Available online at www.sciencedirect.com

ScienceDirect

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

Computers & Security

Covert Computation — Hiding code in code through compile-time obfuscation☆

Sebastian Schrittwieser [a,*], *Stefan Katzenbeisser* [b], *Peter Kieseberg* [c], *Markus Huber* [c], *Manuel Leithner* [c], *Martin Mulazzani* [c], *Edgar Weippl* [c]

[a] *Vienna University of Technology, Favoritenstraße 9–11/188, 1040 Vienna, Austria*
[b] *Security Engineering Group, Darmstadt University of Technology, Hochschulstraße 10, 64289 Darmstadt, Germany*
[c] *SBA Research, Favoritenstraße 16, 1040 Vienna, Austria*

ARTICLE INFO

Article history:
Received 29 July 2013
Received in revised form
25 September 2013
Accepted 30 December 2013

Keywords:
Code obfuscation
Side effect
Code steganography
Semantic-aware malware detection
Compile-time obfuscation

ABSTRACT

Recently, the concept of semantic-aware malware detection has been proposed in the literature. Instead of relying on a syntactic analysis (i.e., comparison of a program to pre-generated signatures of malware samples), semantic-aware malware detection tries to model the effects a malware sample has on the machine. Thus, it does not depend on a specific syntactic implementation. For this purpose a model of the underlying machine is used. While it is possible to construct more and more precise models of hardware architectures, we show that there are ways to implement hidden functionality based on side effects in the microprocessor that are difficult to cover with a model. In this paper we give a comprehensive analysis of side effects in the x86 architecture and describe an implementation concept based on the idea of compile-time obfuscation, where obfuscating transformations are applied to the code at compile time. Finally, we provide an evaluation based on a prototype implementation to show the practicability of our approach and estimate complexity and space overhead using actual malware samples.

1. Introduction

During the last decade, malware detection has become a multi-billion dollar business and an important area in academic research alike. Static analysis, still the predominant technique for client based malware detection (commonly known as virus scanners), has not changed much during the last years. Current virus scanners almost entirely rely on signature based detection mechanisms (Christodorescu and Jha, 2004; Griffin et al., 2009). Malware, on the other side, has evolved significantly throughout the years and often uses sophisticated code obfuscation techniques in order to make detection more difficult. Encryption, polymorphism as well as metamorphism are commonly deployed to defeat signature based detection mechanisms by hiding the malicious functionality in data sections of the binary that look different for each instance of the malware.

To increase detection rates of obfuscated malware, new paradigms of malware analysis have been proposed. Semantic-aware malware detection, which was first introduced by Christodorescu et al. (2005), aims at solving some of

☆ This paper is an extended version of the conference paper (Schrittwieser et al., 2013).

* *Corresponding author*. Tel.: +43 1 5053688.
E-mail addresses: sebastian.schrittwieser@tuwien.ac.at (S. Schrittwieser), skatzenbeisser@acm.org (S. Katzenbeisser), pkieseberg@sba-research.org (P. Kieseberg), mhuber@sba-research.org (M. Huber), mleithner@sba-research.org (M. Leithner), mmulazzani@sba-research.org (M. Mulazzani), eweippl@sba-research.org (E. Weippl).

0167-4048/$ – see front matter © 2014 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cose.2013.12.006

the limitations of signature-based detection strategies by using so-called templates which define malicious behavior independently of its actual implementation. This approach makes the malware detection system more resistant against some types of obfuscating transformations such as *garbage insertion* (Collberg et al., 1997) and *equivalent instruction replacement* (De Sutter et al., 2009). However, a major limitation of this approach is its dependency on an accurate model of the underlying hardware (i.e., the microprocessor). In order to be able to evaluate the maliciousness of a sequence of processor instructions this model has to be detailed enough to map all effects on the hardware's state.

In this paper we show that exactly this fundamental prerequisite for semantic-aware malware detection is difficult to achieve. We introduce a concept called Covert Computation that is based on the idea of implementing program functionality in side effects of the microprocessor that are not covered by a simple machine model. In contrast to packer-based obfuscation which hides code in data sections that cannot be evaluated in static analysis scenarios, we go one important step further in this paper by hiding (malicious) code in real code. The main advantage of this approach over previous ones is that hidden functionality is not identifiable for syntactic malware detectors and extremely difficult to detect with semantic analysis techniques. In detail, the main contributions of this paper are:

- We introduce a novel approach for code obfuscation called Covert Computation, based upon side effects in today's microprocessor architectures. It hides (potentially malicious) code in legitimate code.
- We provide a comprehensive collection of side effects for Intel's x86 architecture and show how they can be used to hide (potentially) malicious functionality in executables.
- We describe a proof-of-concept implementation of our obfuscation technique that is performed at compile-time.
- We finally evaluate the security of our obfuscation approach against semantic-aware malware detection, measure the performance based on our prototype and provide a theoretical discussion on the effects of this obfuscation technique on real-life malware samples.

The remainder of the paper is organized as follows: In Section 2 we discuss related work in the area of malware obfuscation as well as malware detection. In Section 3 we introduce side effects of Intel's x86 instruction set and describe how they can be used to hide malicious functionality inside harmless looking code. In Section 4 we propose our concept of compile-time obfuscation and present a prototype implementation based on the LLVM compiler infrastructure. An extensive evaluation of our concept is described in Section 6. Finally, we summarize the main contributions of our paper and draw conclusions in Section 7.

2. Related work

Today's malware obfuscation approaches often follow the simple concept of hiding malicious code by packing or encrypting it as data that cannot be interpreted by the machine (Nachenberg, 1997). At runtime, an unpacking routine is used to transform the data block back into machine-interpretable code. Polymorphism (Song et al., 2007) and metamorphism (O'Kane et al., 2011) can be seen as improvements to the packer concept aiming at making automated malware detection more difficult. Another variant of packing was introduced by Wu et al. (2010). Their approach – called *mimimorphism* – encodes the program's code as harmless looking code, which is not detectable with previous concepts (such as entropy analysis) as the packed code appears to be code itself. Resulting binaries follow an unobtrusive statistical distribution of instructions and thus are able to trick malware detectors that work on the syntactical layer. However, this concept would not withstand a semantic code analysis. Even though the packed code looks like real code, it is just a sequence of functionally unrelated instructions without any semantic meaning.

On the other side of the arms race the detection and analysis of packed malware has been studied for many years. Many approaches are based on static code analysis. Encrypted code is identifiable based on entropy analysis as shown by Lyda and Hamrock (2007). Bruschi et al. (2006) described an approach for detecting self-mutating malware by matching the inter-procedural control flow graph of software against malware samples. The authors argue that despite its self-mutating nature, the control flow graph of this type of malware is still characteristic enough for reliable detection.

In contrast to the detection of packer-based obfuscation, the analysis of the actual semantics of code was proposed in literature using different approaches. The idea of using model checking for detecting malicious code was proposed by Kinder et al. (2005). Christodorescu et al. (2005) described the concept of semantic-aware malware detection, aiming at matching code with predefined templates specifying malicious behavior; matching of malicious code still works even if the actual implementation of the malicious behavior slightly differs from the reference implementation in the template. Dalla Preda et al. further formalized the approach of semantic-aware malware detection in 2007 (Dalla Preda et al., 2007) and 2008 (Dalla Preda et al., 2008). An important aspect of this type of malware detection is that is heavily depends on the quality of the model of the underlying hardware as the effects of a sequence of instructions has to be matched against the effects of a malware template. The first theoretical discussion on the idea of forcing a detection system into incompleteness was presented by Giacobazzi (2008). However, no practical approach of this idea was given in the paper. Moser et al. (2007) discussed the question whether static analysis alone allows reliable malware detection. The authors argue that semantic-aware detection systems are only effective against malware that is not protected against this particular analysis method and prove their claim with a new binary obfuscation schema that successfully prevents malware identification even by semantic-aware detectors. The paper concludes that simple obfuscation techniques can reliably hide the purpose of a program's code, and thus clearly shows the limits of static analysis.

Another approach against the threat of malware is to dynamically analyze the behavior of software in order to identify malicious routines (Willems et al., 2007). Sharif et al.