Abstract Interpretation-based Protection

Robertto Giacobazzi

Università di Verona, Italy
roberto.giacobazzi@univr.it

Hiding information means both hiding as making it imperceptible and obscuring as making it incomprehensible [9]. In programming, perception and comprehension of code's structure and behaviour are deep semantic concepts, which depend on the relative degree of abstraction of the observer, which corresponds precisely to program semantics. In this tutorial we show that abstract interpretation can be used as an adequate model for developing a unifying theory for information hiding in software, by modeling observers (i.e., malicious host attackers) $\mathcal{O}$ as suitable abstract interpreters. An observation can be any static or dynamic interpretation of programs intended to extract properties from its semantics and abstract interpretation [2] provides the best framework to understand semantics at different levels of abstraction. The long standing experience in digital media protection by obscurity is inspiring here. It is known that practical steganography is an issue where compression methods are inefficient: *"Where efficient compression is available, information hiding becomes vacuous."* [1]. This means that the gain provided by compression can be used for hiding information. This, in contrast to cryptography, strongly relies upon the understanding of the supporting media: if we have a source which is completely understandable, i.e., it can be perfectly compressed, then steganography becomes trivial. In programming languages, a complete understanding of semantics means that no loss of precision is introduced by approximating data and control components while analysing computations. Complete abstractions [3, 8] model precisely the complete understanding of program semantics by an approximate observer, which corresponds to the possibility of replacing, with no loss of precision, concrete computations with abstract ones —some sort of perfect semantic compressibility around a given property. This includes, for instance, both static and dynamic, via monitoring, approaches to information disclosure and reverse engineering [4]. The lack of completeness of the observer is therefore the corresponding of its poor understanding of program semantics, and provides the key aspect for understanding and designing a new family of methods and tools for software steganography and obfuscation. Consider the simple statement, C : $\texttt{x} = \texttt{a} * \texttt{b}$, multiplying `a` and `b`, and storing the result in `x`. An automated program sign analysis replacing concrete computations with approximated ones (i.e., the rule of signs) is able to catch, with no loss of precision, the intended sign behaviour of C because the sign abstraction $\mathcal{O} = \{+, 0, -\}$, is complete for integer multiplication. If we replace C with $\mathfrak{O}(C)$: $\texttt{x} = 0$; $\texttt{if b} \leq 0 \texttt{ then } \{\texttt{a} = -\texttt{a}; \texttt{b} = -\texttt{b}\}$;

`while b` $\neq$ `0` $\{$`x = a + x; b = b - 1`$\}$ we obfuscate the observer $\mathcal{O}$ because the rule of signs is incomplete for integer addition. Intervals, i.e., a far more concrete observer, are required in order to automatically understand the sign computed in $\mathfrak{O}(C)$. We show how this idea can be extended to arbitrary obfuscation methods and exploited for code steganography, providing the basis for a unifying theory for these technologies in terms of abstract interpretation. We show how obfuscation can be viewed as a program transformation making abstractions incomplete and at the same time we show how watermark extraction can be viewed as a complete abstract interpretation against a secret program property, extending abstract watermarking [5] to any watermarking method. Both obfuscation and watermarking can be specified as transformers to achieve completeness/incompleteness in abstract interpretation [7], provided that the transformed code does not interfere with the expected input/output behaviour of programs. This latter correctness criteria can be again specified as a completeness problem by considering abstract non-interference [6] as the method for controlling information leakage in obfuscation and steganography. Our approach is language independent and can be applied to most known obfuscation and watermarking methods, providing a common ground for their understanding and comparison.

References

1. R.J. Andesron and F. Petitcolas. On the limits of steganography. *IEEE J. of Selected Areas in Communications*, 16(4):474–481, 1998.
2. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. of POPL '77*, pp. 238–252, 1977. ACM Press.
3. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. of POPL '79*, pp. 269–282, 1979. ACM Press.
4. P. Cousot and R. Cousot. Systematic design of program transformation frameworks by abstract interpretation. In *Proc. of POPL '02*, pp. 178–190, 2002. ACM Press.
5. P. Cousot and R. Cousot. An abstract interpretation-based framework for software watermarking. In *Proc. of POPL '04*, pp. 173–185, 2004. ACM Press.
6. R. Giacobazzi and I. Mastroeni. Abstract non-interference: Parameterizing non-interference by abstract interpretation. In *Proc. of POPL '04*, pp. 186–197, 2004. ACM-Press.
7. R. Giacobazzi and I. Mastroeni. Transforming abstract interpretations by abstract interpretation. In *Proc. of SAS'08*, LNCS 5079, pp. 1–17. Springer-Verlag, 2008.
8. R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretations complete. *J. of the ACM.*, 47(2):361–416, 2000.
9. F. A. P. Petitcolas, R. J. Anderson, and M. G. Kuhn. Information hiding – A survey. *Proc. of the IEEE*, 87(7):1062–1078, 1999.