Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

SOFIA: Software and control flow integrity architecture

Ruan de Clercq [a,*], *Johannes Götzfried* [b], *David Übler* [b], *Pieter Maene* [a], *Ingrid Verbauwhede* [a]

[a] *ESAT/COSIC and imec, KU Leuven, Leuven, Belgium*
[b] *Department of Computer Science, FAU Erlangen-Nuremberg, Erlangen, Germany*

A R T I C L E   I N F O

Article history:
Received 2 November 2016
Received in revised form 30 January 2017
Accepted 27 March 2017
Available online 3 April 2017

Keywords:
Control flow integrity
Hardware
Security
Computer architecture
Instruction set randomization

A B S T R A C T

Software components are frequently used in cyber-physical systems (CPSes) to control a physical mechanism, such as a valve or brakes on a car. These systems are extremely sensitive to software vulnerabilities, as their exploitation could lead to injury, damage to equipment, or environmental catastrophe. This paper proposes a hardware-based security architecture called SOFIA, which protects software running on microprocessors used in CPSes. SOFIA provides mechanisms to protect software integrity and control flow integrity. This allows the processor to defend against a large number of attacks, including code injection, code reuse, and fault-based attacks on the program counter. In addition, the architecture also defends against software copyright infringement and reverse engineering. All protection mechanisms are enforced in hardware using cryptographic techniques. We are the first to propose a mechanism to enforce control flow integrity at the finest possible granularity using cryptographic techniques. A SOFIA core has been created by implementing the proposed architectural features on a LEON3 microprocessor. The SOFIA core requires that its software conforms to a strict format. To this end, we additionally designed and implemented a software toolchain to compile source code that adheres to the formatting rules. Several benchmarks were compiled with the SOFIA toolchain, and were executed on a SOFIA core running on an FPGA, showing an average total execution time overhead of 106% compared to an unmodified LEON3 core. Our hardware evaluation shows a clock speed reduction of 23.2%.

1. Introduction

Cyber-physical systems (CPSes) enable the physical world to integrate with control systems such as embedded devices or the Internet of Things (IOT). Software algorithms run on embedded devices which use sensors to measure physical processes and actuators to control physical components, such as a valve or the brakes on a car. The software is responsible for monitoring and controlling these physical components to ensure that they are operating correctly. Examples of CPSes include industrial control systems, process control, autonomous automobile systems, and medical implants.

* *Corresponding author.*
E-mail address: ruan.declercq@esat.kuleuven.be (R. de Clercq).

http://dx.doi.org/10.1016/j.cose.2017.03.013
0167-4048/© 2017 Published by Elsevier Ltd.

The correct functioning of CPSes is crucial, as the failure can lead to injury, damage to equipment, or environmental catastrophe. To ensure their correct operation, we need to ensure that the software that runs on the computational components are not compromised. Exploits that target vulnerabilities in the underlying software are increasingly used to obtain full system access. The attack surface is also increasing, as processors become more interconnected through ad-hoc networks and through the public Internet. Despite a significant amount of research to address the underlying problems of software vulnerabilities, there are still a vast number of attacks that threaten the security of software.

This work aims to protect the software running on low-end microprocessors used in CPSes. We specifically target software applications that do not require an operating system. Low-end microprocessors often lack basic architectural support for security, and are frequently deployed in the field, where it is easy to extract and exploit their software. As we rely on software to control and monitor physical processes, we need to know that the software behaves in a predictable manner, and an adversary should not be able to alter their software or tamper with their operation. Ideally, even if an attacker obtains the code running on a device, he should not be able to understand it and know, e.g., which version of the software is being deployed. Not knowing that will make it harder to exploit potential weaknesses in the software, such as overflows or incomplete input validation.

A lot of research has focused on code injection (One, 1996; Pincus and Baker, 2004) and control flow integrity (CFI) (Abadi et al., 2005a, 2005b), but current solutions (Bletsch et al., 2011; Davi et al., 2012; Pappas et al., 2013; Tice et al., 2014; Xia et al., 2012; Zhang et al., 2013; Zhang and Sekar, 2013) have been demonstrated to be breakable (Carlini and Wagner, 2014; Davi et al., 2014; Goktas et al., 2014; Schuster et al., 2015), or they require significant hardware (Arora et al., 2005; Danger et al., 2014; Davi et al., 2014, 2015; Kayaalp et al., 2012, 2014; Mao and Wolf, 2010) or OS support found only on high end general-purpose processors. CFI forces a program to follow a control flow path along a Control Flow Graph (CFG) that can be predetermined or calculated at run-time.

Our contribution. In this paper we propose SOFIA, an architecture that defends against attacks based on code injection, code reuse, software tampering, and fault attacks on control flow. The architecture is deeply integrated in a processor's pipeline stages, and relies on cryptographic methods to protect the running software. In particular, SOFIA enforces control flow integrity, software integrity, and code secrecy. In detail, our contributions are as follows:

- We offer reliable tampered code protection, i.e., tampered instructions or instructions that occur during illegal control flow will not be executed. Furthermore, metadata stored in memory is protected from tampering.
- We protect against powerful adversaries in control of all memory as well as against fault attacks on control flow.
- Our policy is enforced entirely in hardware and does not enforce protection by means of any software stored in memory.

In addition we provide a thorough, real-world evaluation of SOFIA. To this end, we designed and implemented a compiler-based toolchain to transform software to be executed on a SOFIA core. We further implemented a SOFIA core in VHDL as an extension to the LEON3 soft processor. In addition, the performance overhead was evaluated by executing a number of different benchmarks on the SOFIA core programmed on an FPGA.

The remainder of this paper is structured as follows. First, the problem statement is provided including the threat model and system goals. Next, the proposed architecture is described. Afterwards, we present a description of the hardware implementation, followed by implementation details of the toolchain. We then describe the current state-of-the art in control flow integrity and give a security and performance evaluation of our solution. Finally, we conclude our work with an outlook over future research directions.

2. Problem statement

In this section we discuss the system model, composed of a threat model and a set of system requirements.

2.1. *Threat model*

This work considers an adversary capable of the following:

- Full control of program and data memory. The attacker can replace parts or all of the software, including the entire stack, program memory, data memory, ROM and RAM.
- Full control of external I/O pins of the processor. He is able to probe the external memory bus and disconnect external components. However, the internal components of the processor cannot be altered or probed.
- Capable of performing non-invasive fault attacks that target the program flow, such as glitching the clock. However, other types of fault attacks that do not target the program flow, such as glitching the ALU or bus of the processor, are not considered part of the attacker model.
- Side-channel attacks are not considered.
- With respect to cryptographic capabilities, we follow the Dolev-Yao (Dolev and Yao, 1983) model, where an attacker is capable of performing protocol-level attacks, but cannot break cryptographic primitives.

2.2. *System goals*

In this work, a hardware-based security architecture performs run-time verification of the integrity and execution of software. To this end, the requirements of the system are as follows.

Software integrity: The attacker should not be able to execute tampered software on a SOFIA core. We consider the software to consist of instructions and read-only data.

Control flow integrity: The attacker should not be able to change the control flow of running software along an invalid path without this being detected. This includes software-based attacks based on code-reuse such as jump-oriented-