Exploring Kernel Resilience: Fuzzing Linux Memory Management with Syzkaller

PHILIPP ENGLJÄHRINGER, ETH Zürich
MICHAEL KÜRSTEINER, ETH Zürich

This study investigates the security and reliability of the memory management (mm) subsystem and targets in particular the hugetlb part within the Linux kernel through the utilization of the Syzkaller fuzzer. The hugetlb subsystem, responsible for managing huge pages in memory, presents potential vulnerabilities due to its intricate nature and privileged access. Our research aims to comprehensively assess these vulnerabilities via systematic fuzz testing with Syzkaller, focusing on specific system call interfaces relevant to the hugetlb subsystem.

Our primary contributions include a curated corpus of system call descriptions tailored to the mm and hugetlb subsystems and enhanced input mutation/generation techniques. This corpus is able to cover more code of the targeted system in a shorter time span. Furthermore, we asses the shortcomings of the current approach when it comes to the detection of potential bugs of the targeted subsystem.

ACM Reference Format:
Philipp Engljähringer and Michael Kürsteiner. 2024. Exploring Kernel Resilience: Fuzzing Linux Memory Management with Syzkaller. *J. ACM* 37, 4, Article 111 (August 2024), 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Linux systems are integral to a vast array of computing environments, ranging from personal devices to enterprise servers. Central to the operation of these systems is the Linux kernel, an open-source monolithic operating system kernel that orchestrates the management of system resources and processes. However, the Linux kernel's exposure to diverse user inputs from contributors worldwide renders it susceptible to vulnerabilities, necessitating rigorous security assessments.

In this paper, we embark on the critical task of evaluating the security and reliability of specific subsystems within the Linux kernel, with a special emphasis on the memory management (mm) subsystem. Within the mm subsystem, we focus our attention on the hugetlb subsystem, which plays a pivotal role in managing huge pages—a feature crucial for optimizing memory utilization in various computing scenarios.

The hugetlb subsystem operates at a fundamental level, managing large memory pages that offer significant performance benefits for certain workloads, such as database management systems and high-performance computing applications. However, the complexity and privileged nature of the hugetlb subsystem introduce potential vulnerabilities that could be exploited by malicious actors, posing considerable security risks to the system as a whole.

Detailed inspection of the hugetlb code reveals that most of its vulnerability likely lies in problems concerned with concurrency and the complex management of memory regions of different sizes

Authors' addresses: Philipp Engljähringer, philipen@student.ethz.ch, ETH Zürich; Michael Kürsteiner, ETH Zürich, kumichae@student.ethz.ch.

ACM 0004-5411/2024/8-ART111
https://doi.org/XXXXXXX.XXXXXXX

in a system with a high demand for memory and limited resources. In summary, our study aims to reveal these concerns, by harnessing the power of the Syzkaller fuzzer, a renowned tool in the domain of kernel fuzzing.

2 RELATED WORK

Memory management is a cornerstone of operating system design, particularly in Linux, where the efficient allocation and utilization of physical and virtual memory are pivotal to system performance and stability. The insights gleaned from "Memory Management in Linux" by Mary Anitha Rajam[1] provide a valuable foundation for understanding the intricate mechanisms underlying memory management within the Linux kernel. Rajam's comprehensive examination of physical and virtual memory management mechanisms in Linux elucidates the sophisticated strategies employed to optimize memory allocation and utilization across diverse hardware architectures.

Moreover, our research focuses on the hugetlb subsystem within the Linux kernel, which plays a crucial role in managing huge pages for optimized memory utilization. The Huge Translation Lookaside Buffer (HugeTLB) allows memory to be managed in very large segments, reducing the probability of TLB misses and improving performance in applications with large memory requirements[2].

3 APPROACH

We focused on the kernel's allocation and management of memory for huge pages, utilizing system calls such as mmap, munmap, madvice, msync, etc., tailored for 2 MiB page sizes. To ensure that the system always makes use of a hugetlbfs, we mounted a corresponding filesystem in the initialization code. Our main goal was to trigger the BUG statements in the code related to the hugetlb system. These lines are mainly triggered by inspecting the parameters of the resv and subpool structs. Generally, these seem to be triggered when a race conditions arises. To focus execution orders that would cause such a condition, we wrote pseudosyscalls that open files on the hugetlbfs, read from them, write to them and remap them and we combined the standard syscalls as described above to further increase the variety of interactions. Our second goal was to trigger a bug caused by fragmentation, which in turn would cause invalid memory accesses that should be caught by the memory sanitizer KSAN. For this we forced the system to manage a large pool of normal sized pages in combination with huge pages.

4 IMPLEMENTATION AND EXPERIMENTAL SETTING

Outcome: Our goals were to fully cover certain BUG statements and increase the coverage of the targeted files. Despite our rigorous testing efforts, the results were somewhat anticlimactic.

We observed 214 non-reproducing crashes, primarily due to out-of-memory errors. Additionally, we identified two crashes marked as reproducible; however, the reports indicated errors such as "no output from test machine" and "WARNING in exec_debug," which were also non-reproducible.

Figures 1-4 compare the percentage of covered lines in each of the three targeted files and the mm subsystem in general. The blue points represent the coverage achieved by running the original syzkaller. Notably, the last point depicted reflects the value advertised on the Google Doc, which runs syzkaller indefinitely. The red points indicate the percentages achieved by running syzkaller with our extended syscalls and pseudo-syscalls.

We achieved a 1% increase in the coverage of hugetlb.c within 12 hours of fuzzing. The data indicates that not only did we achieve higher coverage for this file, but our coverage increased

[1]Memory Management in Linux
[2]HugeTLB

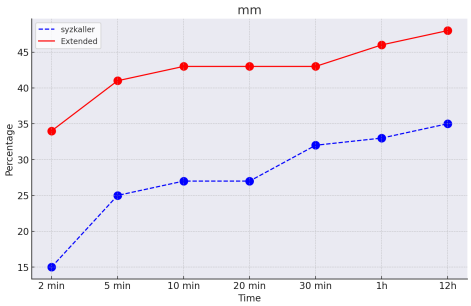

Fig. 1. Comparison mm

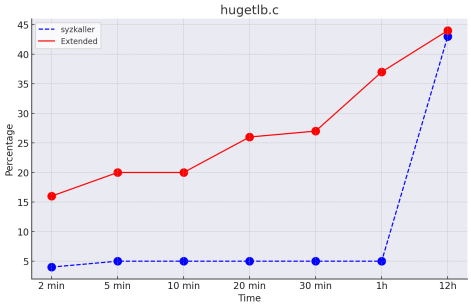

Fig. 2. Comparison hugetlb.c

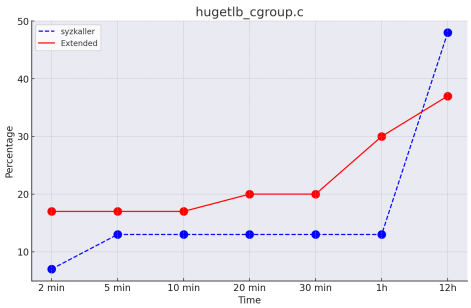

Fig. 3. Comparison hugetlb_cgroup.c

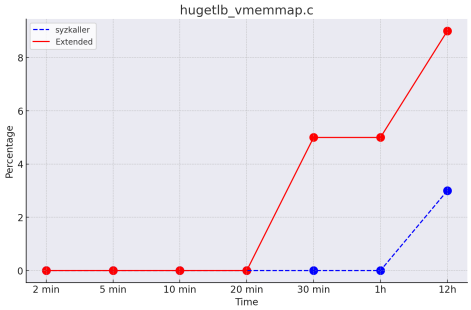

Fig. 4. hugetlb_vmemmap.c

much more rapidly compared to the current syzkaller. This improvement is likely due to our pseudo-syscall, which creates and mounts a directory where all files are mapped to hugepages as soon as the VM boots up. It also directly creates three files in this directory, ensuring that the VM always contains some files using hugepages, which interact with all the other implemented syscalls.

We increased the coverage of the hugetlb_vmemmap.c file from 3% to 9% within just 12 hours of fuzzing. This is most likely due to our pseudo-syscalls which continously create files in the mounted hugepage directory. This will ensure that new mapping to virtual memory get continously created. The pseudo-systemcalls that then write to those files ensure that those mappings get continously accessed. Since the hugetlb_vmemmap.c is responsible for handling those mappings, this was enough to achieve this increase in the coverage.

For the hugetlb_cgroup.c file, we did not achieve the same coverage as Google. However, since our coverage increased more rapidly than the original syzkaller, we believe that with more time or compute resources, we could at least match Google's coverage.

We reached several BUG statements in the hugetlb.c file but did not fully cover any of them. All these BUG statements performed sanity checks on the reservation maps. We attempted to forge invalid reservation maps through the interleaving of various syscalls and pseudo-syscalls that create, write, delete, alter, and re-map different files mapped to hugepages. Despite our rigorous but unsuccessful testing efforts, we conclude that to reliably and fully cover any of the BUG statements, more fine-grained access to the reservation mappings is required to artificially create invalid mappings. This has proven to be impossible using only system calls.

Experimental Setting. Our fuzzing approaches are conducted by Syzkaller (**commit c8349e48534ea6d8f01515335d95de8ebf5da8df**) in a VM emulated by QEMU (QEMU emulator version 6.2.0)[3] and configured according to the default, as given on the Syzkaller github [4]. Additionally, we enabled KCSAN in the kernel config to be able to better catch concurrency issues. The image is based on the Linux kernel (**commit 317c7bc0ef035d8ebfc3e55c5dde0566fd5fb171**). The entire kernel and syz-manager configuration files are part of the final artifact. Fuzzing was performed on 4 VMs in parallel each equipped with 2 GB of memory.

5 CONCLUSION

In this study, we explored the security and reliability of the Linux kernel's hugetlb subsystem using the Syzkaller fuzzer. Despite achieving some improvements in code coverage, particularly in the hugetlb.c and hugetlb_vmemmap.c files, our efforts highlighted the complexities and limitations of current fuzzing methodologies for deeply integrated and concurrency-prone subsystems like hugetlb. The primary challenges included handling race conditions on specific structs of the hugetlb code and fragmentation management as well as efforts in generating meaningful test cases to trigger specific vulnerabilities. Our work underscores the need for more sophisticated tools and techniques to enhance kernel fuzzing and improve the detection of subtle, concurrency-related bugs in critical system components. Another potential idea would be to spawn multiple users in the fuzzing loop with different rights. These users would then try to obtain access to unauthorised memory regions during runtime.

[3] QEMU - Documentation
[4] Syzkaller - VM Config