Binary Decompilation to LLVM IR

Sandeep Dasgupta* Vikram Adve†

August 25, 2016

Analyzing and optimizing programs from their executable has a long history of research pertaining to various applications including security vulnerability analysis, untrusted code analysis, malware analysis, program testing, and binary optimizations.

This work is a step towards the same broader objective by decompiling the input binary into an intermediate form (IR) of LLVM, which is a widely-used compiler infrastructure. The main challenge of the work involves extracting the variable (both scalar and aggregate) and type information from the input binary into a fully functional IR.

Current state-of-the-art static analysis tools (e.g. [1, 5, 8]) for binary analysis & decompilation operate on ad-hoc intermediate representations (IR) of the machine code and not available in public. The corresponding literature has many advanced techniques ([4], [6], [14]), but they are not open source either. There are some frameworks which recover LLVM IR from executable. S2E[12] and RevNIC [10] present a method for dynamically translating x86 to LLVM using QEMU[7]. As pointed out in [4], these methods convert blocks of code to LLVM on the fly which limits the application of LLVM analysis to only one block at a time. RevNIC [10] recovers an IR by merging the translated blocks, but the recovered IR is incomplete and is only valid for current execution; consequently, various whole program analysis will provide incomplete information. RevGen [11] includes a static disassembler to recover an IR for entire binary. However, the translated code retains all the assumptions of the original binary about the stack layout. They do not provide any methods for obtaining an abstract stack or promoting memory locations to symbols, which are essential for the application of several source-level analysis.

For our current work we have used a publicly available tool called McSema [2] which can convert x86 machine code to functional LLVM IR. Moreover this project is actively maintained, BSD3 licensed, and has extensive tests and documentation. One of the downside of McSema recovered IR is that the variable (scalar/aggregate) and type information is missing. The current work is about devising a scalable decompilation solution which can recover those.

As an initial step, we have developed tools [3] on top of Giri[15] for better debugging of McSema generated IR. Notables are "Source Mapper", which maps source (input binary) information to generated LLVM IR, and a "Backward Slicer" for McSema generated IR. Also we have identified optimization opportunities, like scalar replacement of aggregates, in the design of McSema generated IR to improve its quality.

Mcsema uses a big flat array to model the runtime process stack i.e. all the reads/writes made by a binary on its runtime stack are modeled into this array. The first step towards our goal is to identify variables in this array and promote them as separate symbols which requires deconstructing this global array, that Mcsema shares between all the procedure, into per procedure array which is used to model the stack frame of that procedure. Such stack deconstruction is important because doing symbol promotion right on the global array could be very conservative because an indirect write made by a different procedure may prevent symbol promotion in the current procedure. We have implemented a transformation pass which can

*Electronic address: sdasgup3@illinois.edu
†Electronic address: vadve@illinois.edu

do the stack deconstruction. We are planning to research on and implement various variable recovery and symbol promotion schemes as described in [4, 6].

The next step is going to be data-type recovery which aims at representing every symbol in the IR with a meaningful type instead of the generic types in the McSema recovered IR. The inferred types not only enables many sophisticated analysis (e.g. pointer analysis) but can also be used to rewrite optimized machine code for different architectures. The plan is to first develop a simple type inferencer based on external function calls and arithmetic operations; then a more sophisticated one using a polymorphic type inference algorithm [14].

References

[1] *Hex-Rays Decompiler*. `https://www.hex-rays.com/products/decompiler/index.shtml`.

[2] *Mcsema*. `https://github.com/trailofbits/mcsema`.

[3] *Source Mapper and Backward Slicer for Mcsema generated IR.* `https://github.com/sdasgup3/llvm-slicer`.

[4] K. Anand, M. Smithson, K. Elwazeer, A. Kotha, J. Gruen, N. Giles, and R. Barua, *A compiler-level intermediate representation based binary analysis and rewriting system*, in Proceedings of the 8th ACM European Conference on Computer Systems, EuroSys '13, New York, NY, USA, 2013, ACM, pp. 295–308.

[5] G. Balakrishnan, R. Gruian, T. Reps, and T. Teitelbaum, *Codesurfer/x86—a platform for analyzing x86 executables*, in Proceedings of the 14th International Conference on Compiler Construction, CC'05, Berlin, Heidelberg, 2005, Springer-Verlag, pp. 250–254.

[6] G. Balakrishnan and T. Reps, *DIVINE: Discovering Variables in Executables*, in Proceedings of the 8th International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI'07, Berlin, Heidelberg, 2007, Springer-Verlag, pp. 1–28.

[7] F. Bellard, *QEMU, a fast and portable dynamic translator*, in Proceedings of the Annual Conference on USENIX Annual Technical Conference, ATEC '05, Berkeley, CA, USA, 2005, USENIX Association, pp. 41–41.

[8] D. Brumley, I. Jager, T. Avgerinos, and E. J. Schwartz, *BAP: A Binary Analysis Platform*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 463–469.

[9] V. Chipounov and G. Candea, *Dynamically translating x86 to llvm using QEMU*, tech. rep., 2010.

[10] ——, *Reverse engineering of binary device drivers with RevNIC*, in Proceedings of the 5th European Conference on Computer Systems, EuroSys '10, New York, NY, USA, 2010, ACM, pp. 167–180.

[11] ——, *Enabling sophisticated analyses of x86 binaries with RevGen*, in 2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W), June 2011, pp. 211–216.

[12] V. Chipounov, V. Kuznetsov, and G. Candea, *S2e: A platform for in-vivo multi-path analysis of software systems*, in Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS XVI, New York, NY, USA, 2011, ACM, pp. 265–278.

[13] K. ElWazeer, K. Anand, A. Kotha, M. Smithson, and R. Barua, *Scalable variable and data type detection in a binary rewriter*, in Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '13, New York, NY, USA, 2013, ACM, pp. 51–60.

[14] M. Noonan, A. Loginov, and D. Cok, *Polymorphic type inference for machine code*, in Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '16, New York, NY, USA, 2016, ACM, pp. 27–41.

[15] S. K. Sahoo, J. Criswell, C. Geigle, and V. Adve, *Using likely invariants for automated software fault localization*, in Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '13, New York, NY, USA, 2013, ACM, pp. 139–152.