Remix: On-demand Live Randomization

Yue Chen, Zhi Wang, David Whalley, Long Lu

Abstract

Code randomization is an effective defense against code reuse attacks. It scrambles program code to prevent attackers from locating useful functions or gadgets. The key to secure code randomization is to achieve high entropy. A practical approach to boost entropy is on-demand live randomization that works on running processes. However, enabling live randomization is challenging since it often requires manual efforts to solve ambiguity in identifying function pointers.

We propose Remix, an efficient and practical live randomization system for both user applications and kernel modules. Remix randomly shuffles basic blocks within their respective functions. By doing so, it avoids the complexity of migrating stale function pointers, and allows mixing randomized and non-randomized code to strike a balance between performance and security. Remix randomizes a running process in two steps: it first randomly reorders its basic blocks, and then comprehensively migrates live pointers to basic blocks. Our experiments show that Remix can significantly increase the randomness with low performance overhead on CPU and I/O intensive benchmarks and kernel modules, even at very short randomization intervals.

Design

Remix shuffles basic blocks within their respective functions to increase run-time randomness. Specifically, Remix first parses the code into basic blocks, and generates a random ordering of these basic blocks to guide the process. Remix then lays out the basic blocks according to that ordering, and saves the mapping between their old and new positions in a table. This table is used to convert basic block pointers. The first instruction of a function (i.e., the function entry point) is replaced by a direct jump to the first basic block. As shown in Figure 1, some instructions like jump and PC-relative addressing , as well as all the basic block pointers, are updated to keep the program's original control flow .

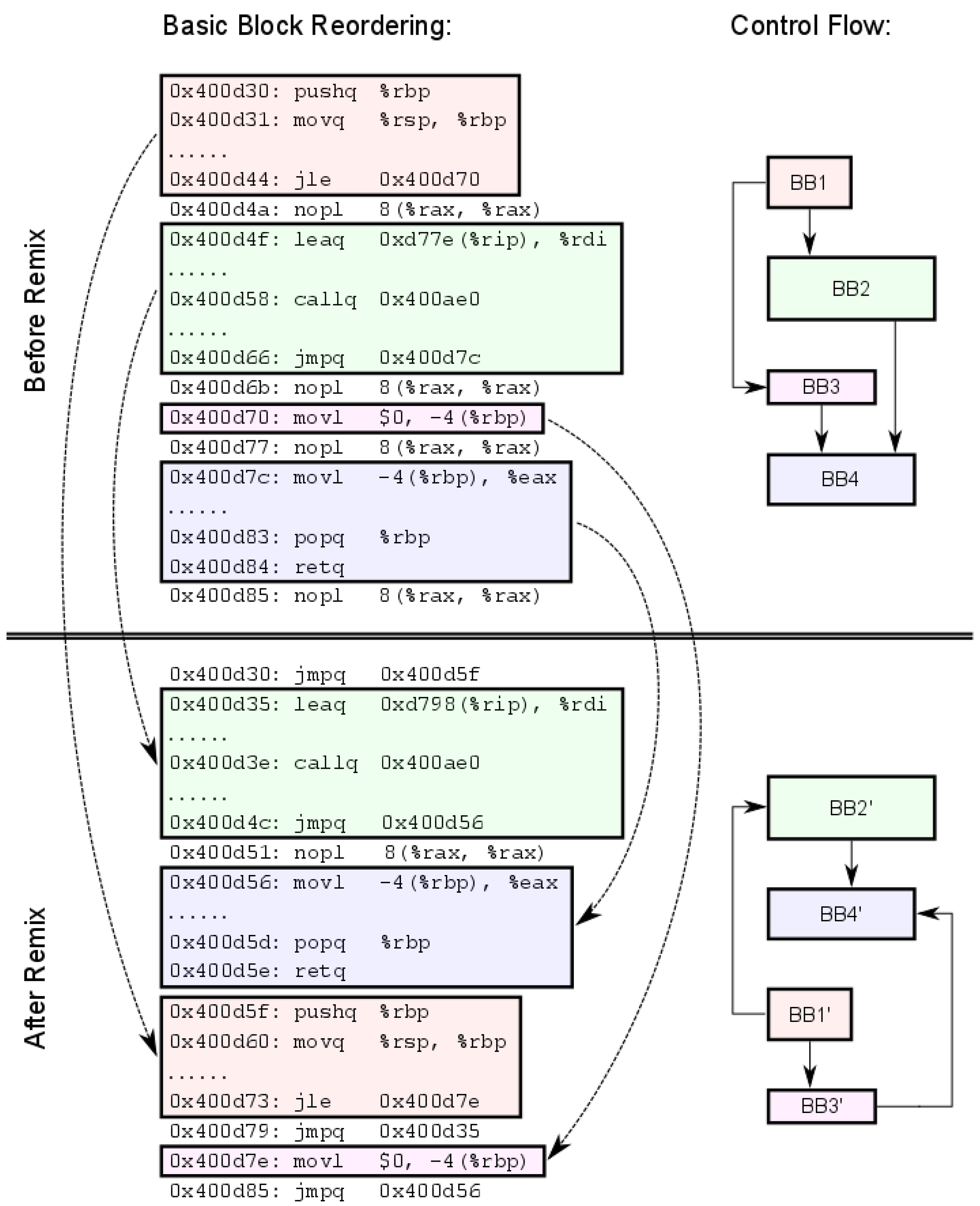

Figure 1: An Example of Remix on x86-64

Performance Overhead

The performance impact of Remix mostly comes from the following two aspects: first, live randomization has to stop the whole process or the kernel to ensure consistency. This introduces some latency to the whole process. Second, Remix rearranges the code layout. Modern computer architectures rely heavily on the cache for performance. Changing the process' code layout can affect its cache profile and by extension the performance. We measure both aspects with standard benchmarks (SPEC CPU2006) and a number of popular applications, with different re-randomization intervals.

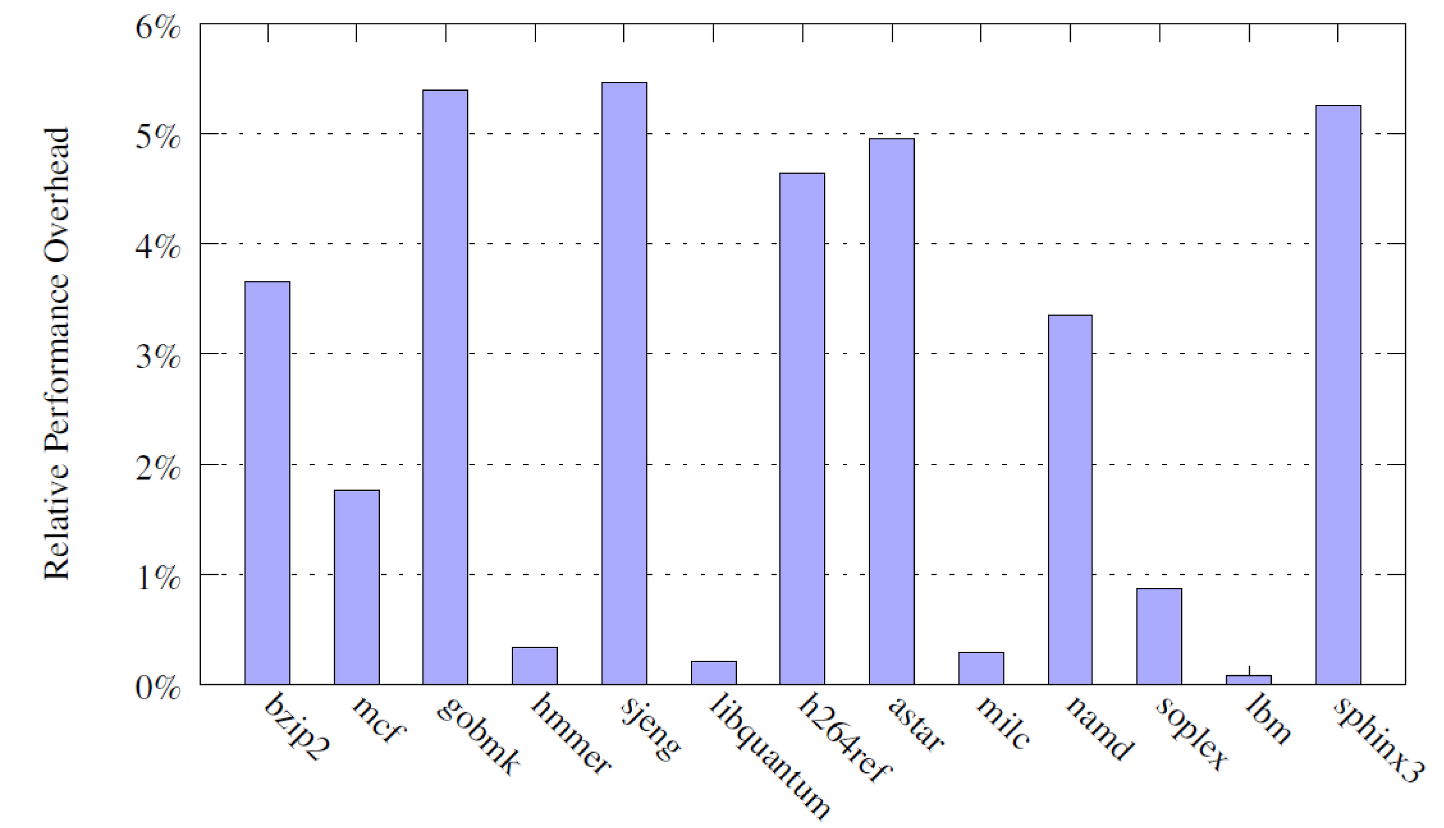

Figure 2: SPEC CPU2006 Performance Overhead

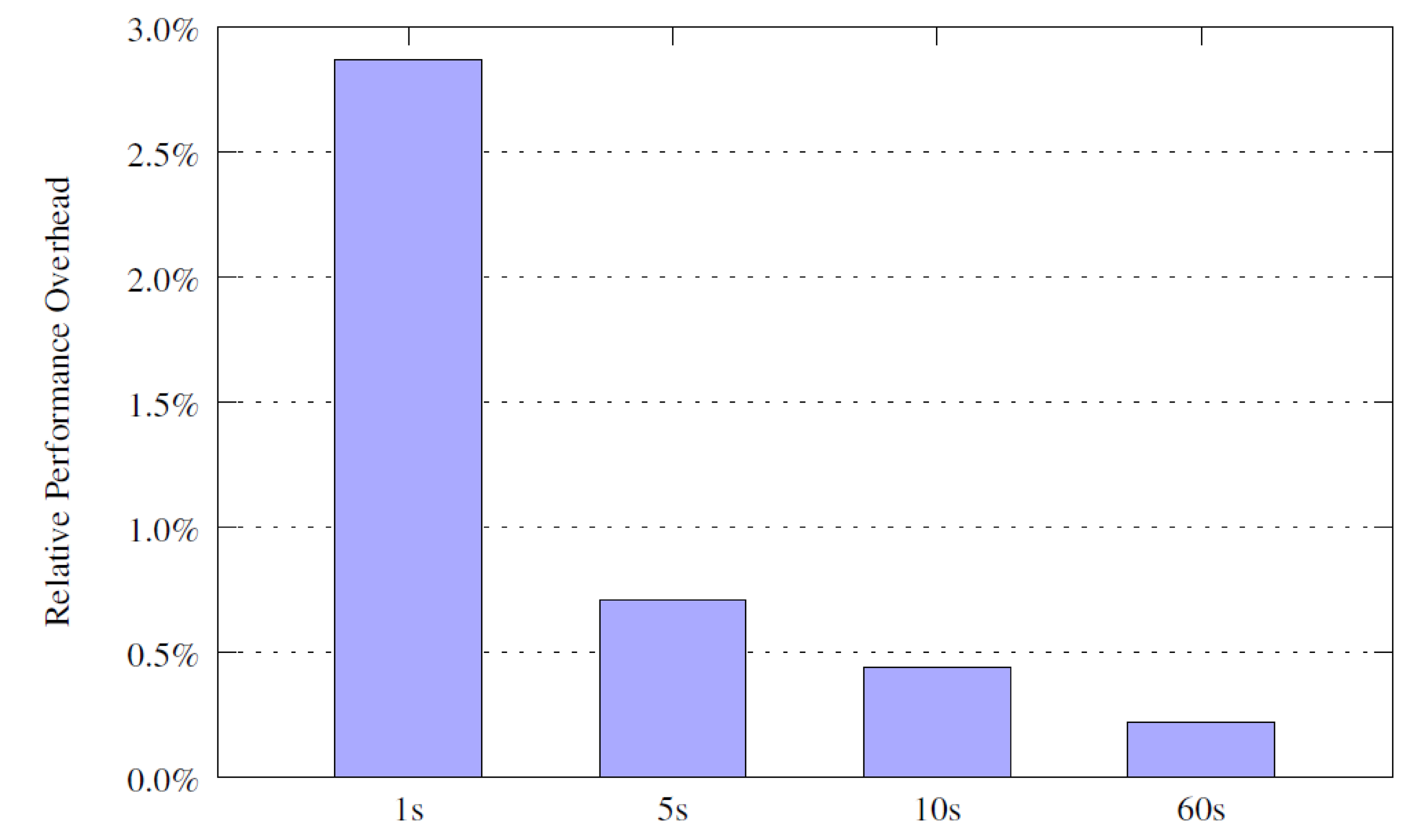

Figure 3: Apache Server Performance Overhead

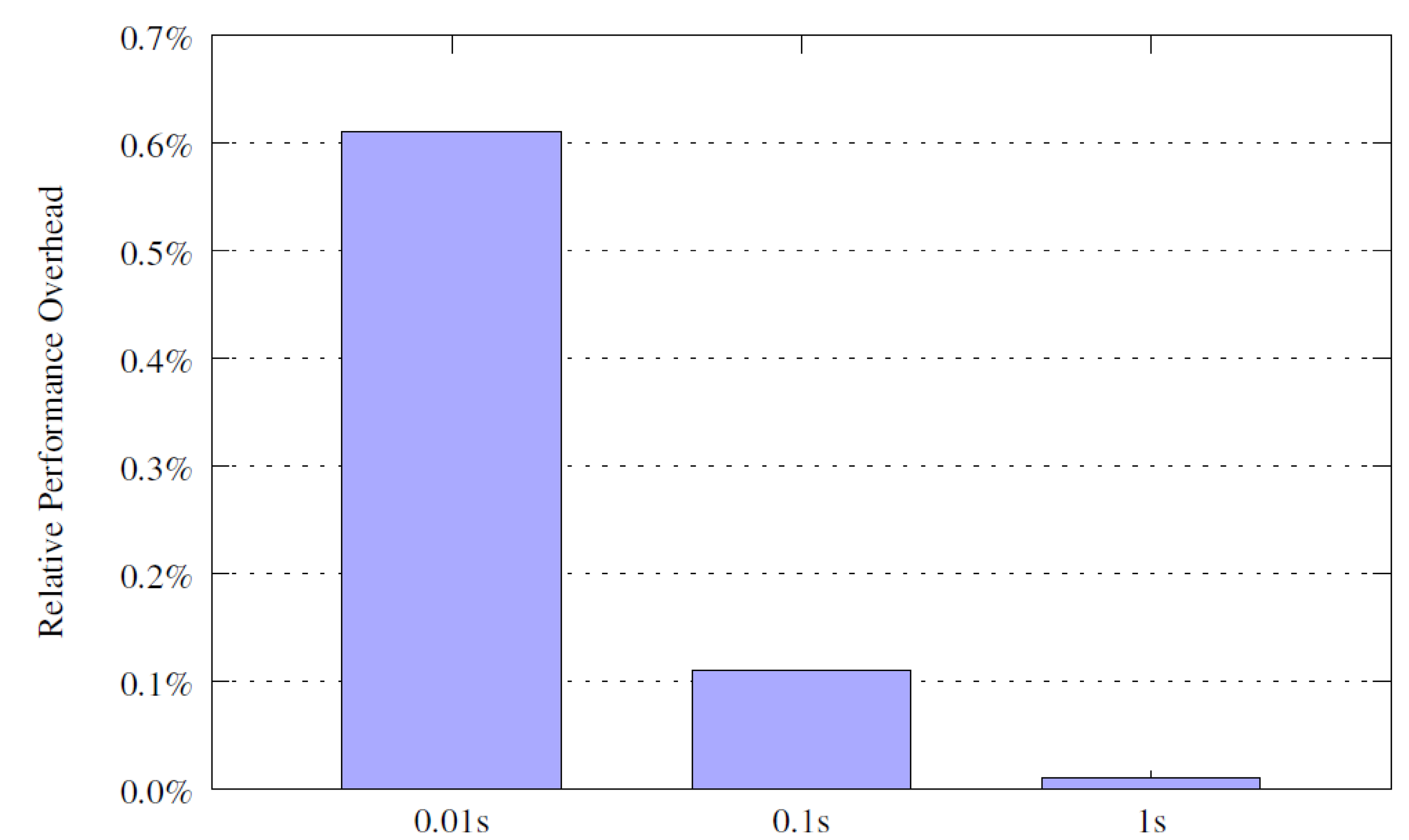

Figure 4: ReiserFS Performance Overhead