Intrusion Detection using Address Monitoring

K. Rahimunnisa1, S. Suresh Kumar2, Kavya T. S1, and Anoop Suraj A3
1 Asst. Professor, Dept. of ECE, Karunya University, Coimbatore, India
   Email: krahimunnisa@gmail.com
2 Professor, Director, School of Electrical Sciences, Karunya University, Coimbatore, India
   Email: ssk@karunya.edu
3 PG Scholars, Dept. of ECE, Karunya University, Coimbatore, India
   Email: kavyats86@gmail.com, anoopsuraj@ieee.org

Abstract— Security is emerging as an important concern in embedded system design. Security of an embedded system is compromised when software that can be trusted is resulting in unintended behaviour such as leakage of sensitive data and execution of spiteful code. Several counter measures have been proposed to counteract these sorts of intrusions. The general idea behind most of the methods is to define the security policy at system level and check for security breach either statically or at run time. A hardware assisted run time address monitoring system is presented here to enhance embedded system security by detecting and preventing unintended program behaviour. The embedded processor is augmented with the hardware monitor that observes the dynamic execution trace of the processor. It checks whether the execution trace falls within the allowed program behaviour, and flags any deviation from the expected behaviour. This technique can counteract several common software and physical attacks. Furthermore it can ensure secure program execution with minimal overheads.

Index Terms— Security, Static Run-Time Monitoring, Secure Architectures, Embedded Processors.

I. INTRODUCTION

As embedded systems pervade in various aspects of our lives, they are often required to deal with sensitive information or perform critical functions, making security an important concern in embedded system design. Security has been the subject of extensive research in the context of general-purpose computing leading to many advances in cryptographic algorithms and security protocols [1]. Malicious attacks often exploit program bugs to obtain unauthorised access to a system [2]. Recent trends have made it clear that most attacks target weaknesses in a system's implementation. It is now well accepted that a secure system implementation is as critical to a system's overall security. Consequently, recent years have seen an increasing awareness that security needs to be considered at various stages of the embedded system design process. System security can be compromised either through the execution of programs that originate from untrusted or unknown sources or through the corruption of binaries while they are being downloaded or stored on the embedded system [1]. Many recent software attacks exploit the weakness in trusted code that is present in the system. Embedded system possesses a number of unique challenges in terms of resource constraints and susceptibility to physical side channel attacks [3]. Embedded systems are often targeted at specific applications. They also offer opportunities to constrain the scope of design security solution that are optimized to the needs of a specific end system.

A. Paper Overview and Contributions

An efficient and scalable framework is presented here that allows program to specify security policies for their data and validate them during program execution. These security policies are specified by associating program data attributes. Two techniques are considered to address this objective, which includes:

- **Hardware-assisted control flow monitoring**: Many attacks, such as stack-based buffer overflows execute malicious code by exploiting vulnerabilities in a trusted program. Protecting against such attacks is, therefore a critical objective. The solution is based on a simple observation that the execution of malicious code will result in behavior or control flow that is different from the normal program behavior. The proposed solution is implemented by designing a separate hardware monitor that model the characterized program behavior by monitoring the program’s execution on the processor.

- **Hardware-assisted validation of program's data properties**: Some attacks do not modify the control flow of a program, but only modify the data associated with a program in the program's stack or heap. A HW/SW framework can be developed that can enforce various security policies. This framework is effective in preventing various kinds of software attacks, including heap-based and format string attacks. This work attempts to provide a unified hardware platform to enforce security of an embedded system.

II. RELATED WORKS

A wide range of techniques has been proposed to enhance software security in the context of general-purpose computing systems. Static techniques include source code scan tools and code review tools that attempt to strengthen security by eliminating vulnerabilities during the software design phase [4]. Studies have been performed to model vulnerabilities that open doors for security exploits [5]. Although these techniques are useful, the protection they offer is not complete. Various techniques have been proposed to address software
security by ensuring that the code comes from a trusted origin and has not been corrupted during network transmission or storage.

The basic concept of using a hardware unit or co-processor to facilitate secure execution has roots dating back to tamper-resistant cryptoprocessors that were used to store cryptographic keys and execute cryptographic algorithms. Recently, enhanced processor architectures, such as XOM and AEGIS, have been proposed [6]. A dedicated security monitor is implemented to oversee the code injection attack during run time [7]. In addition to the above general techniques, a number of attack-specific mechanisms have been developed [8]. Apart from these mechanisms, researchers have proposed a wide range of run-time monitoring techniques [9].

III. HARDWARE MONITOR ARCHITECTURE

This section provides an overview of the hardware monitor architecture. The basic function of a hardware monitor is to address secure program execution by ensuring that the program does not deviate from its permissible behaviour. So the need for a dedicated hardware monitor is introduced. The hardware monitor can be connected to any embedded processor and it will observe the dynamic execution traces of the processor during run time. Monitor checks whether the trace falls within the permissible behaviour, and flag violations by triggering appropriate response mechanisms. Here the communication is between the processor and the monitor and that the monitor is tamper evident. All the I/O devices, external memory, on-chip caches and the buses connecting these components are considered insecure.

The first step in defining the permissible program behaviour is to capture both the coarse-grained and fine-grained program behaviour in a hierarchical manner. This includes the following steps:

- **The Inter Procedural control flow of a program, as represented by its function call graph, FCG.**
- **The Intra Procedural control flow for each function, represented by a basic-block control-flow graph.**
- **The Integrity of the Instruction stream within each basic block.**

A specific technique is proposed to extract these properties from any given program and automatically synthesize a hardware monitor for it.

**A. Architecture Overview**

For ease of illustration, the embedded processor is depicted as a five-stage pipelined architecture. Fig. 1 shows the block diagram of the proposed hardware assisted monitoring architecture [1].

The inputs to the monitor include the program counter (PC) and instruction register (IR) of the completing instruction, and the pipeline status from the pipeline control unit. Effectively, the monitor is provided with a cycle-by-cycle trace of the executing instructions and their program addresses. The monitor’s outputs include a stall signal and an invalid signal.

Whenever a violation of program behavior is detected by the monitor, it asserts the invalid signal. This will generate a non-maskable interrupt to the processor. This interrupt signal can be used to trigger some mechanism that may terminate the program execution. Whenever the monitor fails to match with the pace of processor, it will assert the stall signal. This will happen in very rare cases.

The monitor has three different modules to handle security-related issues in three different granularities. Modelling of these three granularity levels are explained in the next section.

**B. Modelling Permissible Program Behavior**

Several factors should be considered for selecting the program properties to be monitored:

- Invalid behaviours should be accurately indicated.
- Easily derivable for a wide range of programs that are running on the system.
- Hardware overheads should be minimum during run time.

To ensure these properties three parameters are chosen. They are:

**Inter Procedural Control Flow**

The correctness of a program’s inter-procedural control flow is verified at the highest level of granularity. Inter-procedural control flow can be represented using a function call graph (FCG). For ease of hardware implementation, the N-state FCG is converted into N+1 states FSM. A transition between two states in the FSM represents a valid control flow between the corresponding functions. The invalid state will come into action when there is any invalid call or return.

**Intra Procedural Control Flow**

Intra-procedural control flow will track the control flow within each function in the program. This is the logical succession to the previous stage. This can be represented with the help of a control flow graph (CFG).

**Instruction Stream Integrity**

Lowest level of granularity will check the security attacks against program code segment. So to detect such attacks, the integrity of the instruction stream is checked.
with the help of a cryptographic hash function. Hash value of each basic block in the program is computed and stored in the monitor before the program is loaded for execution.

C. Error Reporting

If the monitor detects any violation in the program execution, it is reported to the processor. Monitor will detect the location where the violation happened and provides feedback to the processor about the location. All the three monitor sub blocks have its own flag registers to register any deviation or any violation. So the processor will identify the particular sub block which has flagged the error.

IV. EXPERIMENTAL RESULTS

The model of hardware monitoring system is modelled in three different stages. The first stage is inter-procedural control flow checker. It is modelled to check whether there is any invalid flow in the control of the functions in a program due to any vulnerabilities or invalid function call or return. Before executing each function the starting address of the function is verified with the help of FSM. If there is any variation in the actual address and the predefined address, the program will terminate by sending an invalid signal to the processor.

Sample coding has been written in C language. The main function has two sub functions. Every time when there is a function call or return, it checks for the address of the called or returning function and flags any variation in the address, which then terminates the program. The output window for the above concept is shown in Fig. 2.

This forms the highest level of integrity in execution of a program. Content Addressable Memory (CAM) is also considered the hardware implementation of Inter Procedural Control Flow block. Fine level of integrity such as Inter-procedural control flow and instruction integrity checking also has to be modelled. Any Cryptographic hash function can be used for the implementation of Instruction Integrity Checker block.

CONCLUSION

In this paper a scalable framework for performing safer execution of a program is presented with minimal performance loss. The technique exploits a combination of software and hardware modification to achieve this objective. The predefined control and data properties are monitored and a wide range of security attacks can be prevented. The highest level of integrity in program execution is presented here. Further modification is required to extract the fine grain properties. Ensuring secure software execution in the presence of potential vulnerabilities in program is an important but challenging task. This proposed hardware assisted run time validation of data properties is a promising step in this direction.

ACKNOWLEDGMENT

The authors would like to thank Karunya University management for providing the facilities to conduct the research work.

REFERENCES


