# Decoupling Kernel Address Space Randomization From the Linux Kernel

Ben Holmes
Professor Waterman
Computer Science Dept.

## Motivation

- In the interest of fast boot times for cloud computing, features have been bypassed
- Kernel Address Space Randomization (KASLR), a security feature, is one of the casualties
- Can features be implemented outside of the kernel without sacrificing performance?

## What Is Kernel Address Space Randomization (KASLR)?

- Move the kernel to a random location each time it boots to make it harder for attackers to find exploitable code
- Physical and virtual randomization
- Attacker only needs to guess or leak one offset

## What Is Function Granular KASLR (FG-KASLR)?

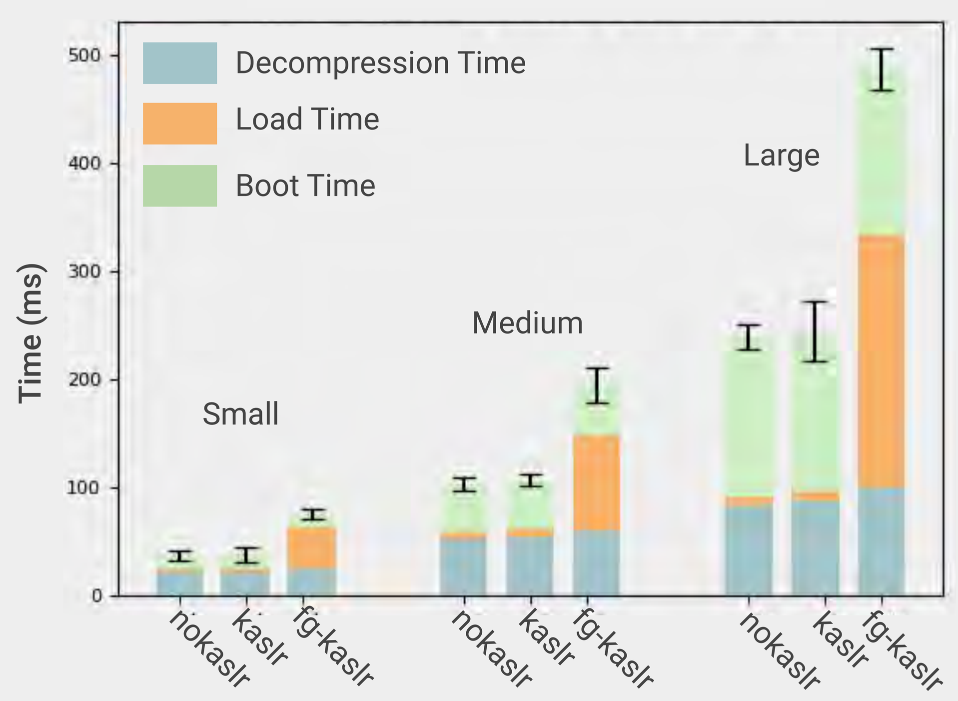
- Fine-grained approach to KASLR
- Move individual code segments to random locations, rather than the entire kernel
- Thousands of random offsets for an attacker to guess, rather than a single offset

## Experimental Setup

- Boot time varies with kernel size; chose small/medium/large kernels for data points
- QEMU to boot compressed kernels, track events with perf, a performance analyzing tool included with the Linux kernel
- Implementations of KASLR and FG-KASLR done in Firecracker, a Virtual Machine Monitor (VMM) written in the Rust language by Amazon Web Services

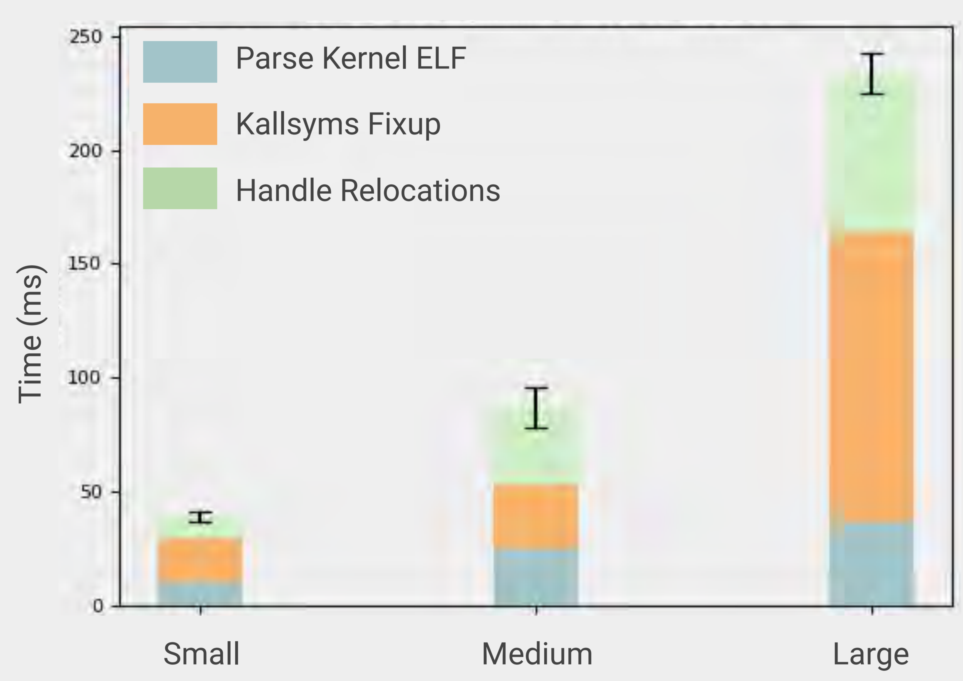
## Performance

Overall Kernel Boot Benchmarks




- KASLR exhibits low overhead
- KASLR and FG-KASLR increase kernel size, so decompression time increases
- FG-KASLR has significant overhead and occurs during load time

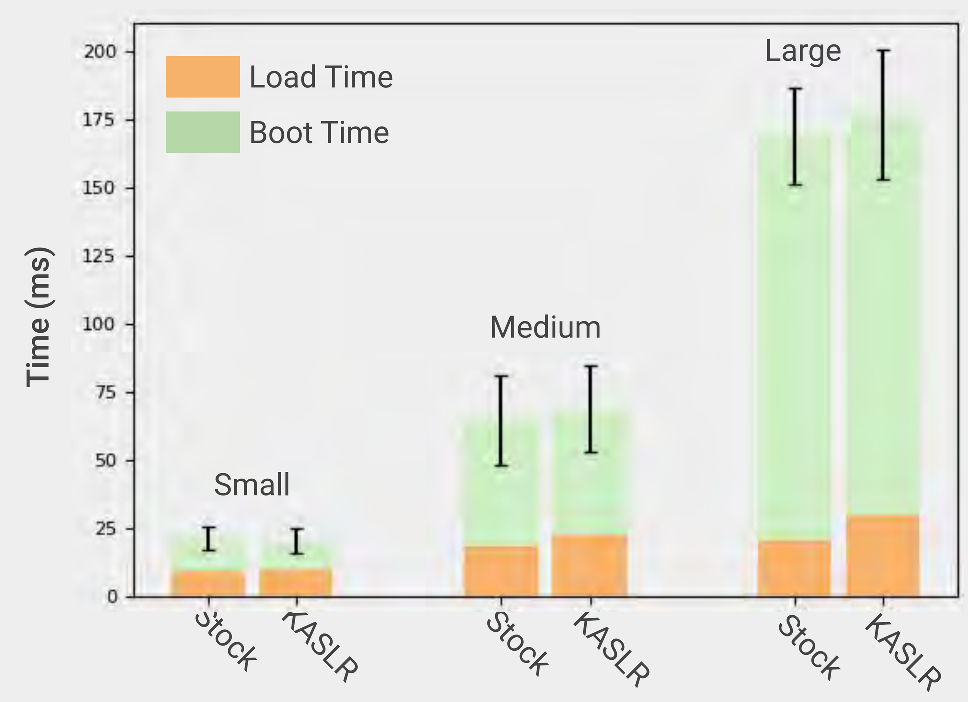
Fine-Grained View of FG-KASLR (Load Time)




- FG-KASLR time increases with kernel size
- Kallsyms Fixup is the majority of overhead due to sorting, but is unnecessary for booting to a shell

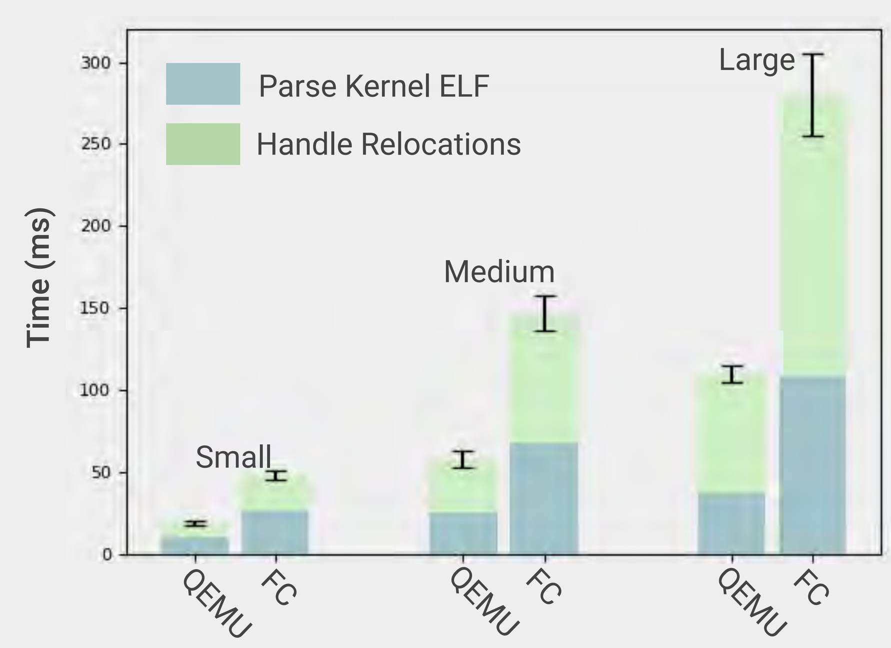
## Implementation

In-Monitor KASLR in Firecracker vs. Stock Firecracker



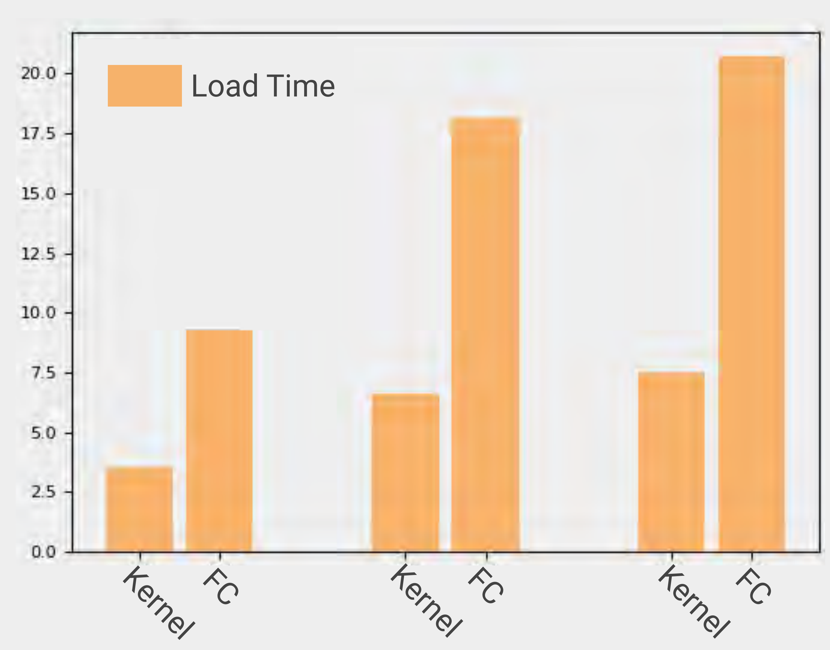

- In-monitor KASLR adds minimal overhead.
- Straightforward implementation
- Relocation entries must be passed to Firecracker
- Medium kernel is AWS configuration; well under target boot time

In-Monitor vs. Kernel Implementation of FG-KASLR




- Omitted kallsyms fixup due to large overhead cost; unnecessary for boot
- FG-KASLR more costly in-monitor than in the kernel

Kernel Bootstrap Loader vs. In-Monitor Bootstrap Loader Without KASLR




- Firecracker bootstrap loader significantly more overhead than Linux bootstrap loader due to protections in Rust not present in C

## Conclusions and Future Work

- KASLR can be implemented in-monitor with little complexity and overhead, allowing uncompressed kernels to have this first line of defense.
- FG-KASLR overhead pushes boot times well beyond the 150ms target set by AWS.
- In-monitor implementations of bootstrapping processes is the only way to decouple the bootstrap loader from the kernel.
- Allows the possibility of loading kernels into guest memory prior to boot, avoiding load time overhead when user requests kernel for boot.

## Acknowledgements

Joseph H. and Florence A. Roblee Foundation