Hunting Obfuscated Malwares by Abstract Interpretation

MILA DALLA PREDA

Dipartimento di Informatica, University of Verona, Italy
dallapre@sci.univr.it

The Problem. A *malware* is a program with a malicious behaviour, that is designed to replicate with no user consent and to damage software and/or data on infected machines. Malwares are generally classified according to their goals and propagation methods into viruses, worms, backdoors, Trojans, etc. A *malware detector* is a system that attempts to verify whether a program presents a malicious behaviour or not. The design of efficient malware detectors is crucial for preventing serious damages caused by malware infection. Current malware detectors (e.g. commercial virus scanners) in general rely on *static signature matching* and, more recently, on *dynamic analyses* [9]. The dynamic approach executes the potentially infected program in a controlled environment (sandbox) thus performing a run-time verification of malicious behaviours. However, smart malwares may foil a dynamic analysis by modifying their behaviour when executed in a sandbox. Static signature matching classifies a program P as infected by a malware M when an instruction sequence of P matches the characteristic instruction sequence of M. Malware writers frequently use *obfuscation* to prevent signature matching detection. Code obfuscation [3] consists in syntactically transforming a program while maintaining its functional behaviour. Recent results [1] show that static signature matching can be defeated using simple obfuscating techniques, including code transposition, substitution of equivalent instruction sequences, opaque predicate insertion and variable renaming. Thus, the signature matching methodology is not resilient to slight modifications of malwares and needs a frequently updated database of malware signatures (one for each version of the malware). The reason way obfuscation can easily foil signature matching lies in the syntactic nature of this approach that ignores program functionality. Program behaviours are precisely described by formal semantics, so that facing the malware detection problem from a semantic point of view could lead to a more resilient detection system. Preliminary work [2] on semantics-aware malware detectors confirms the potential benefits of a semantic approach. Our goal is to provide a *semantic characterization* of malware infection to be used as a basis for designing malware detectors that are resilient to most commonly used obfuscating techniques.

Results. A first necessary step towards designing a semantic malware detector consists in providing a rigorous theoretical framework where code obfuscation can be viewed as a semantic transformation. This is a challenging problem *per se*, since the major drawback of most code obfuscation techniques lies in the lack of an accurate theoretical background; that limits formal reasoning on obfuscation. *Abstract interpretation* [4] turns out to be a suitable framework for formalizing a general semantics-based theory for code obfuscation. In [6] we introduce a novel definition of code obfuscation, based on program semantics, that generalizes the standard notion of obfuscation and shows that every program transformation may potentially act as code obfuscation. We focus on code obfuscation by opaque predicate insertions and in this case our semantic approach is able to drive a systematic design of suitable de-obfuscating techniques [5]. Moreover, experimental results practically show a significant performance improvement of such de-obfuscating techniques [7]. Our study on semantic code obfuscation shows that an obfuscation $\mathcal{O}$ preserves program semantics up to some abstraction $\alpha_{\mathcal{O}}$. Thus the semantics of a malware M and of its obfuscation $\mathcal{O}(M)$ are $\alpha_{\mathcal{O}}$-equivalent, namely $\alpha_{\mathcal{O}}(S[\![M]\!]) = \alpha_{\mathcal{O}}(S[\![\mathcal{O}(M)]\!])$, where $S[\![\cdot]\!]$ denotes program trace semantics. Given a suitable abstraction $\alpha_{\mathcal{O}}$, malware infection of a program P can be semantically characterized by the following relation: $\alpha_{\mathcal{O}}(S[\![M]\!]) \subseteq \alpha_{\mathcal{O}}(S[\![P]\!])$. In fact, P is infected by a malware M when P presents the malicious behaviour M, namely when the (abstract) semantics of P contains the (abstract) semantics of M. Dealing with an abstraction of the semantics allows us to handle possibly obfuscated malwares. The idea is that $\alpha_{\mathcal{O}}$ abstracts from details that are not relevant for identifying the malware, that is, $\alpha_{\mathcal{O}}$ looses those aspects of the malware behaviour that may change in different obfuscated versions of the malware (e.g. names of variables). On the other hand, the abstraction $\alpha_{\mathcal{O}}$ has to be precise enough to capture the essence of the malware behaviour. Clearly, the key point for designing an efficient semantic malware detector consists in determining a suitable abstraction $\alpha_{\mathcal{O}}$. In fact, $\alpha_{\mathcal{O}}$ has to be abstract enough to detect obfuscated versions of the same malware, in order to avoid false negatives, and precise enough to avoid false positives. The abstraction $\alpha_{\mathcal{O}}$ plays here the role of a *semantic normal form* with respect to obfuscation $\mathcal{O}$. In fact, given an obfuscating transformation $\mathcal{O}$, our semantic methodology for malware detection first reduces the semantics of the malware M and of the program P to the normal form with respect to $\mathcal{O}$, i.e. performs the abstraction $\alpha_{\mathcal{O}}$, and then checks if the normal forms match. In general, a malware composes different obfuscating transformations to prevent detection. It is therefore important to investigate how semantic detection works with more than one obfuscation method. We observe that the precision degree of the semantic malware detector is preserved by composition. Let us consider two obfuscations $\mathcal{O}_1$ and $\mathcal{O}_2$ and assume that the corresponding semantic abstractions $\alpha_{\mathcal{O}_1}$ and $\alpha_{\mathcal{O}_2}$ avoid both false positives and negatives. It turns out that, if the obfuscations $\mathcal{O}_1$ and $\mathcal{O}_2$ are independent, namely if they commute respectively with the abstractions $\alpha_{\mathcal{O}_2}$ and $\alpha_{\mathcal{O}_1}$, the composed abstraction $\alpha_{\mathcal{O}_1} \circ \alpha_{\mathcal{O}_2}$ prevents both false positives and negatives when the malware obfuscates itself through the composition $\mathcal{O}_2 \circ \mathcal{O}_1$. This is an important observation because it allows a modular approach to deal with real-life complex obfuscating transformations, that can be viewed as composition of elementary obfuscations. We are able to provide a semantic abstraction that avoids both false positives and negatives for several commonly used obfuscating transformations, including variable renaming, substitution of equivalent sequences of instructions, code transposition, opaque predicate and semantic nop insertion. Moreover, every pair of these transformations satisfies the independency condition, that turns out to be non-restrictive in real-life obfuscators.

Open Issues. Given an obfuscating transformation $\mathcal{O}$, we assumed that the abstraction $\alpha_{\mathcal{O}}$ is provided by the malware detector designer. We are currently investigating how to design a systematic (ideally automatic) methodology for deriving a suitable abstraction

$\alpha_{\mathcal{O}}$ that leads to an efficient semantic malware detector (i.e., no false negatives and positives). We observed that if the abstraction $\alpha_{\mathcal{O}}$ is preserved by the obfuscation $\mathcal{O}$, namely $\alpha_{\mathcal{O}}(S[\![P]\!]) = \alpha_{\mathcal{O}}(S[\![\mathcal{O}(P)]\!])$, then there are no false negatives. This is an interesting result because in [5] we provide a systematic methodology that, given an obfuscation $\mathcal{O}$, derives the most concrete property preserved by $\mathcal{O}$. However, preservation is not enough to eliminate false positives. Hence, an interesting research task consists in characterizing the set of semantic abstractions that prevents false positives. One further step would be to investigate whether and how model checking techniques can be applied to detect malware. Some works along this line already exist [8]. Observe that the abstraction $\alpha_{\mathcal{O}}$, that characterizes the semantic normal form with respect to the obfuscation $\mathcal{O}$, actually defines a set of program traces that are equivalent up to $\mathcal{O}$. In model checking, sets of program traces are represented by formulae of some linear/branching temporal logic. Hence, we aim at defining a temporal logic whose formulae are able to express normal forms of obfuscations together with operators for composing them. This would allow to use standard model checking algorithms to detect malwares in programs. This could be a possible direction to follow in order to develop a practical tool for malware detection based on our semantic model. We expect this semantics-based tool to be significantly more precise than existing virus scanners.

Acknowledgements. This research has been partly conducted during a visit to the Department of Computer Science, The University of Arizona, joining the group of prof. Saumya Debray and has been partly supported by the Italian projects InterAbstract, AIDA and FIRB.

References

1. M. Christodorescu and S. Jha. Testing malware detectors. In *Proc. Internat. Symposium on Software Testing and Analysis. (ISSTA'04)*, 2004, pp. 33-44, ACM Press.
2. M. Christodorescu, S. Jha, S. A. Seshia, D. Song and R. E. Bryant. Semantics-aware malware detection. In *Proc. Symposium on Security and Privacy*, 2005, pp. 32-46, IEEE Press.
3. C. Collberg, C. Thomborson and D. Low. A taxonomy of obfuscating transformations. Technical Report #148, Dept. of Computer Science, The Univ. of Auckland, 1997.
4. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. Symp. on Principles of Programming Languages (POPL'77)*. 1977, pp. 238-252, ACM Press.
5. M. Dalla Preda and R. Giacobazzi. Control code obfuscation by abstract interpretation. In *Proc. Internat. Conf. on Software Engineering and Formal Methods (SEFM'05)*. 2005, pp. 301-310.
6. M. Dalla Preda and R. Giacobazzi. Semantic-based code obfuscation by abstract interpretation. In *Proc. 32nd Internat. Colloquium on Automata, Languages and Programming (ICALP'05)*. 2005, vol. 3580 of LNCS, pp. 1325-1336.
7. M.Dalla Preda, M. Madou, K. De Bosschere and R. Giacobazzi. Opaque predicate detection by abstract interpretation In *Proc. Internat. Conf on Algebraic Methodology and Software Technology (AMAST'06)* to appear.
8. J. Kinder, S. Katzenbeisser, C. Schallhart and H. Veith. Detecting malicious code by model checking. In *Proc. Conf. on Detection of Intrusions Malware & Vulnerability Assessment (DIMVA'05)* 2005, vol. 3548 of LNCS, pp. 174-187.
9. P. Szor. *The art of Computer Virus Research and Defence.* Addison Wesley, 2005.