Microprocessors and Microsystems 45 (2016) 115–128

Contents lists available at ScienceDirect

Microprocessors and Microsystems

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

Design and implementation of instruction indirection for embedded software obfuscation

Naoki Fujieda*, Tasuku Tanaka, Shuichi Ichikawa

Department of Electrical and Electronic Information Engineering, Toyohashi University of Technology, 1-1 Hibarigaoka Tempakucho, Toyohashi, Aichi, 441-8580, Japan

ARTICLE INFO

Article history:
Received 24 July 2015
Revised 7 January 2016
Accepted 14 April 2016
Available online 21 April 2016

Keywords:
Embedded system
Secure processor
Software protection
Instruction fetch
Instruction register file

ABSTRACT

Instruction Register File (IRF) was originally proposed to reduce the power consumption of a microprocessor by providing the indirect access to frequently executed instructions. The IRF is also an attractive and cost-effective unit to protect embedded software from analysis, plagiarism, and falsification. For this purpose, the correspondences between IRF entries and their original instructions must be concealed. This means the instructions in the IRF should be carefully selected both to have more instructions be executed through the IRF and to flatten the distribution of the indices of the IRF.

This paper presents two heuristic algorithms, precision-oriented and time-oriented, to find sub-optimal assignments to the IRF. According to the evaluation results, the precision-oriented algorithm obtained the same as or very close to the optimal assignment of an IRF with 48 or less entries. The time-oriented algorithm found a sub-optimal assignment of a 1024-entry IRF in 16 ms, whose precision was 0.5% inferior to the precision-oriented solution at a maximum. The hardware cost of a 1024-entry IRF on an FPGA was modest: two 18 kib block RAM elements and 0.8% increase of the logic elements.

1. Introduction

To protect intellectual property, the defense of software against analysis, plagiarism, and falsification has become an important issue. For embedded systems, it should be attained with a minimal overhead because of their strict limitations on resource or performance.

Reverse engineering consists of three steps: acquisition of machine code, disassembly and decompilation. Disassembly is a conversion of machine language into assembly language, while decompilation is a translation of an assembly into a human-readable code of high level language. Tamper resistance of software is obtained by obstructing at least either of them. Specifically, there are the following three major ways to protect software:

- encryption of the machine codes to make them meaningless without a hidden key [1,2],
- obstruction of disassembly by confusing disassemblers or preventing them from accessing necessary information about the instructions [3–10], and
- obstruction of decompilation by scrambling the structure of the program [11,12]

Instruction set randomization (ISR) [3–8] is an approach to protect software by obstructing disassembly, where each processor (or group of processors) has a different instruction set. Software is protected from analysis or plagiarism by different or additional instruction coding system that is hidden from attackers. Moreover, diversified instruction sets are naturally resistant to falsification because a malicious instruction sequence for one processor (or one group of processors) does not operate correctly on the others.

The use of an Instruction Register File (IRF) [13,14] is one of the attractive candidates of instruction set randomization. The IRF is a small memory that stores the most common expressions of instructions specified by the compiler. It is referred by an index in fetched instructions. Though it initially aimed at reducing power consumption by packing multiple instructions into a single one [13], it also has resistance to tampering [15]. It is based on the fact that a reference to the IRF is considered as another expression of the instruction, which is incomprehensible as long as the contents of the IRF are hidden. However, the contents may be guessed from the occurrence frequency of indices. Some routines may not be obfuscated enough by the lack of references to the IRF. Such risks was not evaluated in [15], although the code reduction and the execution time with the IRF were measured as its *side effects*.

* Corresponding author.
E-mail addresses: fujieda@ee.tut.ac.jp (N. Fujieda), ichikawa@ieee.org (S. Ichikawa).

http://dx.doi.org/10.1016/j.micpro.2016.04.005
0141-9331/

This paper examines the effectiveness of the IRF against tampering, particularly reverse engineering. It is suitable for embedded systems, such as manufacturing machinery, because of its small overhead on resource and performance. Without protection, valuable know-how and trade secrets that their software contains might be easily uncovered and stolen. Moreover, once their software are analyzed, injection of malicious code by abusing buggy program or unauthorized modification of software might be also possible. When the IRF-based ISR is applied to, most of the program code is expressed by indices of the IRF, which are not understandable without the IRF contents. It can also prevent malfunctions of systems or serious accidents due to falsified software.

The main topic of this paper is proper selection of the contents of the IRF, including the quantification of the efficiency of instruction hiding, proposal of selection algorithms, and their evaluation. This paper also includes an implementation of a large IRF on an FPGA to show that its hardware cost is comparable to other ISR approaches.

The rest of this paper is organized as follows: Section 2 provides overviews of the related studies and prerequisites for the IRF on utilizing it for tamper resistance. In Section 3, we introduce a scale of tamper resistance for the IRF contents considering frequency analysis and show its characteristics. In Section 4, we propose a branch-and-bound algorithm to find the optimal assignment and evaluate it. Section 5 describes heuristic algorithms to find a sub-optimal assignment and their evaluation with two instruction sets. Section 6 presents and evaluates an FPGA implementation of the IRF. In Section 7, we make some discussions about possible attacks to our method. Finally, we conclude the paper in Section 8.

2. Background

2.1. Protection against reverse engineering and instruction set randomization

Encryption of instruction memory is one of the most typical anti-tamper approaches for processors, which was used in the Execute Only Memory (XOM) [1] and the AEGIS architecture [2]. Most of the methods adopt modern ciphers such as AES to decrypt instruction memory that was encrypted at compile time. Even data memory is often encrypted and decrypted on the fly. This approach is useful for applications where security is the most important concern. Nevertheless, it significantly increases the memory access latency and the hardware resources, and thus it is unsuitable for cost-sensitive embedded systems.

Protection of software is also achieved by obstructing at least either of disassembly or decompilation. As an obstruction of disassemblers, Linn and Debray [9] proposed a method to disorder disassemblers to generate erroneous assembly codes in IA-32 by inserting junk bytes that obscure the partitions of instructions. Monden *et al.* [10] proposed a finite state machine-based approach where the interpretation of an opcode varied with the value of the internal state machine. Obfuscation methods against decompilers were structured by Collberg *et al.* [11]. In particular, control flow obfuscation disturbs the reconstruction of the control flow of the original program. It is based on opaque predicates, or conditional codes whose outcomes are actually constant but not easily deduced, followed by unreachable bogus codes [11,12].

Instruction set randomization (ISR) may be categorized into obstruction of disassembly, and also be considered as lightweight instruction memory encryption. Compared to robust but costly modern ciphers, most of them are based on simple substitution ciphers for lower overhead. Both hardware [5,7,8] and software [3,4,6] implementations have been studied. Some methods utilize the characteristics of the target instruction set [5]. Some other methods rely on stream ciphers [4,8], whose characteristics are closer to those of memory encryption: higher safety but higher cost.

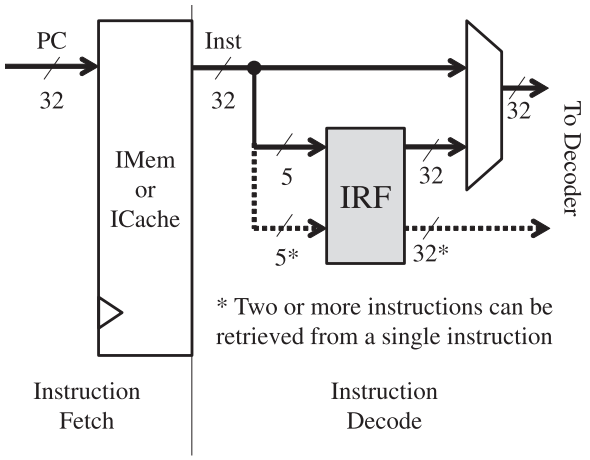

Fig. 1. The Instruction Register File [13] retrieves frequently executed instructions from indices in specialized instructions.

An important measure of ISR is the hardness to guess the original instructions from the *randomized* ones with frequency analysis [16]. Some kinds of instructions might be easily guessed from statistical properties of the obfuscated binary (e.g. the frequencies of opcode values). The analysis might be even easier if heuristics are applied. For example, the reserved fields of specific instructions are always set to zero. To prevent frequency analysis, it is important for ISR to decrease statistical information of the original instructions.

2.2. Instruction register file

Instruction Register File (IRF) [13] is a table of frequently used instructions. Fig. 1 illustrates the IRF. It assumes a 32-bit RISC ISA such as MIPS and ARM. It is placed just before instruction decoder and accessed by indices. If a specialized instruction is fetched, the corresponding instruction(s) to the index (indices) will be read from the IRF and sent to the decoder. In this paper, we refer to instructions that reside in the IRF as IRF instructions.

The IRF has originally been proposed to compress instruction sequences. Since the bit length of an index of the IRF is much shorter than that of an original instruction, multiple IRF instructions can be extracted from a specialized one. In the original proposal [13], both normal and specialized instructions were 32 bits long. Seven bits of specialized instructions were used for identification and the remaining 25 bits were for indices. Since the number of the IRF was set to 32, each index required $\log_2 32 = 5$ bits. Therefore, a specialized instruction contained up to five ($= 25/5$) continuous instructions listed in the IRF.

2.3. Using IRF for anti-tampering

In addition to the reduction of code length, the IRF can provide randomization of a subset of the instructions [15]. It is possible to protect software from analysis by arbitrarily shuffling the mapping from indices to IRF instructions. It provides a protection from plagiarism if the mapping and the corresponding instruction sequence are diversified for each system. It makes the system robust over falsification by prohibiting IRF instructions from being executed directly from instruction memory. In comparison with other ISR methods, the IRF has an advantage of hiding information about operands.