On the Self in Selfie (Invited Talk)

Christoph M. Kirsch
Department of Computer Sciences
University of Salzburg
Austria
ck@cs.uni-salzburg.at

Abstract

Selfie is a self-contained 64-bit, 10-KLOC implementation of (1) a self-compiling compiler written in a tiny subset of C called C* targeting a tiny subset of 64-bit RISC-V called RISC-U, (2) a self-executing RISC-U emulator, (3) a self-hosting hypervisor that virtualizes the emulated RISC-U machine, and (4) a prototypical symbolic execution engine that executes RISC-U symbolically. Selfie can compile, execute, and virtualize itself any number of times in a single invocation of the system given adequate resources. There is also a simple linker, disassembler, debugger, and profiler. C* supports only two data types, `uint64_t` and `uint64_t*`, and RISC-U features just 14 instructions, in particular for unsigned arithmetic only, which significantly simplifies reasoning about correctness. Selfie has originally been developed just for educational purposes but has by now become a research platform as well. We discuss how selfie leverages the synergy of integrating compiler, target machine, and hypervisor in one self-referential package while orthogonalizing bootstrapping, virtual and heap memory management, emulated and virtualized concurrency, and even replay debugging and symbolic execution.

CCS Concepts • **Applied computing** → **Education**; • **Software and its engineering** → *Compilers*; *Interpreters*; *Virtual machines*;

Keywords Self-Referentiality, Symbolic Execution

ACM Reference Format:
Christoph M. Kirsch. 2018. On the Self in Selfie (Invited Talk). In *Proceedings of the 10th ACM SIGPLAN International Workshop on Virtual Machines and Intermediate Languages (VMIL '18), November 4, 2018, Boston, MA, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3281287.3281288

1 Introduction

Selfie [2] is self-referential in three distinct ways. Selfie[1] compiles its own source code[2] into RISC-V code that compiles that source code again into the exact same RISC-V code. In fact, selfie can compile itself and then execute the generated code to compile itself again all in a single invocation of the system. Code generated by the selfie compiler not only runs on the selfie emulator but also on the spike emulator and the pk kernel which are part of the official RISC-V toolchain[3]. While self-compilation is standard, doing that with a system implemented in a single, self-contained file that does not include anything from other sources is not. Selfie only requires five C builtin procedures, discussed below, that appear to be close to the minimal external functionality needed. Since the selfie emulator is implemented in the same file as the selfie compiler and selfie compiles itself, the emulator is able to execute itself. This feature may at first appear to be rather academic but is in fact the logical prerequisite to system virtualization. The selfie emulator running on itself can be seen as an operating system kernel providing full virtualization of the machine on which it runs, just not through context switching and native execution of user code but through interpretation of that code. The question is if, upon executing user code, instead of interpreting that code, we can have the emulator ask the emulator on which it runs to execute that code on its behalf. The answer is yes! The result is a hypervisor that shares most of its implementation with the emulator and can even host all of selfie including itself. We discuss self-execution in more detail first, followed by self-compilation, followed by self-hosting. Symbolic execution with selfie is ongoing work discussed last.

2 Self-Execution

The selfie emulator implements a 64-bit RISC-V machine with 14 instructions (called the RISC-U instruction set) and 4GB of main memory. The implementation of main memory is done through on-demand paging onto physical memory whose size can be configured at boot time. The emulator can therefore run on itself as long as it touches strictly less memory than the physical memory available to the emulator on which it runs. However, the choice of implementation is

[1] http://selfie.cs.uni-salzburg.at
[2] https://github.com/cksystemsteaching/selfie
[3] https://riscv.org

orthogonal to the rest of the discussion here. Segmentation, for example, would work as well. What matters here is how the six system calls in selfie are implemented (`exit`, `open`, `read`, `write`, `brk`, `switch`). The first five are standard system calls whereas the `switch` call performs context switching on behalf of the selfie hypervisor and is only available in selfie. Thus the hypervisor only runs on the selfie emulator. There are essentially two implementation scenarios. System calls are executed either as part of the emulator as if they were machine instructions or as part of a kernel that runs on top of the emulator. Interestingly, the implementation of all but the `switch` call in selfie works either way. In fact, the first five system calls are implemented according to the official RISC-V calling convention so that the selfie emulator properly mimics the spike emulator and the pk kernel. As a result, selfie binaries (except for the hypervisor) run on both platforms. The first four system calls are mapped to the C builtin procedures with the same name whereas `brk` and `switch` have no dependencies. Heap allocation is done through `malloc` which is the only other C builtin procedure selfie depends on (in particular, there is no `free`) while the implementation of `malloc` in selfie maps to the `brk` system call. We have successfully bootstrapped selfie on recent versions of macOS, Linux, and Windows using gcc and clang without any platform-specific information.

3 Self-Compilation

The selfie compiler implements in C* a single-pass, recursive-descent parser for C* (inspired by the Oberon-0 compiler [4]) and targets the RISC-U instruction set supported by the selfie emulator. C* is a strict subset of C that features just five statements (assignment, while, if, return, procedure call), two data types (`uint64_t`, `uint64_t*`), the unary `*` operator as the only means to dereference memory, hence the name C*, three types of literals (integer, character, string), and five builtin procedures (`exit`, `open`, `read`, `write`, `malloc`). There are arithmetic and comparison operators for unsigned integers and pointers but no bitwise or Boolean operators. The compiler can compile and link multiple C* files in memory and then execute the generated code right away on the selfie emulator or generate RISC-V assembly as well as proper ELF binaries that can later be loaded back into memory. A selfie-generated binary contains wrapper code for the five builtin procedures that interfaces calls to these procedures with the previously described system calls. There is an interesting synergy here between the wrapper code and the system call implementations. They are in fact implemented next to each other in the source code. Similarly, the encoding of machine instructions in the backend of the compiler is implemented next to the decoding of the machine instructions in the frontend of the emulator. The fixed-point of self-compilation can be demonstrated in a single invocation of the system by having selfie compile itself and generate a binary from that, and then execute that binary to compile itself again and generate another binary from that equivalent to the first binary [1].

4 Self-Hosting

The selfie hypervisor implements a virtualized version of the RISC-V machine emulated by the selfie emulator. Its core design is inspired by microkernels [3] while most of its implementation is shared with the emulator, in particular, on-demand paging, exception handling, and the boot loader. The only difference in the implementation is not to interpret user code but to switch machine contexts so that the underlying emulator interprets the user code instead. In fact, selfie can even alternate between emulation through interpretation and virtualization through context switching at runtime. Moreover, the hypervisor can host all of selfie including itself any number of times given adequate resources. For this to work selfie needs to remember the parent-child relationship of hypervisors so that exceptions created in a machine context can properly be delegated to the hypervisor that created the context. For simplicity and more efficient paging, the selfie emulator caches machine contexts created by self-hosting hypervisors. An interesting yet simple exercise is to enhance the emulator such that it can create multiple machine contexts loaded with different binaries and then multiplexes their execution for concurrency. This corresponds to an operating system kernel that implements concurrent processes through interpretation of user code. With selfie the step towards a more realistic kernel that switches contexts instead is then immediate and works out of the box since the hypervisor does logically exactly the same as the emulator.

5 Conclusions

Selfie enables teaching the design and implementation of programming languages and runtime systems using a single, self-contained system that makes the underlying, fundamental but rather involved self-referentiality transparent. The simplicity and self-containment of selfie has recently inspired us to go beyond teaching and explore symbolic execution of RISC-U code from within the system. We have already enhanced the emulator with replay of code execution upon encountering runtime errors and are now working on a minimal symbolic execution engine in the spirit of the selfie.

Acknowledgments

This work has been supported by the National Research Network RiSE on Rigorous Systems Engineering (Austrian Science Fund (FWF) Grant S11411-N23) and a Google Research Grant. Selfie is joint work with many people. Ideas, code, and documentation have been contributed by A. Abyaneh, M. Aigner, S. Arming, C. Barthel, S. Bauer, T. Hütter, A. Kollert, M. Lippautz, C. Mayer, P. Mayer, C. Mösl, S. Oblasser, C. Poncelet, S. Seidl, A. Sokolova, and M. Widmoser.

References

[1] A.S. Abyaneh and C.M. Kirsch. 2018. You can program what you want but you cannot compute what you want. In *Edward A. Lee Festschrift (LNCS)*, Vol. 10760. Springer.

[2] C.M. Kirsch. 2017. Selfie and the Basics. In *Proc. ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!)*. ACM.

[3] J. Liedtke. 1996. Toward Real Microkernels. *Commun. ACM* 39, 9 (Sept. 1996), 70–77. https://doi.org/10.1145/234215.234473

[4] Niklaus Wirth. 1996. *Compiler Construction.* Addison Wesley.