Capturing the Essence of Practical Obfuscation

J. Todd McDonald

School of Computer and Information Sciences,
University of South Alabama, Mobile, AL USA.
E-mail: jtmcdonald@usouthal.edu

Abstract. In the realm of protecting programs from illegitimate use, obfuscation offers a modicum of defense against malicious reverse engineering and tampering. As a field of study, obfuscation would benefit from a unifying framework that has solid theoretical foundation yet provides value in empirical study and implementation. The essence of obfuscation (in practice) is best described as a measurable loss of abstraction. We argue that mathematical frameworks such as abstract interpretation and Boolean algebras may provide an ideal marriage of theory and practice, providing focused direction for future research.

1 Background

Obfuscation is a transformation process that produces semantically equivalent versions of a given program or circuit, with a definable security goal related to hiding or understandability of an original program. Notable obfuscation results prove that virtual black-box (VBB) simulation as an information theoretic goal is impossible for *all* functions [1] or for *all* functions with cryptographic applications [2]. Other theoretic models relate security to the distributions produced by the obfuscator itself [3, 4]. A steady stream of positive results over the last decade show that security can be proven in specific contexts or for specific families of programs: access control [5], point functions [6], re-encryption [7, 8], probabilistic encryption [9], private-key schemes [10], straight-line arithmetic programs [11], single use programs with hardware support [12], and time-limited obfuscation [13]. Formal expressions for obfuscation have also grown to include term rewriting systems [14] and abstract interpretation [15, 16].

Real-world obfuscators focus on preventing reverse engineering or tampering [17–19] to guard intellectual property investment. Successful reverse engineering aims to correctly identify system components, component interrelationships, or system representations at higher levels of abstraction [20], whether for beneficial or malicious purposes. Deobfuscation finds relevance in malware analysis and virus detection research [21, 22]. Circuits and embedded hardware protection [23] have likewise seen positive results, particularly where fully trusted [24, 25] or partially trusted hardware [26] is leveraged. Copy prevention through digital fingerprinting and watermarking [27, 28] and hiding of circuit features have been demonstrated empirically or proven generally [29, 30].

2 Practical Obfuscation as Loss of Abstraction

Two (often divergent) paths have shaped the meaning of obfuscation. In one approach, the obfuscator is defined mathematically with a corresponding security requirement that must be met for all possible programs: VBB, computational indistinguishability, best possible, non-malleability, etc. Based on the security test, proofs typically demonstrate negative results by producing a counter-example. Weakened definitions and specific contexts often provide positive results. Lifting specific constraints, such as semantic equivalence, can provide greater possibilities as well [31, 32]. Alternatively, creating obfuscators and making observations about their capabilities provides real-world context. In this paradigm, practical metrics or formal characterizations are in view: targeting steps in the reverse engineering process, computing loss of abstraction, preventing specific attack vectors, increasing cost-complexity for an adversary, relating adversarial tasks to known hard problems, or measuring heuristics that relate security to program attributes. Practitioners have a harder road to travel because characterization is not an exact science. Defining an adversary's power and correlating those to security objectives may require intuition or experience. Metrics and attack vectors may be valid today but change tomorrow as adversaries employ new strategies.

Real-world obfuscators operate primarily on the syntax/structural level (supporting real-world application) whereas theoretic definitions operate at the functional/semantic level (supporting statements about entire families). The fundamental property of obfuscation, from either perspective, is the characterization of *change*. White-box polymorphic variation is a structural transformation process that preserves functional semantics. Using program diversity itself (creating large numbers of functionally equivalent variants) can have stand-alone value as a remedy to piracy [33] and limiting impact of exploitable vulnerabilities. Obfuscation may best be described as the hiding of one or more semantic level properties, which is to say, achieving a measurable loss of abstraction. To mature the definition of obfuscation and lend credence to practical techniques, we need quantitative measures that adequately describe this loss and a framework that allows comparison of algorithms themselves.

The use of abstract interpretation (AI) [34] has shown promise as a method to achieve obfuscation [35, 16] and to support deobfuscation for malware detection [36, 37]. Following [16], a syntactic transformer (or variation process) is an *obfuscator* if it meets requirements for *potency*, *semantic equivalence*, and *conservative transformation*. The AI framework provides a method to clearly identify loss of abstract information by defining adversaries as properties that are either preserved or obfuscated. It also provides the tools for quantitative analysis of obfuscating transforms, whereby conservative transformations formally express the more fuzzy notion of obscurity. Abstract interpretation represents one contender for strengthening analytical studies, giving a rigorous, formal methodology that separates description of the variation engine from quantitative assessments of obfuscation.

3 Logic Programs: Bridging Theory and Practice

Boolean logic circuits are ideal for studying obfuscation because they bridge the gap between the often disparate worlds of theory and practice. They have rich history in theoretic study related to computational complexity, proof systems, logic algebras, and computational models. They rival Turing machines (TM) in answering fundamental questions of computability and possibility. Circuits have only one polynomial representation space (size), while Turing machines have two (size and running time). It is possible to show that all (physical) machines with bounded memory are constructible via sequential circuits and binary memory units. Further, we can completely simulate machines (whose computations terminate) with circuits. As an observation, many theoretical obfuscation results are expressed as proofs on circuit classes versus Turing machines.

Circuits also have real-world manifestations that are foundational to every day computing tasks, giving them the unique ability for practical realization. Focusing on logic circuits that derive from Boolean algebra may help establish foundational principles for obfuscation research. In addition, lessons learned from polymorphic circuit variation can be applicable to general programs: if the essence of obfuscation can be captured at this level, it may support an ideal marriage of theory and practice.

Circuit Representation and Variation: Circuits have a well studied semantic in the form of Boolean algebras and reasoning systems. Natural representation of Boolean algebras as lattices supports their evaluation under abstract interpretation. For example, a Boolean algebra β defines a lattice $\langle B, \leq \rangle$ where for any x and y in B, the set $\{x, y\}$ has a least upper bound defined as $x \vee y$ and a greatest lower bound defined as $x \wedge y$. Boolean algebras have clear relationships between syntax and semantics: physical circuits (traditionally AND, OR, NOR, $NAND$, XOR, $NXOR$, NOT gates) can be derived from logic statements (semantics) and logical expressions have multiple equivalent structural implementations. For example, the logical $\wedge$ operation maps to a physical OR gate and has a precise semantic, with the same holding true for the logical $\vee$. Given a complete basis set of operators $\{\vee, \wedge, \neg\}$, where $\neg$ represents negation, we can express any programmatic functionality. In addition, the lattice $\langle B, \leq \rangle$ immediately derives laws for idempotence, commutativity, associativity, absorption, partial orders for equivalence, and partial orders for implication. Circuit variation (a structural concept) is ultimately a reflection of Boolean logic variation (a semantic concept). We can formally define any variation technique as an application of one or more logic theorems.

Logical Redundancy and Reduction: In digital logic applications, considerable research has been geared at reducing circuits to their smallest (logical) form for efficiency and cost. Theoretically, obfuscators are allowed a polynomial overhead in circuit features such as size or depth. Practically, any overhead introduced might be viewed as a step backwards. Assuming that circuit candidates for obfuscation are in a minimally reduced form to begin with, circuit variation effectively introduces logical redundancy. Variation (aiming to achieve obfuscation) would essentially work backwards from the viewpoint of Boolean logic reduction.

Laws of idempotence, identity (0 and 1), complementation, and involution would provide the necessary generation mechanism for expanding terms. For example, reduction would replace the function $F_0 = (A \vee B \vee C) \wedge \neg(A \wedge \neg B \wedge \neg C) \wedge \neg C$ with its reduced form $F_0 = B \wedge \neg C$ via repeated application of logic laws. Going in reverse, variation would expand the function $F_0 = B \wedge \neg C$ forward and $F_0 = (A \vee B \vee C) \wedge \neg(A \wedge \neg B \wedge \neg C) \wedge \neg C$ represents one possible expansion. As a key observation, redundancy introduced by variation algorithms *should* be reducible. From empirical study [38, 30], any single transformation (an aspect of confusion) is trivially reducible; however, combinations of overlapping redundancies (an aspect of diffusion) may not be removable via heuristic methods. These observations validate Cohen's [39] original approach to programmatic transformation using confusion/diffusion primitives. Redundancy and reduction form a primary analysis aspect of circuits, where Boolean logic reduction serves as an adversarial analysis vector. Other circuit features such as topology, signals, components, and control provide primary means for assessing loss of abstraction.

4 Conclusions

Empirical study of practical obfuscation can be a trial and error method. Defining security properties of interest and developing variation algorithms that obscure those properties in a quantitative approach would serve to validate practical techniques. Several questions of interest arise in this pursuit:

1. How do we characterize the polymorphic white-box variation process?
2. How do we know when variation becomes obfuscation (a loss of abstraction)?
3. How do we compare different obfuscation algorithms?
4. How do we quantitatively asses the overhead of introduced redundancy with the protection gained?

As a way forward, formal analysis frameworks such as abstract interpretation and Boolean logic algebras provide a good starting point that may ultimately bring theory and practice closer together.

References

1. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (im)possibility of obfuscating programs. Electronic Colloquium on Computational Complexity **8** (2001)
2. Goldwasser, S., Kalai, Y.T.: On the impossibility of obfuscation with auxiliary input. In: Proc. of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS '05), Washington, DC, USA, IEEE Computer Society (2005) 553–562
3. Goldwasser, S., Rothblum, G.N.: On best-possible obfuscation. In: Proceedings of the 4th conference on Theory of cryptography. TCC'07, Berlin, Heidelberg, Springer-Verlag (2007) 194–213
4. Yasinsac, A., McDonald, J.T.: Tamper resistant software through intent protection. Intl Journal Network Security **7** (2008) 370–382

5. Lynn, B., Prabhakaran, M., Sahai, A.: Positive results and techniques for obfuscation. In: EUROCRYPT. (2004) 20–39
6. Wee, H.: On obfuscating point functions. In: Proc.of the 37th Annual ACM Symposium on Theory of Computing (STOC '05), New York, NY, USA, ACM (2005) 523–532
7. Hohenberger, S., Rothblum, G.N., Shelat, A., Vaikuntanathan, V.: Securely obfuscating re-encryption. In: TCC. (2007) 233–252
8. Chandran, N., Chase, M., Vaikuntanathan, V.: Collusion resistant obfuscation and functional re-encryption. IACR Cryptology ePrint Archive **2011** (2011) 337
9. Hofheinz, D., Malone-Lee, J., Stam, M.: Obfuscation for cryptographic purposes. Journal of Cryptology **23** (2010) 121–168 10.1007/s00145-009-9046-1.
10. Hada, S., Sakurai, K.: A note on the (im)possibility of using obfuscators to transform private-key encryption into public-key encryption. In Miyaji, A., Kikuchi, H., Rannenberg, K., eds.: Advances in Information and Computer Security. Volume 4752 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2007) 1–12 10.1007/978-3-540-75651-4.
11. Narayanan, S., Raghunathan, A., Venkatesan, R.: Obfuscating straight line arithmetic programs. In: Proceedings of the nineth ACM workshop on Digital rights management. DRM '09, New York, NY, USA, ACM (2009) 47–58
12. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: One-time programs. In: Proceedings of the 28th Annual conference on Cryptology: Advances in Cryptology. CRYPTO 2008, Berlin, Heidelberg, Springer-Verlag (2008) 39–56
13. Beaucamps, P., Filiol, E.: On the possibility of practically obfuscating programs towards a unified perspective of code protection. Journal in Computer Virology **3** (2007) 3–21
14. Walenstein, A., Mathur, R., Chouchane, M.R., Lakhotia, A.: Normalizing metamorphic malware using term rewriting. In: SCAM '06: Proceedings of the Sixth IEEE, Washington, DC, USA, IEEE Computer Society (2006) 75–84
15. Dalla Preda, M., Giacobazzi, R., Debray, S., Coogan, K., Townsend, G.: Modelling metamorphism by abstract interpretation. In: Proc. of the 17th. International Static Analysis Symposium (SAS). (2010)
16. Dalla Preda, M., Giacobazzi, R.: Semantic-based code obfuscation by abstract interpretation. In: ICALP. (2005) 1325–1336
17. Majumdar, A., Thomborson, C.: Manufacturing opaque predicates in distributed systems for code obfuscation. In: ACSC '06: Proceedings of the 29th Australasian Computer Science Conference, Darlinghurst, Australia, Australia, Australian Computer Society, Inc. (2006) 187–196
18. Collberg, C., Thomborson, C.: Watermarking, tamper-proofing, and obfuscation - tools for software protection. IEEE Transactions on Software Engineering **28** (2002) 735–746
19. Madou, M., Anckaert, B., Moseley, P., Debray, S., Sutter, B.D., Bosschere, K.D.: Software protection through dynamic code mutation. In: Proc. of the 6th Int"l Workshop on Information Security Applications. (2005) 194–206
20. Chikofsky, E., Cross, J.H., I.: Reverse engineering and design recovery: a taxonomy. Software, IEEE **7** (1990) 13–17
21. Lakhotia, A., Kumar, E.U., Venable, M.: A method for detecting obfuscated calls in malicious binaries. IEEE Transactions on Software Engineering **31** (2005) 955–968
22. Christodorescu, M., Jha, S., Seshia, S.A., Song, D., Bryant, R.E.: Semantics-aware malware detection. In: Proceedings of IEEE Symposium on Security and Privacy, Washington, DC, USA, IEEE Computer Society (2005) 32–46

23. Kim, Y.C., McDonald, J.T.: Considering software proteciton for embedded systems. Crosstalk: The Journal of Defense Software Engineering **22** (2009) 4–8
24. Chandran, N., Goyal, V., Sahai, A.: New constructions for uc secure computation using tamper-proof hardware. In Smart, N., ed.: Advances in Cryptology EUROCRYPT 2008. Volume 4965 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2008) 545–562 10.1007/978-3-540-78967-3.
25. Ding, N., Gu, D.: A general and efficient obfuscation for programs with tamper-proof hardware. In: Proc. of the 7th Int"l Conference on Information Security Practice and Experience. ISPEC'11, Berlin, Heidelberg, Springer-Verlag (2011) 401–416
26. Bitansky, N., Canetti, R., Goldwasser, S., Halevi, S., Kalai, Y.T., Rothblum, G.N.: Program obfuscation with leaky hardware. Cryptology ePrint Archive, Report 2011/660 (2011) http://eprint.iacr.org/.
27. Castillo, E., Meyer-Baese, U., García, A., Parrilla, L., Lloris, A.: Ipp@hdl: Efficient intellectual property protection scheme for ip cores. IEEE Trans. Very Large Scale Integr. Syst. **15** (2007) 578–591
28. Charbon, E., Torunoglu, I.: Watermarking techniques for electronic circuit design. In: Proceedings of the 1st international conference on Digital watermarking. IWDW'02, Berlin, Heidelberg, Springer-Verlag (2003) 147–169
29. Chakraborty, R.S., Bhunia, S.: Hardware protection and authentication through netlist level obfuscation. In: Proc. of the IEEE/ACM Int"l Conference on Computer-Aided Design. ICCAD '08, Piscataway, NJ, USA, IEEE Press (2008) 674–677
30. McDonald, J.T., Trias, E.D., Kim, Y.C., Grimaila, M.R.: Using logic-based reduction for adversarial component recovery. In: Proc. of the 25th ACM Symposium on Applied Computing (SAC). (2010)
31. Sander, T., Tschudin, C.: On software protection via function hiding. In: Information Hiding. Volume 1525 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (1998) 111–123 10.1007/3-540-49380-8.
32. McDonald, J.T., Kim, Y.C., Yasinsac, A.: Software issues in digital forensics. ACM Operating Systems Review **42** (2008)
33. Anckaert, B., Sutter, B.D., Bosschere, K.D.: Software piracy prevention through diversity. In: DRM '04: Proceedings of the 4th ACM workshop on Digital rights management, New York, NY, USA, ACM (2004) 63–71
34. Cousot, P.: Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. Theor. Comput. Sci. **277** (2002) 47–103
35. Dalla Preda, M., Giacobazzi, R., Visentini, E.: Hiding software watermarks in loop structures. In Alpuente, M., Vidal, G., eds.: SAS. Volume 5079 of Lecture Notes in Computer Science., Springer (2008) 174–188
36. Dalla Preda, M., Madou, M., Bosschere, K.D., Giacobazzi, R.: Opaque predicates detection by abstract interpretation. In Johnson, M., Vene, V., eds.: AMAST. Volume 4019 of Lecture Notes in Computer Science., Springer (2006) 81–95
37. Dalla Preda, M., Christodorescu, M., Jha, S., Debray, S.: A semantics-based approach to malware detection. SIGPLAN Not. **42** (2007) 377–388
38. McDonald, J.T., Kim, Y.C., Grimaila, M.R.: Protecting reprogrammable hardware with polymorphic circuit variation. In: Proc. of the 2nd Cyberspace Research Workshop 2009. (2009)
39. Cohen, F.B.: Operating system protection through program evolution. Comput. Secur. **12** (1993) 565–584