Kernel Concurrency Bugs

Concurrency bugs are notoriously difficult to test for as they may only be triggered under a small subset of all possible interleavings. Systematic approaches to finding concurrency bugs typically rely on the ability to control the scheduler to maximize coverage of the interleaving space. Such approaches cannot be directly applied to an OS kernel without extensive modifications, possibly rendering the results unrepresentative of the unmodified kernel. The problem is further complicated by the fact that kernel code frequently uses various finely tuned, architecture-specific mechanisms in order to minimize the performance impact of synchronization.

SKI is a recent approach for detecting kernel concurrency bugs that relies on virtual machine introspection and works on unmodified kernels.

The objective of this seminar report is to critically discuss the SKI approach and compare it to earlier approaches that attempt to address kernel concurrency bugs.

References:

Fonseca, Rodrigues, Brandenburg (2014):
SKI: Exposing Kernel Concurrency Bugs through Systematic Schedule Exploration.
**Page Table Isolation**

Page table isolation, first implemented for Linux under the name KAISER, is a security feature originally intended to harden the kernel against KASLR bypass attacks. In addition to this original purpose, page table isolation acts as an effective mitigation for the Meltdown vulnerability, leading to the implementation of similar mechanisms for other operating systems.

The objective of this seminar report is to give an overview of PTI and discuss the benefits and limitations of the approach as a mitigation for microarchitectural attacks or information leaks.

References:


**Verified Kernels**

Substantial research effort has been dedicated to the formal verification of operating system kernels. The most notable project in this space is the seL4 microkernel, but a variety of other projects targeting the verification of kernels or parts of kernels exist. One recent approach, Hyperkernel, is an attempt to design a Unix-like kernel in such a manner as to enable verification with a large degree of automation.

The objective of this seminar report is to give an overview of the Hyperkernel design and discuss how it differs from prior work, such as seL4, in terms of the approach and design as well as the verified properties.

References:

https://doi.org/10.1145/3132747.3132748

https://doi.org/10.1145/1629575.1629596
Secure Page Fusion

Modern operating systems commonly employ page fusion, a technique that allows the system to use the same physical frame for pages with identical content, thereby reducing memory pressure. While this approach can result in substantial reductions in memory usage, it may also introduce side channels and enable “memory massaging” (i.e., allow an attacker to manipulate the system to use a specific physical memory location for specific data, helpful for Rowhammer attacks).
Recent work has attempted to tackle these security issues while retaining the reductions in memory usage enabled by page fusion.

The objective of this seminar report is to give an overview of page fusion and the associated security issues and critically discuss VUsion as a possible solution to these issues.

References:

Oliverio, Razavi, Bos, Giuffrida (2017):
Secure Page Fusion with VUsion.
In: Proc. SOSP ’17.
https://doi.org/10.1145/3132747.3132781
Control Flow Integrity with Intel PT

Techniques to enforce control-flow integrity (CFI), which restricts indirect control transfers to prevent code-reuse attacks, are being researched in the community. While hardware-assisted approaches can have advantages in performance compared to software-only solutions, current hardware-assisted techniques are often incomplete or not efficient. The introduction of hardware features for control-flow logging (e.g., Intel Processor Trace aka Intel PT) provides new opportunities for hardware-assisted CFI enforcement approaches. While Intel PT was primarily designed for offline debugging and failure diagnosis, researchers have propose GRIFFIN, a system to provide online CFI enforcement over unmodified binaries using Intel PT. GRIFFIN has been implemented in the Linux 4.2 kernel and enables complete CFI enforcement. Experiments show that GRIFFIN is effective with a performance comparable to software-only techniques.

The objective of this seminar report is to explain basic CFI concepts and critically discuss the GRIFFIN approach compared to earlier approaches.

References:

Xinyang Ge, Weidong Cui, and Trent Jaeger:
GRIFFIN: Guarding Control Flows Using Intel Processor Trace.
In Proc. of ASPLOS 2017
Available online at https://dl.acm.org/citation.cfm?id=3037716
Protecting the OS from Devices

The close interaction between OS kernel and hardware devices can cause potential reliability and security issues. DMA is often considered a major threat as hardware devices are allowed to read and write directly from/to memory for performance reasons. DMA potentially allows a device to access critical kernel data as DMA-accessible buffers are collocated with kernel data (shared address space). Several approaches attempt to protect the kernel from device memory accesses by using an IOMMU. However, IOMMUs allow for page-level protection only rather than byte-level protection. Moreover, many approaches trade off security for performance thereby opening vulnerability windows. A recent approach (see references) makes use of a buffer copying approach that seems to perform better than other approaches while closing vulnerability windows.

The objective of this seminar report is to critically discuss approaches that rely on IOMMU techniques to protect the OS from malicious (or faulty) devices.

References:

The Portable Operating Systems Interface (POSIX) standard(s) has been around for more than 25 years. It is intended to help developers in achieving application portability across different UNIX-based (but also other) OSs. A central part of POSIX is the application programming interface (API) specification that defines a set of abstractions (functions, data types, etc.) which application developers can rely on for a different OSs (e.g., Linux, Android, macOS, BSD, Windows). Applications, including their use cases and requirements, have changed over the years, but the POSIX APIs have remained unchanged. But does POSIX still provide the “right” abstractions for modern systems? How do applications use the POSIX APIs nowadays? Are other abstractions needed? A recent study explores contemporary POSIX API usage on modern systems across different OSs (see references).

The goal of this seminar report is to critically discuss whether POSIX is still suitable for modern applications and OSs and compare the findings of the referenced study with other related works (consistency check).

References:

Securing OS Kernels by Path Restriction

Zero-day flaws may be triggered in Operating Systems (OSs). Even popular VM- or other isolation-based solutions may not provide enough protection (or are too performance expensive). The authors of the referenced paper propose a new security metric that shows strong correlation between “popular paths” and kernel vulnerabilities. The authors verify that the OS kernel paths accessed by popular applications in everyday use contain fewer security bugs than less-used paths. Based on that observation, they propose an approach to restrict an application to use only popular OS kernel paths. This can prevent the triggering of known zero-day kernel bugs better than other competing approaches.

The goal of this seminar report is to explain the basic concepts of the Lock-in-Pop approach and critically discuss it compared to similar approaches.

References:

Yiwen Li, Brendan Dolan-Gavitt, Sam Weber, and Justin Cappos:
Available online at https://www.usenix.org/conference/atc17/technical-sessions/presentation/li-yiwen