Set the Configuration for the Heart of the OS: On the Practicality of Operating System Kernel Debloating

Hsuan-Chi Kuo

hckuo2@illinois.edu

University of Illinois at Urbana-Champaign
USA

Sibin Mohan

sibin@illinois.edu

University of Illinois at Urbana-Champaign
USA

Jianyan Chen

jianyan2@illinois.edu

University of Illinois at Urbana-Champaign
USA

Tianyin Xu

tyxu@illinois.edu

University of Illinois at Urbana-Champaign
USA

ABSTRACT

This paper presents a study on the practicality of operating system (OS) kernel debloating—reducing kernel code that is not needed by the target applications—in real-world systems. Despite their significant benefits regarding security (attack surface reduction) and performance (fast boot times and reduced memory footprints), the state-of-the-art OS kernel debloating techniques are seldom adopted in practice, especially in production systems. We identify the limitations of existing kernel debloating techniques that hinder their practical adoption, including both accidental and essential limitations. To understand these limitations, we build an advanced debloating framework named COZART which enables us to conduct a number of experiments on different types of OS kernels (including Linux and the L4 microkernel) with a wide variety of applications (including HTTPD, Memcached, MySQL, NGINX, PHP and Redis). Our experimental results reveal the challenges and opportunities towards making kernel debloating techniques practical for real-world systems. The main goal of this paper is to share these insights and our experiences to shed light on addressing the limitations of kernel debloating in future research and development efforts.

ACM Reference Format:

Hsuan-Chi Kuo, Jianyan Chen, Sibin Mohan, and Tianyin Xu. 2020. Set the Configuration for the Heart of the OS: On the Practicality of Operating System Kernel Debloating. In *ACM SIGMETRICS / International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '20 Abstracts)*, June 8–12, 2020, Boston, MA, USA. ACM, New York, NY, USA, 2 pages. <https://doi.org/10.1145/3393691.3394215>

1 EXTENDED ABSTRACT

Commodity operating systems (OSes), such as Linux and FreeBSD, have grown in complexity and size over the years. However, each application usually requires a small subset of OS features, rendering the OS kernel bloated. The bloated OS kernel leads to increased attack surface, prolonged boot time and increased memory usage. Application-oriented OS kernel debloating—*reducing the kernel code*

that is not needed by the target applications—is reported to be effective in mitigating many of the above issues. For example, Kurmus *et al.* report that 50%–85% of the attack surface can be reduced by debloating the Linux kernel for server software. Alharthi *et al.* show that 34%–74% of known security vulnerabilities can be nullified in the Linux kernel if it only includes kernel modules that are needed by the target application(s).

Debloating OS kernels by hand-picking kernel features is impractical. The complexity of commodity-off-the-shelf (COTS) OSes (e.g., Linux) makes manual debloating infeasible. Linux 4.14, for example, has more than 14,000 configuration options with hundreds of new configuration options introduced every year. Kernel configurators (such as KConfig) do not help debloat the kernel but only provide a user interface for selecting configuration options. Given the poor usability and incomplete and even incorrect documentation, it is prohibitively difficult for users to select the minimal kernel configuration manually.

Recently, several *automated* kernel debloating techniques and tools have been proposed and built. Despite their different implementations, existing kernel debloating techniques share the same working principle with the following three steps: (1) running the target application workloads, meanwhile tracing the kernel code that were executed during the application runs; (2) analyzing the traces and determining the kernel code needed by the target applications; (3) building a debloated kernel that only includes the code required by the applications.

However, despite their attractive benefits regarding security and performance, automated kernel debloating techniques are seldom adopted in practice, especially in production systems. This is certainly not due to a lack of demand—we observe that many cloud vendors (e.g., Amazon AWS, Microsoft Azure and Google Cloud) reduce the Linux kernel code in a handcrafted manner, which is not as effective as automated kernel debloating techniques. The goal of this paper [1] is to understand the limitations of kernel debloating techniques that hinder their practical adoption and to share our experience to shed light on addressing the limitations.

1.1 Accidental and Essential Limitations

We identify five major limitations of existing kernel debloating techniques.

- *No visibility of kernel boot phases.* Existing debloating techniques use `ftrace` that can only be initiated after the kernel

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

SIGMETRICS '20 Abstracts, June 8–12, 2020, Boston, MA, USA

© 2020 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-7985-4/20/06.

<https://doi.org/10.1145/3393691.3394215>

boot and hence cannot observe what kernel code is loaded during the boot phase. As a consequence, critical modules (e.g., disk drivers) might be missing, and the debloated kernel generated by existing techniques oftentimes cannot boot. Our experiments show that up to 79% of kernel features can only be captured by observing the boot phase. Even if the debloated kernel can boot, the kernel could miss certain performance and security features loaded at the boot time, leading to reduced performance and security.

- *Lack of support for fast application deployment.* Using existing tools, debloating for a new combination of applications or deploying a new application on a debloated kernel would require going through the entire three-step debloating process of tracing, analysis and building. The process is time-consuming (can take hours or even days) and thus prevents agile deployment—starting up a new application would have to wait until the generation of the debloated kernel.
- *Coarse-grained tracing.* Existing techniques use `ftrace`, which can only trace kernel code at the function level. However, function-level tracing is too coarse a granularity to track configuration options that affect intra-function code.
- *Difficulties in covering the complete kernel footprint.* Since kernel debloating uses dynamic tracing, it requires application workloads (typically encoded in benchmarks) to drive kernel code execution. However, it is challenging for benchmarks to cover the complete kernel footprint of the target applications, and the debloated kernel would crash if the application reaches kernel code that was not observed during tracing.
- *Not distinguishing what is executed versus what is needed.* Existing techniques treat any kernel features that are reached during the application workloads, even though the executed code may not be actually needed. For example, a second filesystem might be initialized but never needed.

To analyze the above limitations, we divide them into *essence* (the limitations inherent in the nature and assumptions of kernel debloating) and *accidents* (those that exist in current kernel debloating techniques but that are not inherent). We believe that the first three limitations are accidental and can be addressed by improving the design and implementation of the debloating techniques, while the last two limitations are essential that require efforts beyond the debloating techniques themselves.

1.2 Our Study and Results

This paper studies the limitations of existing OS kernel debloating techniques towards making OS kernel debloating practical, effective and fast. We focus on OS kernels deployed in cloud environments that are commonly used as guest OSes to run untrusted applications often with stringent performance requirements. To study the limitations, we carefully designed and implemented an advanced OS kernel debloating framework named COZART¹ built on top of QEMU. COZART employs a low-level tracing provided by QEMU to obtain visibility of the boot phase. COZART uses instruction-level tracing to track the kernel code and map the kernel code to kernel configuration options. COZART moves kernel tracing off the critical path: with COZART, one can generate CONFIGLETS (a set

of configuration options) specific to an application or a deployment environment *offline*. We term the former APPLETS and the latter BASELETS. Given a set of target applications, COZART can generate a debloated kernel by directly *composing* the APPLETS and the BASELET generated offline into a full kernel configuration. Such composability enables COZART to *incrementally* build new kernels by reusing CONFIGLETS (saving repetitive tracing efforts) and previous build files (e.g., the object files and LKMs).

We use COZART as a vehicle to conduct a number of experimental studies. Our studies lead to the following findings and results:

- Existing kernel debloating techniques initialize in-kernel tracing methods (e.g., `ftrace`) too late and cannot observe the boot phase, which is critical to producing a bootable kernel. COZART leverages the tracing feature provided by the hypervisor to support end-to-end observability (including the complete boot phase and application workloads) and produce stable kernels that can always boot and run applications stably. The debloated kernel by COZART does not have performance regressions (but slightly improves application performance) while achieving more than 13% reduction of boot time and more than 4MB memory savings.
- Kernel debloating can be done within tens of seconds if the configuration options of target applications are known. The composability of COZART allows users to prepare APPLETS offline. To deploy a new application on a debloated kernel, COZART composes the APPLLET of the target application with the CONFIGLETS of the current kernel to generate a new debloated kernel within tens of seconds.
- Using instruction-level tracing (instead of `ftrace`) can address kernel configuration options that control intra-function features. The overhead of instruction-level tracing is acceptable for running test suites and performance benchmarks, given that those can be done offline.
- An essential limitation of using dynamic-tracing based techniques for kernel debloating (which are the *de facto* approach) is the imperfect test suites and benchmarks. Specifically, the official test suites of many open-source applications have low code coverage. Combining different workloads (e.g., using both test suites and performance benchmarks) to drive the application could alleviate the limitation to a certain extent.
- Domain-specific information can be used to further debloat the kernel by removing the kernel modules that were executed in the baseline kernel but are not needed by the actual deployment. Take Xen and KVM as examples. We can use COZART to further reduce the kernel size by 40+% and 39+% based on the `xenconfig` and `kvmconfig` configuration templates provided by the Linux kernel source.
- Application-oriented kernel debloating can lead to further kernel code reduction for microkernels (e.g., L4) and extensively customized kernels (e.g., the Firecracker kernel). The reduction is significant: 47.0% for the L4 microkernel and 19.76% for the Firecracker kernel.

REFERENCES

- [1] Hsuan-Chi Kuo, Jianyan Chen, Sabin Mohan, and Tianyin Xu. Set the Configuration for the Heart of the OS: On the Practicality of Operating System Kernel Debloating. *Proc. ACM Meas. Anal. Comput. Syst.*, 4(1), March 2020.

¹COZART is publicly available at <https://github.com/hckuo/Cozart>.