Innovative Operating System Elements
Seminar Course, SoSe 2019

Description
This seminar deals with the design, implementation, and application of innovative and state-of-the-art operating systems elements. We dive into the inner workings of modern operating systems and explore different system designs, from micro-kernel systems like seL4 over classic monolithic systems like Linux to systems somewhere in-between such as NetBSD. We also look into OSs for different requirements and application areas, from highly specialized OSs for safety critical embedded systems to commodity OSs running on server or desktop machines.

We explore recent scientific papers published in established software and systems conferences and other useful sources. Our focus is on the critical discussion of OS related issues and proposed approaches/solutions and the comparison of different approaches that tackle the same or a similar problem.

Note
The provided references lists, if any, for the individual topics are not exhaustive and should only be considered as a starting point for the literature research. The goal of the seminar, in general, is that students familiarize themselves with recent advances in engineering, validating, and hacking (in the broadest sense!) of operating systems. The suggested topics and accompanying references should be well understandable with background knowledge from the Operating Systems basic and RSWOS lectures.

Course Website
https://www.deeds.informatik.tu-darmstadt.de/teaching/sose2019/innos

Contact
Nicolas Coppik  nc@cs.tu-darmstadt.de  Tel: +49 6151 16-25227  Room: S2|02 E219
Oliver Schwahn  os@cs.tu-darmstadt.de  Tel: +49 6151 16-25227  Room: S2|02 E219
<table>
<thead>
<tr>
<th>Topic Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Flow Integrity with Intel PT ................................................................. 3</td>
</tr>
<tr>
<td>Securing OS Kernels by Path Restriction .............................................................. 4</td>
</tr>
<tr>
<td>Intra-Kernel Privilege Separation ........................................................................... 5</td>
</tr>
<tr>
<td>Protecting the OS from Devices .............................................................................. 6</td>
</tr>
<tr>
<td>Safe Linux Kernel Extensibility ............................................................................... 7</td>
</tr>
<tr>
<td>POSIX Abstractions in Modern OSs ........................................................................... 8</td>
</tr>
<tr>
<td>Kernel Concurrency Bugs ......................................................................................... 9</td>
</tr>
<tr>
<td>Page Table Isolation ............................................................................................... 10</td>
</tr>
<tr>
<td>Verified Kernels ...................................................................................................... 11</td>
</tr>
<tr>
<td>Secure Page Fusion .................................................................................................. 12</td>
</tr>
</tbody>
</table>
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 proposed 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
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
Intra-Kernel Privilege Separation

The monolithic design of most contemporary commodity operating systems is a threat to the secure and reliable operation of computer systems. Monolithic kernels consist of a large body of code that is impossible to prove correct or test exhaustively. Bugs in the kernel or kernel extensions such as device drivers can easily crash the system, being a reliability issue. Successful attacks on the kernel, e.g., using malicious device drivers, compromise security properties of the whole system. Privilege separation within the kernel, i.e., the ability to assign different parts of the kernel different privileges, can greatly improve the situation. For instance, the access to certain memory regions can be restricted within the kernel or subsystems within the kernel can be isolated from each other. A recent advancement in this area is the nested kernel architecture that aims to reduce a system’s trusted computing base by nesting a small isolated kernel within a commodity kernel.

The objective of this seminar report is to critically discuss the nested kernel approach and compare it to other approaches for privilege separation inside the kernel with a focus not only on security but also reliability and fault tolerance aspects.

References
Dautenhahn, N., Kasampalis, T., Dietz, W., Criswell, J., & Adve, V.
In Proc. of ASPLOS 2015
Available online at http://doi.acm.org/10.1145/2694344.2694386
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
Markuze, A., Morrison, A., & Tsafrir, D.
True IOMMU Protection from DMA Attacks: When Copy is Faster Than Zero Copy.
Available online at http://doi.org/10.1145/2872362.2872379
Safe Linux Kernel Extensibility

Extending kernel functionality in a safe way is an important design goal of many modern operating systems. OSs like Linux are structured for extensibility to support many platforms and environments from a unified code base. However, safety has remained a major issue as bugs in kernel extensions still cause problems and system breakdowns. The authors of the referenced HotOS paper investigate these issues and mitigation approaches proposed in the past. One of the findings is that there is a common set of typical errors that cause issues and that existing solutions are heavyweight and require high effort. Hence, they propose to develop kernel extensions in high-level, typesafe languages. They explore the use of Rust for that purpose and argue this to be an appealing approach.

The goal of this seminar report is to give a broader overview of the problems with Linux kernel extensions and discuss the current solution or mitigation approaches with a special focus on the Rust-based approach proposed in the referenced HotOS paper.

References
POSIX Abstractions in Modern OSs

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
Available online at http://doi.org/10.1145/2901318.2901350
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
Gruss, Lipp, Schwarz, Fellner, Maurice, Mangard (2017):
KASLR is Dead: Long Live KASLR.
https://link.springer.com/chapter/10.1007/978-3-319-62105-0_11

Lipp, Schwarz, Gruss, Prescher, Haas, Mangard, Kocher, Genkin, Yarom, Hamburg (2018):
Meltdown
**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**


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