Bypassing Malware Obfuscation with Dynamic Synthesis

Fabrizio Biondi, Sébastien Josse, Axel Legay

Bypassing Malware Obfuscation with Dynamic Synthesis

by Fabrizio Biondi, Sébastien Josse, and Axel Legay, Inria

Black-box synthesis is more efficient than SMT deobfuscation on predicates obfuscated with Mixed-Boolean Arithmetics

Malware - programs that exhibit malicious behaviour - poses a significant threat to computer security. Government, industry, and individuals spend billions of euros each year to defend themselves from malware or recover from malware attacks. To fight malware, antivirus programs such as those produced by Norton, Kaspersky and Malwarebytes must be able to determine whether a file contains malware; this is known as malware analysis. Malware analysis can be performed either on-demand, i.e., when requested by the user, or on-access, i.e., when the user invokes a file and it has to be analysed before the system runs it. On-access analysis has to be completed in a very short time to avoid making the system unresponsive. The TAMIS team at the High Security Lab of Inria Rennes is pushing the state of the art in open-source automated malware analysis by unifying many different techniques into a single tool and developing new theoretical foundations for malware deobfuscation and classification.

Executable files are the most common carriers of malware, since they already have execution privileges. Since statically analysing or dynamically executing every possible execution trace of the executable would be prohibitively expensive, particularly in the on-access analysis scenario, antivirus programs use symbolic analysis. Important tools for symbolic analysis include the KLEE-based S2E tool and the popular IDA Pro reverse engineering environment. Symbolic analysis represents a set of traces as a set of constraints over the variables of the traces, and are able to analyse all such traces at once. However, this depends on the ability to create and maintain a set of constraints representing such traces. SMT solvers are employed to decide the satisfiability of the constraints.

To hinder symbolic analysis, malware creators employ obfuscation techniques [1]. Obfuscation consists of complicating the malware's code into an equivalent representation that is harder to analyse symbolically. For instance, obfuscation can be used to:

- Make the control flow of the executable depend on symbolic variables by inserting into the assembly code of the executable a conditional jump statement whose target depends on a variable. The analysis will not be able to coherently keep track of all possible executions of the program since they depend on the possible values of the variable, and thus will not be able to know how to proceed with the analysis of the traces.
- Embed a virtual machine with instructions generated on-the-fly in the code and execute the real code on the virtual machine, making it harder to follow the program flow.
- Encrypt and pack parts of the malware to be decrypted and executed from memory only at runtime, hindering static analysis.
- Modify the malware by acting on its memory space.
- Bloat the size of the constraint set to be maintained by the symbolic representation. This is the case with Mixed-Boolean Arithmetic (MBA) obfuscation [2], where a polynomial function f and some statements equivalent to zero $e_i...e_k$ are used to transform and diversify a given statement s into a complex equivalent statement using the formula

$$s = f^{-1}\left(\sum_{i=1}^{k} e_i + f(s)\right)$$

making it much harder to analyse. MBA obfuscation is commonly used by DRM systems deployed by Irdeto, for instance, but has also been detected in malware compilation chains.

- Make it impossible for the antivirus to reconstruct the control flow of the malware, thus hindering fingerprinting and allowing the malware creator to hide malicious code in parts of the binary that the analysis mistakenly considers as dead code. Obfuscating conditionals (e.g., with MBA obfuscation) makes it hard for the antivirus to recognize which parts of the code are reachable and which are dead.

Given the wide variety of available techniques and their complexity, it is very hard to counteract obfuscation in the limited time given by the on-access scenario. We will focus on the last of the cases listed, where the common deobfuscation approach is to use SMT solvers like STP, Z3 or Boolector to determine the satisfiability of the MBA-obfuscated conditionals.

The solution we propose is to employ dynamic synthesis techniques [3] to decide satisfiability instead of deobfuscation. While deobfuscation tries to understand the behaviour of an obfuscated statement by studying it, synthesis instead interrogates the statement as a black box by feeding it inputs and studies the corresponding outputs. Since a conditional statement is just a function from its inputs to a Boolean value, finding inputs for which the statement can be true and false is sufficient to determine its satisfiability. Hence, synthesis is able to determine the satisfiability of a statement without having to understand it, just by analysing its input-output behaviour on an appropriate number of experiments.

We have analysed MBA-obfuscated 64-bit conditional statements and tried to understand their behaviour using multiple state-of-the-art SMT solvers, and using our own synthesis-based approach. Since the strength of the MBA obfuscation depends on the degree of the polynomial used, we have experimented with different polynomial degrees. The results are reported in our working paper, available at https://hal.inria.fr/hal-01241356v1 , and summarised in the following table:

MBA polynomial degree	Solution time for SMT solvers (s)			Dynamic synthesis time (s)
	STP	Z3	Boolector	
2	0.679	3.680	0.748	5.022
3	4.339	26.976	3.183	6.000
4	17.368	101.520	31.916	6.407
5	54.895	172.228	36.439	7.935

Table 1: Solution time for SMT solvers and dynamic synthesis against MBA obfuscation

It is clear from the table that dynamic synthesis is less affected by the degree of the polynomial used in the obfuscation than SMT solvers. Since the size of the obfuscated statement grows exponentially with the degree of the obfuscation polynomial, SMT solving time also tends to grow significantly. On the other hand, since synthesis just interrogates the obfuscated statement, the statement's size is much less relevant. While on small-degree polynomials SMT solvers outperform synthesis, such a low level of obfuscation is not representative of the sophisticated techniques used by malware in the wild. On the other hand, the scalability of the synthesis method shows promise against real malware obfuscation, since synthesis sidesteps the deobfuscation problem.

On-access malware analysis is a crucial protection against infection, and efficient deobfuscation techniques are fundamental to detecting malware with limited time. Thanks to synthesis, we will be able to create more efficient malware analysis techniques, taking an important step towards a more secure cyberenvironment.

Our research is supported by the Pôle d'excellence Cyber, bringing together Inria, DGA, Supélec and the Bretagne region.

[1] S. Schrittwieser, S. Katzenbeisser, J. Kinder, G. Merzdovnik, and E. Weippl: *Protecting Software through Obfuscation: Can It Keep Pace with Progress in Code Analysis?,* ACM Computing Surveys (CSUR) 49 (1), 4.

[2] Y. Zhou, A. Main, Y. X. Gu, and H. Johnson: *Information Hiding in Software with Mixed Boolean-Arithmetic Transforms,* Information Security Applications: 8th International Workshop, WISA 2007, Jeju Island, Korea, August 27-29, 2007, Revised Selected Papers.

[3] R. Balaniuk: *Drill and join: A method for exact inductive program synthesis,* in Logic-Based Program Synthesis and Transformation - 24th International Symposium, LOPSTR 2014, Canterbury, UK, September 9-11, 2014. Revised Selected Papers.

Please contact:
Fabrizio Biondi and Axel Legay
Inria, France
fabrizio.biondi@inria.fr and axel.legay@inria.fr