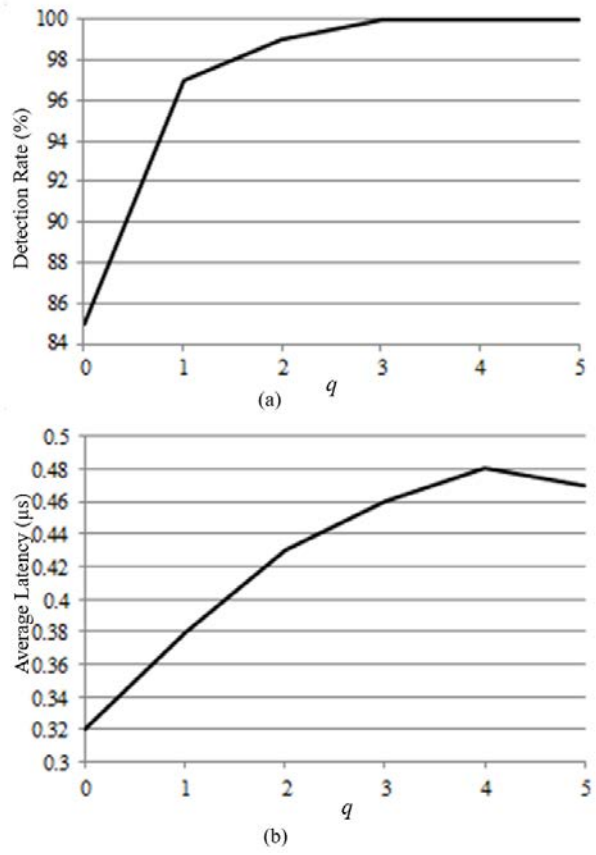

Fig. 4 Detection latency. (a) Detection rate and (b) Average detection latency with q value

The test results in figure 4 shows that increasing the q value can improve the detection rate, but the detection latency also increases. But when q=3, the detection rate reaches the maximum and it is kept. Since almost all interrupt service routines execute several instructions protecting the field first, q=3 had been enough to distinguish them. The proposed method has the ability to detect the unexpected behavior of the control application within 0.5μs, and it can not only meet the requirements of the security-critical control systems, but also outperforms the detection method based on control flow and execution time.

6. Conclusion

This paper proposes a hypervisor-based multi-core embedded controller security architecture and control application runtime unexpected behavior detection method. The architecture integrates the security-critical system and the non-security-critical system, and they are processed by different processor cores. One of the cores is used as the security core and combined with the instruction stream tracking module (ISTM) to dynamically detect unexpected behavior of the control application. This mechanism has certain advantages over independent (off-chip) security detection systems :(1) rapidity, i.e., a CPU core can more closely monitor the execution behavior of software running on other potentially unsecure cores; (2) this is not easy to reverse engineer.

Intrusion detection methods based on instruction stream realizes the detection of the fast, independent, application free of interference. Compared with the current mainstream detection methods based on control flow and execution time, the method proposed by this paper has better detection granularity and lower detection latency.

In the future works, on the one hand, we will further study an effective measure to enhance the security of the hypervisor used in the system, because the hypervisor itself is threatened in the first place, the whole security mechanism may collapse. On the other hand, we should further optimize the detection algorithm to further shorten the detection latency.

Acknowledgment

The authors would like to express their cordial thanks to reviewing experts for his valuable advice.

References

[1] D. Kushner, “The Real Story of Stuxnet”, IEEE Spectrum, Vol.50, No.3, pp. 48–53, 2013.

[2] K. Koscher, A. Czeskis, F. Roesner, et al, “Experimental Security Analysis of A Modern Automobile”, In Proceeding of the IEEE Symposium on Security and Privacy, pp.447–462, 2010.

[3] D. Shepard, J. Bhatti, T. Humphreys, “Drone hack: Spoofing Attack Demonstration on A Civilian Unmanned Aerial Vehicle” , GPS World, Vol.23, No.8, pp.30-33, 2012.

[4] M.Pieter, G. Johannes, C. Ruan, et al, "Hardware-Based Trusted Computing Architectures for Isolation and Attestation", IEEE Transactions on Computers, Vol.67, No.3, pp. 361-374, 2017.
[5] S. Ponomarev, T. Atkison, "Industrial Control System Network Intrusion Detection by Telemetry Analysis", IEEE Transactions on Dependable and Secure Computing, Vol.13, No.2, pp. 252-260, 2016.
[6] F. Wang, Z.Q. Huang, Z.B. Yang, S.L. Kan, G.H. Shen, G.Y. Chen, "A Requirements Traceability Approach for Safety-critical Embedded System", Chinese Journal of Computers, Vol.40, Online Publishing No. 140, 2017.
[7] F. Johannes, U. Sascha, "Closed Loop Controller for Multicore Real-time Systems", Architecture of Computing Systems, Vol.10793, pp.45-56, 2018.
[8] G. Macher, M. Bachinger, M. Stolz, "Embedded Multi-core System for Design of Next Generation Powertrain Control Units", In Proceedings of 2017 13th European Dependable Computing Conference (EDCC), Geneva, Switzerland, pp.4-8, Sept., 2017.
[9] D. Arora, S. Ravi, A. Raghunathan, N.K. Jha, "Secure Embedded Processing through Hardware-assisted Run-time Monitoring", Design, Automation & Test in Europe , Vol. 1, No.1, pp. 178-183, 2005.
[10] X. Wang, Q. Shen, P. Du, R. Zhang, W. Wang, L. Li, B. Xu, H.H. Ji. "Hardware-assisted Monitoring for Code Security in Embedded System", 2015 IEEE 12th International Conference on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conference on Autonomic and Trusted Computing and 2015 IEEE 15th International Conference on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom), pp.1393-1396, 2015.
[11] N. Li, T. Nakajima, "Local-Memory-Based Integrity Checking for Embedded Systems", in Proceedings of the 2013 IEEE 16th Computational Science and Engineering (CSE), pp.742-750, 2013.
[12] R.G. Ragel, J.A. Ambrose, S. Parameswaran. "SecureD: A Secure Dual Core Embedded Processor", Computer Science, 2015.
[13] D.Y. Deng, D. Lo, G. Malysa, S. Schneider, G.E. Suh, "Flexible and Efficient Instruction-Grained Run-Time Monitoring Using on-chip Reconfigurable Fabric", In Proceedings of the IEEE/ACM International Symposium on Microarchitecture, pp.137–148, 2010.
[14] M. Abadi, M. Budiu, U.Erlingsson, J. Ligatti, "Control-flow Integrity Principles, Implementations, and Applications", ACM Transactions on Information and System Security, Vol.13, No.1, pp.1-40, 2010.
[15] A. Rajabzadeh, S.G. Miremadi, "CFCET: A Hardware-based Control Flow Checking Technique in COTS Processors Using Execution Tracing", Microelectronics and Reliability, Vol.46, No. 5, pp.959-972, 2006.
[16] F.A.T Abad, J.V.D. Woude, Y. Lu, et al, "On-Chip Control Flow Integrity Check for Real Time Embedded Systems", 2013 IEEE 1st International Conference on Cyber-physical Systems, Networks, and Applications (CPSNA), pp.26-31, 2013.
[17] D.F. Li, X. Zhang, Q.L. Tong, X.C. Zou, Z.L. Liu, "The Desighn and Implementation of Embedded Security CPU Based on Multi-strategy", Chinese Journal of Electronics, Vol. 25, No.5, pp.801-806, 2016.
[18] D.Sanjeev, W. Zhang, Y. Liu. "A Fine-Grained Control Flow Integrity Approach against Runtime Memory Attacks for Embedded Systems", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol.24, No.11, pp.3193-3207, 2016.
[19] S. Mohan, S. Bak, E. Betti, H. Yun, L. Sha, M. Caccamo, "S3A: Secure System Simplex Architecture for Enhanced Security and Robustness of Cyber-physical Systems", In Proceedings of the ACM International Conference on High Confidence Networked Systems, pp.65-74, 2013.
[20] Z Gu, C. Wang, M. Zhang, Z. Wu, "WCET-Aware Partial Control-Flow Checking for Resource-constrained Real-time Embedded Systems," IEEE Transactions Industrial Electronics, Vol. 61, No.10, pp. 5652-5661, 2014.
[21] A. Crespo, M. Masmano, J. Coronel, S. Peiró, P. Balbastre, J. Simó, "Multicore Partitioned Systems Based on Hypervisor", IFAC Proceedings Volumes, Vol.47, No.3, pp.12293-12298, 2014.
[22] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hllberg, J. Hgberg, F. Larsson, A. Moestedt, B. Werner, "Simics: A Full System Simulation Platform", IEEE Computer, Vol.35, No.2, pp.50–58, 2002.
[23] H.R. Wang, Y.T. Tian, S.Y. Fu, et al, "Nonlinear Control for Output Regulation of Ball and Plate System", In Proceedings of the 27th Chinese Control Conference, Kunming, IEEE, pp.382-387, 2008.

Xiaosheng Wang received his Diploma and M.Sc. Degree from Computer Science and Technology of Jilin University, China, in 1988 and 2003, respectively. He is currently a professor, engaged in teaching and research work in Computer Architecture and Embedded Systems in School of Data Science and Computer at Shandong Women's University. His research interests include computer architecture, embedded systems, and intelligent computing.

Yanchun Yang received the B.E. degrees of Software Engineering from Shandong University, China in 2006. Currently, she is an associate professor of Information Technology School at Shandong Women's University, China. Her research interests include computer networks, information security, etc.

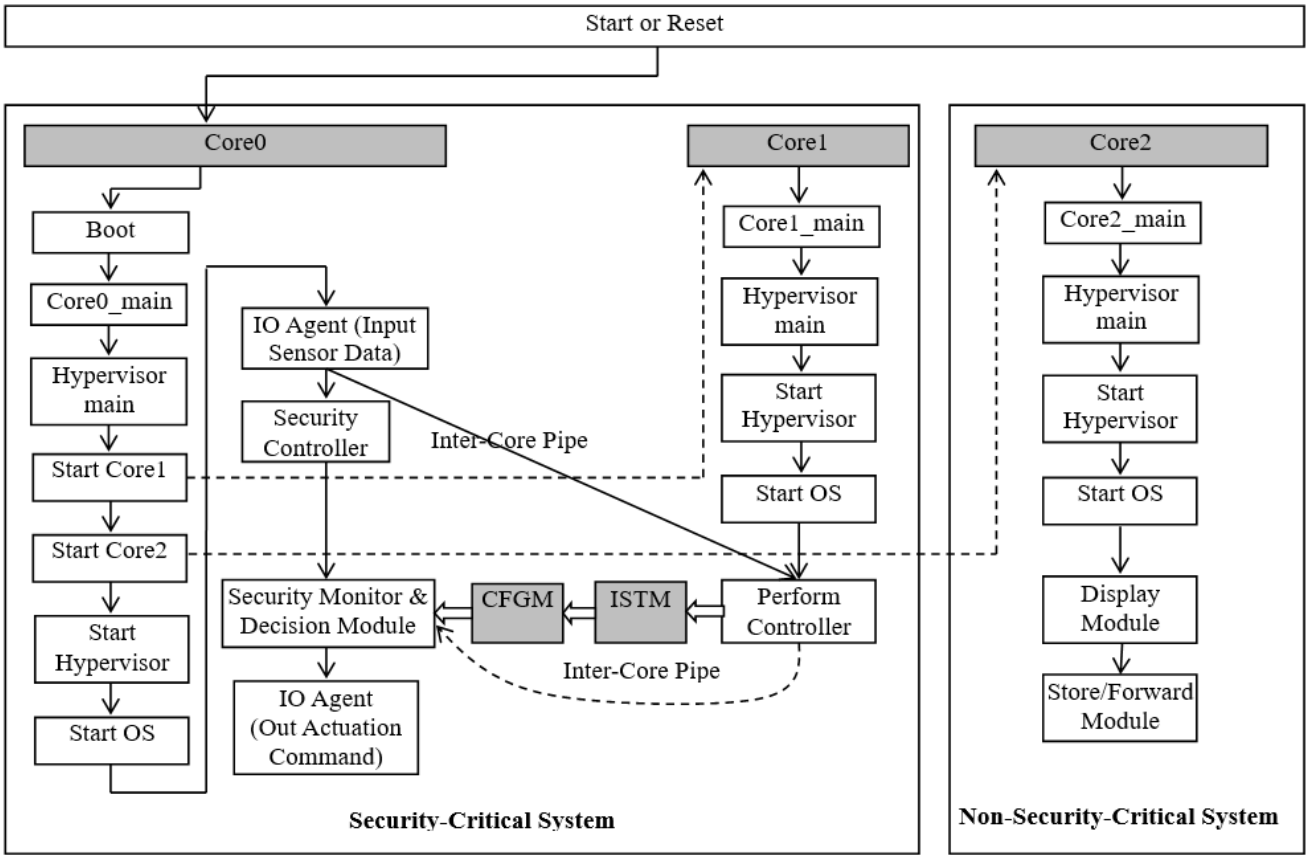

Fig. 3 Start-up and execution flow of the secure controller based on hypervisor