Granary: Comprehensive Kernel Module Instrumentation

University of Toronto

Peter Goodman* Angela Demke Brown
{pag, demke}@cs.toronto.edu

Akshay Kumar* Ashvin Goel
{akshayk, ashvin }@eecg.toronto.edu

Kernel modules extend the functionality of operating systems (OSes). Modules are used to support new devices (e.g. network and graphics cards) and provide new features (e.g. file systems). The kernel and its modules execute in a complex and dynamic environment. Understanding how modules behave in and affect this environment is important. However, analyzing module behavior is challenging. Static analysis of module source code is difficult because of the tight interaction between modules and the kernel. Some modules, however, are only distributed in a binary format, which makes static analysis intractable.

We created Granary to address the challenges of module analysis. Granary is a framework that efficiently instruments *arbitrary*, binary Linux kernel modules. Granary uses dynamic binary translation to dynamically rewrite and comprehensively instrument kernel modules. Our extensive use of compile-time meta-programming enables efficient, dynamic analyses that are driven by static kernel type information.

While designing Granary, we identified four goals for practical module analysis: i) comprehensively analyze *all* modules; ii) impose no performance overheads on non-module kernel code; iii) require no changes to modules and minimal changes to the kernel, and; iv) be easily portable between different hardware and kernel versions. Prior research based on source code analysis and annotations [2] fails to meet goals (i) and (iii), while work based on special hardware features or virtualization [3] fails to meet goals (i) and (iv), and work based on whole-OS or -system instrumentation/emulation [1] fails to meet goal (ii).

Granary meets all four stated goals:

i) Granary is comprehensive because it controls and instruments the execution of all module code. Granary maintains control by ensuring that normal module code is never executed. Instead, only decoded and translated module code is executed. Translated module code contains instrumentation and always yields control back to Granary. All modules can be instrumented in this way because dynamic binary translation operates on binaries and does not depend on any hardware features.

ii) Kernel code runs without overhead because Granary relinquishes its control whenever an instrumented module executes kernel code. Granary implements a novel technique for re-gaining control when kernel code attempts to execute module code. Each time some instrumented module code invokes a kernel function, each of that function's arguments are *wrapped*. Argument wrappers are type- and function-specific, and ensure that potential module entry points (e.g. module function pointers) are replaced with behaviorally-equivalent values that first yield control to Granary.

iii) We have changed less than 100 LOC in the Linux kernel in order to support Granary.

iv) Granary's wrapping mechanism is portable across different kernel versions because the majority of wrappers are automatically generated by a GCC plugin and several meta-programs.

Granary is a work in progress. It works on multi-core processors with pre-emptive kernels, and incurs a modest decrease in throughput of 10% to 50% for network device drivers. We have used Granary to isolate and comprehensively instrument several network device drivers (e1000, e1000e, ixgbe, tg3) and file system modules (ext2, ext3). We have used Granary to develop an application which enforces partial control-flow integrity policies. These policies disallow modules from executing dangerous control-flow transfers. As a future work, we plan on implementing more optimizations and applications.

References

[1] P. Feiner, A. Demke-Brown, and A. Goel. Comprehensive Kernel Instrumentation via Dynamic Binary Translation. In *ASPLOS*. ACM, 2012.

[2] Y. Mao, H. Chen, D. Zhou, X. Wang, N. Zeldovich, and M. F. Kaashoek. Software fault isolation with api integrity and multi-principal modules. In T. Wobber and P. Druschel, editors, *SOSP*, pages 115–128. ACM, 2011. ISBN 978-1-4503-0977-6.

[3] D. T. Xi Xiong and P. Liu. Practical protection of kernel integrity for commodity os from untrusted extensions. San Diego, CA, USA, 2011. NDSS.

* Student