Poster: Comparative Evaluation of the Effectiveness of Constraint Solvers against Opaque Conditionals

Fabrizio Biondi
ESTASYS Team
INRIA/IRISA (France)
fabrizio.biondi@inria.fr

Sébastien Josse
DGA (French Ministry of Defense)
sebastien.josse@polytechnique.edu

Axel Legay
ESTASYS Team
INRIA/IRISA (France)
axel.legay@inria.fr

Software protection against reverse engineering has become a subject of interest for security researchers. *Obfuscation* transformations are designed to increase the cost of information extraction. We note that the analysis of obfuscated code is hampered by the presence of *opaque conditionals*, that is to say expressions conditioning connections to different execution paths of the program. These expressions are opaque to the extent that it is difficult for an analyst to determine their values. As a consequence, the analyst is unable to extract manually or statically a relevant representation of the program, starting with the control flow graph. It is therefore necessary to execute (concretely or symbolically) the program to determine the values of these expressions.

Symbolic and concrete/symbolic analyses use constraint solvers (SMT solvers) to decide the truth value of an expression conditioning a branch in the target code. We note that the effectiveness of these solvers against such obfuscation mechanisms has little been studied. Many techniques for hiding both data and control flow have been proposed, along with evidence of their resilience against static analysis. Experience shows that many of them do not provide acceptable security when assessed by analysts in the real world, using a conjunction of static, symbolic and dynamic analysis tools.

We illustrate this problem by studying a candidate method [ZMGJ07, JXZ08] to generate opaque conditionals. Authors of this (patented) method claim that "Such transforms resist reverse engineering with existing advanced tools and create NP-hard problems for the attacker". In particular, they assess the robustness of these opaque constraints against some well known computer algebra systems (Maple and Mathematica).

We propose to compare the effectiveness of several tools (including computer algebra system, several SMT solvers and a dynamic synthesis method) against such opaque conditionals, to evaluate their limitations and define areas for improvement.

1. MBA-based obfuscation

1.1. Mixed Boolean Arithmetic

Authors of [ZMGJ07] propose a general obfuscation method to hide information (secret constants, intermediate values, algorithms) in software, based on mixed mode computation over Boolean-arithmetic algebras (MBA) and on invertible polynomial functions over $Z/(2^n)$. Both the polynomial and its inverse must be of limited degree so that polynomial code transformations are efficient.

1.2. MBA constraints generation

Let us give an example of application of the method given in [ZMGJ07]. Consider the following conditional to be obfuscated:

```
if (IN == 0x87654321 )
        ..
else
        ..
```

Using the two linear MBA identities:

```
2y = -2*(x|(-y-1))-((-2x-1)|(-2y - 1)) - 3;
x + y =(x ^ y) - ((-2x - 1)|(-2y - 1)) - 1;
```

and one invertible polynomial transform:

```
F(x) = 727318528x*x + 3506639707*x + 6132886
```

The following opaque constraint is generated:

```
a = x*(x1 | 3749240069);
b = x*((-2*x1 - 1) | 3203512843);
d = ((235810187*x + 281909696 - x2) ^ (2424056794 +
x2));
e = ((3823346922*x + 3731147903 + 2*x2) | (3741821003 +
4294967294*x2));
f = 4159134852*e + 272908530*a + 409362795*x +
136454265*b+ 2284837645 + 415760384*a*a +
415760384*a*b+1247281152*a*x+ 2816475136*a*d
+1478492160*a*e+3325165568*b*b+2771124224*b*x+1408237568
*b*d+2886729728*b*e+4156686336*x*x
+4224712704*x*d+70254592*x*e+1428160512*d*d +
1438646272*d*e+1428160512*e*e+135832444*d;
if (IN == f )
        ..
else
        ..
```

The output value of `f` is always the constant `K = 0x87654321` regardless of values in `x`, `x1` and `x2` because:

```
F-1(4076439043*MBA1 + 859287276*MBA2 + F(K)) = F-1(F(K))
= K
```

2. Evaluation

We investigate in this section:

- the possibilities given by computer algebra systems to simplify the MBA-based opaque conditional described in section 1
- the behavior of several SMT solvers against randomly generated MBA-based opaque conditionals.
- the possibilities given by a hybrid dynamic-symbolic approach, especially crafted to simplify such obfuscated constraints.

2.1. Robustness against Computer Algebra Systems

Let us observe that even if computer algebra systems may encounter some problem to simplify the above formula, such a construction can be recognized, analyzed and simplified manually, by first factoring the resulting multivariate polynomial, thus identifying the polynomial `F`, then calculating F^{-1} and extracting `MBA1` and `MBA2` before finding back $F^{-1}(F(K)) = K$.

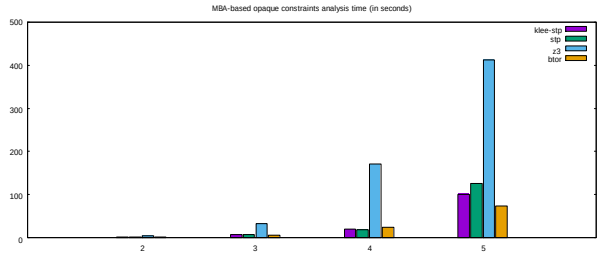
2.2. Robustness against SMT solvers

We use the LLVM [LA04] compilation framework to implement the MBA-based obfuscation transformations.

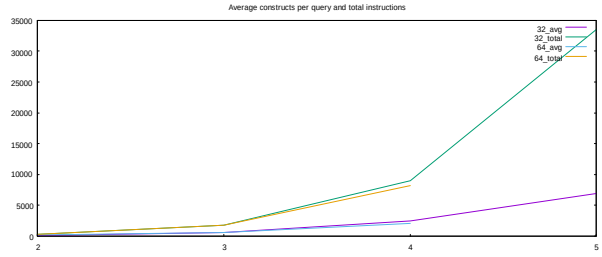
To evaluate the effectiveness of constraints solvers against MBA-based opaque conditionals, we use the KLEE symbolic execution engine and the multi-solver support in KLEE provided by the MetaSMT framework.

We use the STP, Z3 and Boolector SMT solvers for the quantifier-free theory of bit-vectors, in combination with the quantifier-free extensional theory of array. The logic QF_AUFBV extends QF_BV (quantifier-free fixed size bit-vectors) with arrays, arbitrary sorts and function symbols. Experiments are conducted on a Intel Core i5 1.60-2.30 GHz.

The following figure gives the time (in seconds) required to analyze randomly generated MBA-based opaque constraints, for MBA expressions of 16 variables and polynomials of degree 2, 3, 4 and 5.



The following figure gives the average number of constructions per (solver) query and the total LLVM instructions for MBA expressions of 16 variables and polynomials of degree 2, 3, 4 and 5 over integers of size 32 and 64 bits.



The number of coefficients in the Newton's multinomial formula explains the instruction number growth with the polynomial's degree[1].

2.3. Robustness against dynamic synthesis

Several methods have been applied to extract dynamically a compact representation from an opaque constraint, modeled as the combination of affine and non-linear layers. From this representation we can efficiently synthesize a compact formula, easier to analyze manually.

Several works take advantage of concrete tests to improve symbolic execution efficiency (this is called concolic execution, that is to say, both concrete and symbolic). However, our method has not (in our knowledge) been investigated against opaque conditionals and never used to drive concolic execution.

2.4. Preliminary results and future work

The approach described in section 2.1 is efficient against the MBA multinomial example given in section 1.2 but is in our opinion not generic enough.

From experiments given in section 2.2, we observe that the 3 solvers are not equally competitive against MBA-based opaque conditionals. We also observe that the complexity of the analysis grows with the complexity of the resulting formula (and its size!). Lastly, the size of the ring $Z/(2^n)$ has a low impact.

Symbolic execution appears to be an efficient angle of attack for these opaque constraints. However, we infer from these preliminary results that there is still room for improvement against such obfuscation transformation. It is planned to compensate the observed limitations of symbolic execution engines against opaque conditionals by the realization of a specialized solver, taking advantage from concrete executions of the target function.

The idea is to modify the usual concolic strategy to enable compact representations of the opaque constraints, more synthetic and easy to read. In addition, the simplified constraint may be possibly easier to solve. We make the conjecture that such an approach may be applied to a large class of opaque constraints (in addition of the MBA-based obfuscation transformation).

3. References

[Bru09] Robert Daniel Brummayer, "Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays", in *15th TACAS International Conference*, pp 22-29, 2009.

[CDE08] Cristian Cadar and Daniel Dunbar and Dawson Engler, "KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs", 2008.

[HFF+11] Finn Haedicke and Stefan Frehse and Görschwin Fey and Daniel Große and Rolf Drechsler, "metaSMT: Focus On Your Application Not On Solver Integration", 2011.

[LA04] C. Lattner and V. Adve, "LLVM: a compilation framework for lifelong program analysis & transformation", in *International Symposium on Code Generation and Optimization*, pp. 75--86, 2004.

[ZMGJ07] Yongxin Zhou and Alec Main and Yuan Xiang Gu and Harold Johnson, "Information Hiding in Software with Mixed Boolean-Arithmetic Transforms", 2007.

[JXZ08] Harold Johnson and Yuan Xiang Gu and Yongxin Zhou, "System and method of interlocking to protect software-mediated program and device behavior", US Patent, 2008.

[1] Complexity (d+m)!/(d!m!) is the number of distinct m-tuples of non-negative integers, whose sum is lesser or equal d, where d is the degree of the polynomial and m the number of bit-vectors. Here, m = 16 and d = 2,..,5.