

Automated Binary Analysis: Optimizing solver queries in BINSEC

Frédéric Recoules and Sébastien Bardin

CEA, List

Site de NanoInnov (Palaiseau, Plateau de Saclay)

`frederic.recoules@cea.fr` `sebastien.bardin@cea.fr`

`https://binsec.github.io/`

Context. Symbolic execution [3, 1] is a powerful technique for automated program analysis, bug detection, and vulnerability assessment. The open-source BINSEC [5, 8] platform, developed at CEA, leverages symbolic execution for binary-level analysis, enabling applications such as bug characterization [7], side-channel leakage detection [4], and fault injection evaluation [6]. However, symbolic execution suffers from two major scalability issues: path explosion and constraint bloating.

Source-level symbolic execution tools, most notably KLEE [2], mitigate query costs by using optimization techniques such as expression rewriting, counterexample caching, and constraint independence identification [2, 9]. Rather than querying the SMT solver on a full path predicate — where exact cache hits are extremely rare — KLEE slices path formulas into smaller, independent sub-formulas (connected components of dependent constraints). This enables fine-grained query result caching and reuse across distinct execution paths [9, 1].

The problem. At the binary level, this approach fails because program memory is modeled as a single monolithic array. Pervasive symbolic memory accesses create artificial dependencies that merge distinct sub-formulas into large equivalence classes (e.g., via union-find). This breaks formula independence and degrades caching. Restoring these optimizations requires partitioning the global memory array into disjoint regions.

Goal. The primary objective of this internship is to enhance SMT query efficiency in BINSEC by designing a memory-aware formula slicing and caching pipeline. Using abstract domain analyzers (e.g., memory region partitioning or value-set analysis), the intern will decompose monolithic memory constraints into independent sub-problems.

Specifically, the work involves: (1) designing a slicing algorithm tailored to BINSEC’s low-level intermediate representation, (2) integrating it into the

SMT solver pipeline, (3) developing an incremental caching framework inspired by KLEE to reuse partial models, and (4) evaluating the approach on a representative benchmark suite.

Expected Outcomes. A functional prototype integrated into BINSEC and an experimental evaluation quantifying performance gains in formula independence, SMT solving efficiency, and overall runtime.

Candidate profile. Experience with functional programming languages with a keen interest in algorithmic problem-solving and the formal aspects of computer science, including logic, automata theory and compilation/compiler techniques. Familiarity with binary analysis or reverse engineering is a plus.

Supervision. The internship will be co-supervised by Frédéric Recoules and Sébastien Bardin. There will be a regular weekly follow-up meeting, either on site at Nano-Innov (Plateau de Saclay) or remote.

References

- [1] Roberto Baldoni, Emilio Coppa, Daniele Cono D’Elia, Camil Demetrescu, and Irene Finocchi. A survey of symbolic execution techniques. *ACM Comput. Surv.*, 2018.
- [2] Cristian Cadar, Daniel Dunbar, and Dawson Engler. Klee: unassisted and automatic generation of high-coverage tests for complex systems programs. OSDI’08, 2008.
- [3] Cristian Cadar and Koushik Sen. Symbolic execution for software testing: Three decades later. *Commun. ACM*, 2013.
- [4] Lesly-Ann Daniel, Sébastien Bardin, and Tamara Rezk. Binsec/rel: Symbolic binary analyzer for security with applications to constant-time and secret-erasure. *ACM Trans. Priv. Secur.*, 2023.
- [5] Adel Djoudi and Sébastien Bardin. Binsec: Binary code analysis with low-level regions. In *TACAS*, 2015.
- [6] Soline Ducousso, Sébastien Bardin, and Marie-Laure Potet. Adversarial reachability for program-level security analysis. In *ESOP*, 2023.
- [7] Guillaume Girol, Benjamin Farinier, and Sébastien Bardin. Not all bugs are created equal, but robust reachability can tell the difference. In *CAV’21*.
- [8] CEA List. Binsec. <https://binsec.github.io/>.
- [9] Willem Visser, Jaco Geldenhuys, and Matthew B. Dwyer. Green: reducing, reusing and recycling constraints in program analysis. FSE ’12, 2012.