Research Statement

David Kohlbrenner

My research interests span software security, systems, and hardware architecture. My goal is to discover the ways that hardware design impacts software security and build hardware and software systems that are deserving of trust.

Computer science successfully builds itself on layers of abstraction from transistors to CPUs to software APIs. This layering is critical to our ability to build complex modern software systems like web browsers. However, the abstractions we rely on can be leaky and details of how hardware operations are performed can manifest in software execution. As a result, software security can unknowingly rest on incorrect assumptions about the hardware abstraction. In my research I find ways to leverage leaky hardware abstractions into novel software attacks and then build principled defenses for both software and hardware. These abstraction failures manifest as a new cache attack strategy, new side channel mechanisms, and even entirely new hardware failure modes such as Rowhammer. I want to build hardware and software systems that are capable of holding security guarantees on such shaky ground.

With my previous work I have proposed and prototyped architectural solutions for both software and hardware addressing the novel high-level security impacts of low-level hardware behaviors I’ve demonstrated. With my future work I expect to expand my previous defensive efforts against timing side channels to build a hardware and software stack we can prove properties of, as well as demonstrate and mitigate vulnerabilities leveraging new hardware deployment paradigms and features.

How hardware abstractions leak

The primary focus of my thesis work was finding assumptions software makes about hardware and demonstrating resulting security failures.

Floating-point timing attacks Instruction-data was proposed as a mechanism for timing attacks from the beginning [1], but until we demonstrated privacy breaking attacks on Mozilla Firefox [2] no such attacks had been demonstrated on general purpose Intel CPUs. This style of timing attack relies on differences in the number of cycles individual instructions take to complete, rather than the standard diverging code paths or shared state (ex: cache) changes. Our demonstrated attacks use timing differences arising solely from changing operand values to individual math instructions. These attacks leveraged performance variation between IEEE-754 floating-point value types in Intel and AMD processors’ floating-point units (FPUs).

Browser privacy and security is reliant on the core principle of the Same Origin Policy (SOP). SOP at a high level states that one web origin should not be able to unilaterally read information about another web origin. This is part of what stops a malicious web page from embedding resources from a banking website and reading out account information.

To break SOP and read data cross-origin, we repurposed the SVG-filter pixel-stealing timing attack model from Stone [3] to use our instruction-data timing channel. Specifically, we construct a scenario from an attacker-controlled web page via JavaScript and SVG filters where a series of floating-point divisions, multiplications and additions occur between a secret value (a pixel color value from another web origin) and an attacker controlled value. This was enough for us to build a timing side channel attack that steals usernames, login information, and web browsing history. While there had been a prior series of fixes to Mozilla Firefox to specifically address timing side channels in SVG filter implementations, they had not fully anticipated the threat posed by the individual arithmetic operations. Using the same underlying performance variations, we also presented a timing side channel attack against a research prototype of a database system specifically designed to keep data safe and prevent timing attacks.

Browsers adapted to the threat of new SVG filter timing attacks and patched in several different defensive solutions. As a follow-up I audited the new SVG filter implementations of, and found new vulnerabilities in, Mozilla Firefox, Google Chrome, and Apple Safari (CVE-2017-5407, CVE-2017-5107, and CVE-2017-7006 [1]) using variations on the previous Firefox vulnerability [4]. Several of these attacks bypassed the defenses specifically deployed to defeat floating point timing side channels. In response, all browsers deployed fixes based on our reports and communication with them. We are most excited about the changes made after communication with the Safari team: Safari deployed a major set of changes to how SVG filters are handled in the browser to completely remove the SVG filter pixel-stealing attack vector. This is the set of changes we had advocated for since the publication of the first series of vulnerabilities.

https://cseweb.ucsd.edu/~dkohlbre/floats
**Undermining SIMD-based defenses** Any defensive strategy proposed for the floating-point timing side channel must leverage detailed knowledge of the hardware’s behaviors. One promising solution, Escort [5], proposed using parallel instructions (SIMD) as a blinding mechanism for sensitive math operations. I examined the details of its implementation while auditing the browser timing side channel defenses [4]. To do this, I expanded a benchmarking toolkit I developed for measuring floating-point performance specifically focused on identifying value class performance variation. With these tools I was able to identify minute performance variations still present in Escort’s implementation as well as the probable microcode optimizations responsible. These variations were orders of magnitude smaller than the ones leveraged in our browser attacks, but were still consistently detectable under the right conditions. Unfortunately, the lack of introspection available to security researchers into microcode implementations of Intel CPUs stops us from being able to repair this defense. With our measurement toolkit, I also re-measured a variety of floating-point performance measurements from our previous work and found that far more floating-point values than we previously appreciated exhibit complex timing characteristics in most commodity x86 processors.

**TSX cache attack** When a vendor introduces a new processor feature, it always has the potential to enable new avenues of attack or defense. Intel released the Transactional Synchronization eXtensions (TSX) as a hardware implementation of transactional memory in recent processor generations as a performance tool. We examined the uses and failure cases it exposed and found that we can abuse TSX to run an efficient and timer-free cache attack we called Prime+Abort [6]. In a telling coincidence, at the same conference we presented Prime+Abort an independent group presented a cache timing-attack defense using the same TSX mechanisms. To make our attack work we put a specific set of cache lines into the read or write set of a memory transaction, at which point Prime+Abort gets a callback whenever activity disturbs those specific cache lines. This enables extremely high-speed detection of cache activity by another process, as well as bypassing all clock and timing related defensive schemes. We found that Prime+Abort is faster and has fewer false positives than similar Prime+Probe attacks.

Understanding the exact details of how processor features or performance trade-offs are made can enable us to both develop new attacks and to develop new defenses. My research works to understand these details and ensure that defenses new and old benefit or adapt. Collaboration with hardware architecture researchers, such as on the TSX attack, is a critical part of developing that understanding.

**Defensive paradigms for software and hardware** Motivated by the endless march of new attack techniques and side channel mechanisms, I focus my defensive efforts on verifiable and architectural methods. I believe we must develop systems that are inherently resistant to side channel attacks in the same way that broad mitigations against control-flow hijack attacks have been developed and deployed. In my research I have worked on several defensive efforts designed to mitigate hardware-based side channel attacks.

**libftfp** As part of our work on SVG timing attacks [2] we developed a fixed-point constant-time math library called libfixedtimefixedpoint(libftfp) as a potentially safe alternative for non-integer computation. This library is written in C carefully designed to coerce most compilers into producing constant-time code. It supports customizable partitioning of its 64-bit internal type at compile time and implements fixed-point versions of most SSE floating-point operations. libftfp was proven independently to be constant time when compiled to LLVM bytecode [7].

Constructing this library served two purposes: to make available a non-integer constant-time math library, and to demonstrate the impossibility of assured constant-time behaviors. We cannot confidently state that every processor available will handle the limited instructions we use in constant time. This is because we cannot get a guarantee from Intel that a given instruction runs in a data independent number of cycles. We can only rely on using instructions that our measurements indicate are probably constant time. This was a strong motivator for both our later work and for my future research. My interest in developing provably constant-time hardware and software combinations was a direct result of this project. libftfp is available on github [3] for research or other purposes.

**Fermata and Fuzzyfox** Inspired by timing attacks on web browser privacy by ourselves and others we proposed the Fermata browser [8] and developed a prototype called Fuzzyfox. We believe that many of the origin isolation and timing attack challenges we face today in browsers mirror the challenges tackled in

---

2https://github.com/dkohlbre/qtt
3https://github.com/kmowery/libfixedtimefixedpoint
multi-level secure operating systems of the late 1980s and early 1990s. Fermata is a proposal for application of the Fuzzy Time IO concept from such previous systems to web browser architectures as a systemic and provable mitigation to timing attacks. Fermata can guarantee (as Fuzzy Time IO does) that any timing attack an attacker finds has a bounded channel bandwidth in bits-per-second. Fuzzy time accomplishes this by granularizing time and making all events (including clock updates) arrive in batches with a randomized delay. This defeats the attacker’s ability to measure time at fine granularities using any clock method.

As a prototype of the Fermata proposal I developed Fuzzyfox; a fork of Mozilla Firefox. Fuzzyfox cannot obtain the formal guarantees of Fermata, but implements the core non-breaking changes proposed. We found that many previously developed attacks and timing techniques against current browsers were no longer usable against Fuzzyfox. All previous techniques we had developed were now correctly capped in their information leakage rate. Fuzzyfox is available on github and has been used by several other security research groups.

Future research

I see two connected thrusts to my future work: identify new software vulnerabilities enabled by leaky abstractions of hardware, and then use my understanding of the underlying causes to design software and hardware architectures that are provably resistant to entire attack classes.

Hardware reliability and security assumptions Rowhammer is the most prominent example of hardware reliability failures being triggered from software leading to security failures. I believe there are many such opportunities to leverage hardware reliability problems – known and unknown – into high-level software attacks. These attacks are not confined to commodity processors or DRAM; any component that is widely available (or soon to be) is a potential target.

My previous work has identified places where specific performance characteristics or details of feature implementation can leak to undermine high-level software security guarantees. We have processors and platforms today that integrate more components and performance features than ever before, all of which need to be considered as a potential security benefit or risk. Collaboration with hardware architects, both academic and industrial, to watch the ‘fire-hose’ of hardware changes will help mitigate potential vulnerabilities and formalize defensive opportunities.

As an example of such a target, FPGAs are seeing rapid deployment into consumer devices and cloud providers such as AWS. I believe that there are both new and old attacks enabled by this proliferation of customizable and effectively analog hardware. With commodity processors from Intel shipping on-chip FPGAs entire classes of side channel attacks have the potential to become fully remote software attacks. We need to identify what the new threats posed in this scenario are to develop an adequate defensive strategy before abuse occurs.

Designing resistant hardware and software I plan to continue work in line with Fermata, Fuzzyfox, andlibftfp further than the browser. A clear starting point is building hardware and software systems that together have provable performance and timing properties. This will require developing prototype hardware platforms using new tools like Clepsydra to prove timing behaviors about ALUs/FPUs and software stacks that leverage existing tools for proving timing properties. The ability to prove timing characteristics of hardware will allow the existing software tools make fully justified assumptions about the underlying hardware implementations of operations. As compared to the current state, the best that could be achieved was to make whitelists of believed ‘safe’ operations. I believe RISCV is a prime target for this type of platform as it has widespread compiler support, mainline Linux kernel support, and has open implementations of many different versions of major components.

https://github.com/dkohlbre/gecko-dev/tree/fuzzyfox-rebase
References


