Towards a Theory of Special-purpose Program Obfuscation

Muhammad Rizwan Asghar, **Steven Galbraith**, Andrea Lanzi,
Giovanni Russello, Lukas Zobernig

University of Auckland, New Zealand
Universita degli studi di Milano, Italy

Thanks

- Thanks to the conference organisers in this difficult year.
- Thanks to the Trustcom reviewers for their positive comments.
- We thank Christian Collberg for several suggestions on the early version of this work, and especially for pointing out the connection with obfuscation competitions.
- This research is funded in part by the Royal Society of New Zealand Marsden fund project 16-UOA-144.

Plan

- The motivation for special-purpose obfuscation.
- Our formalism.
- Obfuscation competitions.
- Example of applying the formalism.
- Composition of obfuscations.
- Limitations

Full version of paper: `https://arxiv.org/abs/2011.02607`

I'm happy to answer questions or discuss the paper by email: s.galbraith (at) auckland.ac.nz

Obfuscation

- Program obfuscation is often informally stated as "to hide semantic properties of program code from a Man-At-The-End (MATE) attacker who has access to an executable file of the program".
- What does this mean?
- More than twenty years of research in software security has not given a rigorous and practical formalism for what it means to "hide semantic properties of program code".
- How can we formally prove that an obfuscation scheme "hides semantic properties"?

Cryptographic approaches to obfuscation

- In 1996, Goldreich and Ostrovsky suggested that obfuscation should prevent "the release of any information about the original program which is not implied by its input/output relation and time/space complexity".
- The landmark paper by Barak, Goldreich, Sahai, Impagliazzo, Vadhan and Yang in 2001 introduced the notion of Virtual Black Box (VBB) obfuscation, which means attacker can learn nothing more, when given the code, than what could be learned from running it (i.e., oracle access).
- Unfortunately, Barak et al show that this notion is impossible to achieve in general.

Two worlds of obfuscation

- Software security: Gives general purpose and practical obfuscation tools, but no formal definitions or security proofs.
- Cryptography: Gives formal definitions that are very strong, but no practical general-purpose obfuscation tools.
  (Recent paper claims complete solution: Aayush Jain, Huijia Lin and Amit Sahai, "Indistinguishability Obfuscation from Well-Founded Assumptions")
- How to bridge this gap? Can we have practical obfuscation tools and also security proofs?
- Other researchers have noticed this problem, eg Kuzurin, Shokurov, Varnovsky and Zakharov (2007), Xu, Zhou, Kang and Lyu (2017).

Two worlds of obfuscation

- It is not known how to combine practical solutions and theoretical analysis.
- We feel *general-purpose* obfuscation is too ambitious and too hard to achieve.
- Further, general-purpose obfuscation is probably unnecessary for most practical applications.
- In practice, software developers usually have a particular secret or intellectual property that they wish to protect, rather than desiring to secure “all semantic properties” of a program.
- Hence we argue for *special-purpose obfuscation*.

Special-purpose obfuscation

- There is a good track record of special-purpose obfuscation giving provably secure yet practical obfuscators.
- The aim of this paper is to give a formalism for special-purpose obfuscation.
- We hope it is a useful starting point to address some of these open problems, and that it will be further extended by the research community.
- Importantly, the formalism is rigorous ands falsifiable.
- Our formalism uses the notion of **asset**, which has been considered by many authors.
- See the paper for more discussion. `https://arxiv.org/abs/2011.02607`

Our model

There are three entities in our formalism:

- **Programmer/developer:** A human who has developed a program (written in some high-level source code) that contains some asset or intellectual property.
- **Obfuscator/defender:** A randomised algorithm that takes as input a program in some format and outputs a program in some format.
- **Attacker/de-obfuscator:** An algorithm that takes program code in some format and tries to learn the asset in the program.

Key features of our formalism:

- **Program class**
- **Asset verifier**

Program class

- A **program class** $\mathcal{C}$ is a set of programs (or program segments), all written in the same language. For each $n \in \mathbb{N}$, there is a (possibly empty) subset $\mathcal{C}_n \subseteq \mathcal{C}$.
- Can think of $\mathcal{C}_n$ as programs with n-bit input, or n-bit output, or where some internal state has n bits.
- $P \in \mathcal{C}_n$ runs in time bounded by $p(n)$.
- $\mathcal{C}_n$ is asymptotically super-polynomial in size,
- There is an efficient **program generator** $\mathrm{Gen}_{\mathcal{C}}$ for the class $\mathcal{C}$.
- Formally, $\mathrm{Gen}_{\mathcal{C}}(n)$ outputs $C \in \mathcal{C}_n$ and auxiliary data aux.

Assets

- An **asset space** $\mathcal{A}$ for $\mathcal{C}$ is a family of sets $(\mathcal{A}_n)_{n\in\mathbb{N}}$.
- An **asset** for $\mathcal{C}$ is a sequence of functions $\mathrm{a}_n : \mathcal{C}_n \to \mathcal{P}(\mathcal{A}_n)$, where $\mathcal{P}(\mathcal{A}_n)$ is the power set of $\mathcal{A}$.
- There is an efficient **asset verifier** $\mathcal{V}$ such that for all $n \in \mathbb{N}$ and all $(P, \mathrm{aux}) \leftarrow \mathrm{Gen}_{\mathcal{C}}(n)$ (so $P \in \mathcal{C}_n$), and all $a \in \mathcal{A}_n$, $\mathcal{V}(P, \mathrm{aux}, a)$ outputs 1 if $a \in \mathrm{a}_n(P)$ and 0 otherwise.
- Note: Asset does not have to be unique for a given program P.

Our model

- A **secure obfuscator** for $(\mathcal{C}, \mathcal{A}, \mathrm{a})$ is a randomised algorithm $\mathrm{Obf}(P, \mathrm{aux})$ that outputs P'.
- Correctness: $P'(x) = P(x)$ for all x.
- Efficiency: P' runs in time polynomial in the running time of P.
- Security: An attacker is a polynomial-time algorithm A that knows $\mathrm{Gen}_{\mathcal{C}}, \mathrm{Obf}, \mathcal{V}$. The attacker wins if

$$\mathcal{V}(P, \mathrm{aux}, A(P')) = 1.$$

The obfuscator is secure if the probability an adversary wins, over $(P, \mathrm{aux}) \leftarrow \mathrm{Gen}_{\mathcal{C}}(1^n)$ and $P' = \mathrm{Obf}(P, \mathrm{aux})$, is negligble.

Our model

- Program classes are defined by software developers.
- The requirement that $\mathcal{C}$ is defined by a random program generator is analogous to Kerckhoff's principle.
- It is the software developer's job to define "useful" or "meaningful" assets.
- We require that assets are not learnable from just running the program and observing input-output.

Obfuscation Competitions

- The problem of defining obfuscation security is exactly the same as providing automated obfuscation challenges.
- Example: the Tigress project reverse engineering challenges.
- For a challenge, one needs to have a well-defined class of programs (that is known to the competitors) and an algorithm Gen to generate a random program from this class.
- One needs a well-defined security goal (the asset).
- One needs an efficient and automated method to judge if a contestant has found the asset (the asset verifier).
- See the paper for more discussion.

A simple example of the formalism in action

- *Point functions* are functions that output 1 on input a fixed string and 0 otherwise.
- Examples include licence checks and password checks.
- Obfuscating point functions using hash functions is well-understood.
- Class $\mathcal{C}_n$ is the set of programs that take an n-bit input x, and are zero on all except one input $c \in \{0,1\}^n$ (the password or the licence code).
- Gen chooses a random $c \in \{0,1\}^n$ and outputs $P(x)$ that is `(x == c)`.
- We set $\mathrm{aux} = c$.

A simple example of the formalism in action

- The asset is the secret value $c \in \{0,1\}^n$.
- The asset verifier checks that a is equal to c.
- An *obfuscator for point functions* chooses a random $r \in \{0,1\}^n$, sets $c' = H(r\|c)$ and outputs the program P' given by (r, c') and $H(r\|x) == c'$.
- One can prove this is a secure obfuscator if the hash function H is hard to invert.
  (See the paper.)

Real-World Examples

The following problems can be studied using our formalism (we don't claim solutions to all these problems):

- Evasive functions (e.g. Hamming distance, pattern matching with wild cards, fuzzy biometric matching).
- White-Box Cryptography.
- Evasive Finite Automata.
- Machine Learning

See paper for explanation of what is the program class and asset.

Results

- Steven D. Galbraith and Lukas Zobernig, Obfuscated Fuzzy Hamming Distance and Conjunctions from Subset Product Problems, in D. Hofheinz and A. Rosen (eds.), Theory of Cryptography TCC, Springer LNCS 11891 (2019) 81–110.
- Steven D. Galbraith and Lukas Zobernig, Obfuscating Finite Automata, to appear in proceedings of SAC 2020.
- Steven D. Galbraith and Trey Li, Big Subset and Small Superset Obfuscation, eprint 2020/1018.

Composition of Obfuscations

- Section 2.1 of Schrittwieser, Katzenbeisser, Kinder, Merzdovnik and Weippl (2016) note that “many commercial obfuscators employ (and indeed recommend to use) multiple obfuscations at the same time”.
- We give the first formal analysis of this.
- Suppose obfuscators Obf_1 and Obf_2 are each designed to protect a different asset in a different way.
- We show (under certain conditions) that the composition $\mathrm{Obf}_2(\mathrm{Obf}_1(P))$ protects both assets.
- Note that “composition of obfuscation” has a different meaning in the cryptography literature.

Limitations of Our Formalism

- Choosing the Right Asset: A developer might think they have defined their asset adequately, but have failed to anticipate an attack that computes some related information or partial information.
- Control Flow Obfuscation:
  1. It is necessary to specify a class $\mathcal{C}$ of programs that have "rich" control flow.
  2. If a(P) is the Control Flow Graph (CFG) of the code produced by the software developer, and obfuscated program P' has "more complex" CFG, then $\mathcal{V}$ needs to decide if the original CFG has been computed.

Conclusion

- We argue that general-purpose obfuscation is too hard and ambitious, and may not be needed in practice.
- We encourage study of special-purpose obfuscation.
- We have presented a new formalism.
- Future work will be to find new applications and obfuscation solutions.

Thank You