A Review of Paper 1

znnnnnnn

We review "Harmonizing Performance and Isolation in Microkernels with Efficient Intra-kernel Isolation and Communication" by Jinyu Gu, Xinyue Wu, Wentai Li, Nian Liu, Zeyu Mi, Yubin Xia and Haibo Chen.

Summary

The authors observe that a major cost of a microkernel-based OS architecture, as opposed to a monolithic OS architecture, is that the former heavily employs IPC calls to communicate between OS components, while the latter typically uses simple function calls, which are much cheaper. They argue that only a few OS components are necessary in order to service most system calls, and so it is chiefly communication between *those* components that needs to be accelerated.

The authors speed up IPC between these components by moving them into the kernel's address space and executing them in privileged mode, where IPC with other systems in kernel space can be performed, ideally, with little more than a function call. This makes IPC much cheaper, since it no longer requires two or more address space switches, or an expensive `syscall` instruction.

However, this approach raises many obvious security concerns. The first major issue is that the components now operate in privileged mode, even though they are intended to run unprivileged. The authors address this by using virtualization hardware to trap and appropriately handle privileged accesses. Then, to keep kernel operations fast, they allow a small selection of privileged operations to be performed freely, and use binary rewriting to ensure that they are not performed by anyone but the kernel.

The other major security issue is that components moved into the kernel address space can access and modify each other's data. This is addressed via *memory protection keys* (MPKs), which allow the kernel to tag page table entries with any of 16 tags. Whether a page can be read from or written to depends on the tag of that page and the current value of the `PKRU` register. The kernel uses binary rewriting and the virtualization hardware to prevent components from changing the `PKRU` register.

Unfortunately, MPKs cannot protect code from malicious *execution* by another component. In order to protect the *IPC gates* (the sections of code that switch

execution between components, changing the `PKRU` register) from such execution, each IPC gate has a random token baked into it when it is created. The token is stored into a register before authentication, and read back and verified after the `PKRU` register is changed. Since no component knows this token, any attempts to jump past the authentication check will be caught.

This main contributions within this paper are the design of the aforementioned system, dubbed *UnderBridge*, an implementation of UnderBridge within three popular microkernels and a prototype named *ChCore*, and a performance evaluation of these implementations.

Strengths

- Reducing the cost of IPCs is a worthwhile problem.
- The general concept of tag-based memory isolation is old, but this method of bringing it to current architectures seems novel.
- The concept seems sound, and could significantly reduce the overhead of modular microkernel-based operating systems.
- They perform both microbenchmarks and macrobenchmarks.
- The necessary hardware appears to be widely available, and so the system can be implemented right now.
- The design has been integrated with existing microkernels.
- The paper makes good use of diagrams.

Weaknesses

- The paper's approach to security is very optimistic.
- Unlike much prior work, the optimization in this paper applies only for certain IPC calls.
- The benchmarks only test the performance of IPC, do not show absolute numbers, and are not reproducible.
- This approach may necessarily be vulnerable to Meltdown and Spectre.

Criticism

Limited Scope

The advantages of the optimization in the paper manifest only for communication within a set of up to 15 components. This means that for a microkernel-based OS to benefit significantly from UnderBridge, it needs to have a relatively high degree of centralization. Similarly, if the microkernel makes excessive use of privileged instructions, it will either run slowly due to constant VMExits, or it will require additional binary rewriting. Moreover, the processor must be an Intel processor, with support for both MPKs and hardware virtualization.

Although these limitations greatly restrict the scope of this paper, the authors successfully argue that they hold true for many popular systems.

Meltdown and Spectre

A mere paragraph is devoted to Meltdown and Spectre mitigations. The authors do not propose any solutions; address space layout randomization is necessary, but it does not constitute a full defense. The authors merely *imply* that their UnderBridge implementations currently lack the necessary mitigations, but it is unreasonable to compare UnderBridge with microkernels that incorporate kernel page table isolation solely to combat Meltdown attacks, without stating upfront that UnderBridge is susceptible to these attacks.

Moreover, the fact that UnderBridge is vulnerable to Meltdown and Spectre means that although it can *run* on contemporary machines, it also fails to provide useful security guarantees on those machines, which heavily dampens its appeal.

Unspecified Behavior

The security of their solution seems to depend on the fact that "MPK enforces permission checks on any user-accessible memory page". How do they know this? All we know is that they conducted a "detailed investigation". They do not discuss how they verified this or what platforms they verified this on, and in any case, it seems reckless to predicate the security of a system on unspecified hardware behavior.

Unauthorized Execution

The authors do not seem to address the execution of arbitrary code by a malicious component. Specifically, what happens when a component discovers the memory location of another component's code, and jumps to it? Given that the code would execute with the memory access privileges of the malicious component, this would not leak *much* information even if it worked, but it seems strange that this scenario was not addressed.

Trusted Computing Base

The authors state that their implementation increases the TCB by 1800 LOC, not including 1000–1500 LOC of modifications to seL4. It is debatable whether this is an acceptable increase to the TCB for the performance gains demonstrated, since the entirety of seL4 is only an order of magnitude greater.

More importantly, however, the system's isolation guarantees now depend on the the virtualization hardware, on unspecified behavior within the memory protection hardware, and on the binary rewriting implementation. Given how hard it is to trust these components, I believe that the authors did not adequately consider them in their Section 5 analysis of the TCB.

Migration and Harmonization

The paper could have explored the migration process in more detail. In particular, the word "harmonizing" in the title led me to believe that the paper would explore how to *dynamically* decide which components should be moved into the kernel's address space. Disappointingly, this idea is only mentioned in passing.

Benchmarking

Methodology

The authors list the hardware and operating system used, which provides context and aids reproducibility. However, they do not describe how they configured the kernels that they used.

The microbenchmarks consist of simply testing the round-trip latency of an IPC, and so should be easy to reproduce. Most of the macrobenchmarks, however, are neither standard nor published, and so are impossible to reproduce.

Moreover, the purpose of macrobenchmarks is to provide a *realistic* workload. By disabling the page cache, the authors predictably and drastically increase the number of system calls. They then use these results to claim an improvement of "up to 13.1×" in the paper's abstract, which seems deceptive.

Regressions

Both the microbenchmarks and the macrobenchmarks stress the IPC call mechanism, which we already know is improved by UnderBridge. The authors should have also conducted benchmarks to evaluate the degree to which other parts of the system have been slowed down. Of particular concern is the performance impact on CPU-heavy workloads, which do not perform any IPC calls, and on process creation, which requires binary rewriting.

Subsetting

The authors present only the YCSB-A results, claiming that the other workloads in the YCSB benchmark "give similar results". They also do not explain why the HTTP server benchmark was only run with Zircon, and they omitted many results from Table 3 claiming that they are "similar" to what is already shown. It is especially strange that the entries they kept were SkyBridge on Fiasco.OC, UnderBridge on ChCore, and seL4 unmodified, when they could have used the same microkernel for all three to draw a more meaningful comparison.

Relative Numbers

Almost every measurement in the application benchmark has been normalized. This makes the benchmarks much less useful, since we have no idea of the magnitude of the differences. This is compounded by the fact that the benchmarks

are not easily reproducible, and so we have no way of finding out how significant the improvements are.

Variability

No measure of variability (e.g. standard deviation, error bars) is given for any measurement. These benchmarks are likely to be fairly consistent between runs, but some confirmation is necessary.

Rust

There is a very brief mention of Rust, in which they mention that their design could be used to isolate `unsafe` code. Rust code usually uses `unsafe` because it interacts directly with the hardware, or for performance reasons. Isolating such code using runtime mechanisms with non-negligible cost seems counterproductive, so some elaboration from the authors would have been nice.

Generalizability

At the end of the paper, the authors list tagged memory features that are expected to be implemented in future architectures. It is reasonable to expect that the ideas in UnderBridge will carry over to these platforms, hopefully without the reliance on unspecified behavior.

Minor Issues

- There are a few typos, but the prose flows well enough.
- The section on related works should come at the start, to clearly establish what is novel and what is not.
- Regarding uses of Fiasco.OC and seL4 later in the paper, it could be made more explicit whether KPTI is enabled.
- In Figure 5, the label "-KPTI" makes it look like KPTI is *removed*, not *added*.
- In Table 1, the latter three rows should be doubled, to draw a clearer comparison with the first two rows.
- The paper assumes familiarity with x86-64. For example, it does not explain what the FS/GS or CR0 registers are. Depending on the audience, this may be an acceptable assumption.

Verdict

Although the benchmarks are suspect, and the particular method employed in the paper depends on unspecified behavior, the authors design and implement an interesting way to combine the performance characteristics of a monolithic

kernel and the isolation properties of a microkernel on current hardware, so I conclude that it is a good paper.