Analysis of Obfuscation as a Protection Mechanism for Distributed Source Code

Prepared by Brayden Bowler for Dr. Steve Hendrikse

SECU73000 – ASN 2

Due: October 21, 2022

Contents

Obfuscation Analysis

In Canada, software is protected under intellectual property law with all the rights that the term entails (Government of Canada, 2022). However, even with the legal protections afforded, intellectual property ("IP") cases are often intricate, lengthy, and costly (Government of Canada, 2021). If the goal, then, is to avoid such events what other methodologies can entities use to protect their source code?

The concept of program obfuscation is one such possible methodology and will be the focus of this paper. Obfuscation refers to the process of mystifying a program's source code so that it is no longer intelligible to humans (You & Yim, 2010). After the obfuscation process, the program remains just as functional as before, but the source code can no longer be read in any plain language.

The application of obfuscation to the realm of IP protection, at least theoretically, would make great sense. Companies could obfuscate their programs prior to delivery, thereby ensuring their exact source code could not be read, while still allowing users access to the functionality of the program.

However, the major drawback of obfuscation is that the original program can often be reverse engineered from a copy of the obfuscated code. The source code can also be retrieved as an instruction set from the processor (Vigna, 2007). Matthew Green (2014), a John Hopkins University cryptologist summarizes the deficiency succinctly: obfuscation techniques are not mathematically rigorous enough (para. 12). Suffice to say that most obfuscation efforts amount to little more than roadblocks for those skilled enough to know how to unravel the obfuscation process. In a world where time, talent, and tools are abundant roadblocks may not be

satisfactory security. The question now becomes: can we modify obfuscation techniques enough for them to become a viable source code protection tool?

To start, a clearer definition of obfuscation is required. Barak et al. (2010) provide guidance on a clearer description of an obfuscator. Crucially, the landmark paper adds that an obfuscator must satisfy two requirements. First, that the obfuscated program has the same functionality as it's legible counterpart. Second, that any conclusion that can be made about the source code through access to the obfuscated program can be efficiently concluded with oracle access[1] to the legible counterpart (pg. 2).

Restated, the second requirement suggests that an entity should gain no additional information about the source code from access to the obfuscated copy of the program than they would gain from interacting with the legible program via oracle access.

Rather frustratingly, Barak et al. (2010) go on to prove that, under their newly defined notion of an obfuscator, true obfuscation is impossible. The authors developed a set of programs that were unobfuscatable in that access to the obfuscated program always yielded more conclusions about the source code than could be gained by any algorithm with oracle access to the legible program (pg. 4).

Now this would appear to be the full stop to the previously stated question – that perhaps obfuscation is not a reliable technique for protecting source code. However, cryptologists are persistent.

When Barak et al. proved the impossibility of general-purpose obfuscation under their notion of obfuscator requirements, the group hinted at other, softer, notions of obfuscation that were theoretically achievable. A group of cryptological researchers from

[1] Oracle access suggests that a user can query a program with inputs, and view the corresponding output, but has no additional access to the program or it's source code.

UCLA and IBM fleshed out the idea of indistinguishability obfuscation (Garg et al., 2013). A group of authors from Cornell arrived at extractability obfuscation, a derivative of indistinguishability obfuscation at around the same time (Boyle at al., 2014). Indistinguishability obfuscation proposes a new set of requirements for an obfuscator to adhere to. The re-invented definition states that the obfuscation of two equivalent programs should be computationally indistinguishable from one another (Garg et al., 2013, pg. 1). Garg et al. (2013) further surmise that their candidate obfuscator could find practical application in the restricted-use software space. Particularly, as an intellectual property mechanism (pg. 7).

The alleged application of indistinguishability obfuscation involves a particular case – some entity has developed a new piece of IP (say an algorithm) that closely resembles existing and widely sourced IP. Since their obfuscator yields the same obfuscation results when programs are similar, a small change to IP may be obfuscated without distinguishability between the two algorithms.

This application of obfuscation provokes two concerns. First, if some algorithm already exists, and is widely adopted, would a new algorithm, treated as secret IP by the developing entity, really justify such protection? Any entity could simply use the open-source version to achieve the same result. Second, how does the indistinguishability of two obfuscated programs attest, at all, to the security of the actual obfuscation process itself? It would not matter if an adversary could not fully distinguish between two obfuscated programs if they were able to reverse-engineer both, or either, back to plain-text. While the first concern is a matter of business objective, the second is worth investigating.

It is understood that stylistic as well as functional identifiers exist within source code which can be analyzed to attribute authorship (Krsul and Spafford, 1997). Further evolutions of study on similar topics suggest that stylistic and functional identifiers persist through the compilation process that assist in authorship attribution, and even reverse-engineering source code in the case of the functional identifiers (Rosenblum et al., 2011). Finally, research was conducted on whether these same identifiers could be extracted from obfuscated binaries. As it turns out, even obfuscated binary files maintain enough functional and stylistic identifiers such that a skilled individual (or machine learning algorithm) could separate the obfuscated distractions from the source code (Hendrikse, 2017).

Based on the recent literature, it's clear that tools exist that are capable of disassembling obfuscated source code. If indistinguishability obfuscation suggests that two obfuscated programs should be indistinguishable from one another and yet either, or both, pieces of obfuscated code can be disassembled, then the notion of using this specific technique to protect IP is absurd.

Circling back, it seems that and the original definition of obfuscation proposed by Barak et al. would in fact need to be met for program obfuscation to securely protect the underlying source code. Becoming more lenient with the definition of obfuscation reduces the security of the obfuscated program such that any IP is at risk of a skilled adversary disassembling the obfuscated program.

To conclude, obfuscation techniques are not yet sufficient as an IP protection mechanism for general-purpose programs. Too easily are the obfuscation techniques disassembled and the original source code made accessible.

Future research on the topic matter could take two paths. One avenue of valid research would be to investigate new methods of obfuscation that would adhere to the principles laid forth in Barak et al. If some such obfuscation algorithm could be formulated that complies with the black-box principle, then source code security would be changed forever.

Alternatively, further research could be conducted on entirely new mechanisms for protecting source code IP. Can we limit access to source code by allowing users to remotely access our programs instead of distributing our programs? Could we protect the internal state of a processor enough so that an adversary could not extract instruction sets to reverse engineer programs? How would this mechanism interact with other hardware architecture to ensure normal computing functionality in all other cases?

Though these questions are fascinating, they are far beyond the scope of this work. Simply stated the objective of this paper was to review modern obfuscation literature and determine whether modern obfuscators are secure enough to protect distributed source code from disassembly. Based on the facts reviewed it's plain to see that modern obfuscation techniques are not sufficient.

References

Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., & Yang, K. (2010). *On the (Im)possibility of Obfuscating Programs*. https://www.wisdom.weizmann.ac.il/~oded/PS/obf4.pdf

Boyle, E., Chung, K., & Pass, R. (2014). *On Extractability (a.k.a. Differing Inputs) Obfuscation*. https://eprint.iacr.org/2013/650.pdf

Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., & Waters, B. (2013). *Candidate Indistinguishability Obfuscation and Functional Encryption of all circuits.* https://eprint.iacr.org/2013/451.pdf

Government of Canada. (2022). *Intellectual property rights in software in Canada* (2022 ed.). https://www.ic.gc.ca/eic/siTe/cipointernet-internetopic.nsf/vwapj/CIPOCS-1965-_wr05067-eng.pdf/$file/CIPOCS-1965-_wr05067-eng.pdf

Government of Canada. (2021). *Settling intellectual property disputes in court.* https://www.ic.gc.ca/eic/site/cipointernet-internetopic.nsf/eng/h_wr04900.html

Green, M. (2014). *Cryptographic obfuscation and 'unhackable' software*. https://blog.cryptographyengineering.com/2014/02/21/cryptographic-obfuscation-and/

You, I. & Yim, K. (2010). *Malware Obfuscation Techniques: A Brief Survey.* DOI: 10.1109/BWCCA.2010.85.

Krsul, I., & Spafford, E. H. (1997). Authorship analysis: Identifying the author of a program. *Computers & Security, 16*(3), 233-257.

Rosenblum, N., Zhu, X., & Miller, B. P. (2011). Who wrote this code? Identifying the authors of program binaries. *Computer Security – ESORICS 2011* (pp. 172-189).

Steven Hendrikse. (2017). *The Effect of Code Obfuscation on Authorship Attribution of Binary Computer Files.* https://nsuworks.nova.edu/gscis_etd/1009.

Vigna, G. (2007). *Static disassembly and code analysis*. Malware Detection (pp. 19-41).