0.1 Abstract

GrapheneOS responded to my first paper. Some of it was fair. I oversold isolatedProcess. Called it "the strongest sandbox" when it's just the standard Android option. Didn't mention V8 sandbox. Or Oilpan. Or CFI. Made the comparison look one-sided.

The core point still works. Chrome is built for containment. Sandboxes, site isolation, MTE, CFI. Firefox is built for prevention. Rust rewrites, RLBox, smaller API surface. Different approaches to the same problem.

I also missed things I should have caught. V8 has a memory sandbox that locks JIT code into a reserved region. SpiderMonkey has no CFI.

Fixed now.

0.2 1. Introduction

I wrote the first paper because GrapheneOS commented on desktop Firefox. Said Firefox's sandbox is substantially weaker. On Windows Firefox is at sandbox parity with Chrome. That's verifiable. The Android gap is real but it's not the whole picture.

GrapheneOS responded on Reddit and Discord. Some criticism was fair. I made mistakes. Oversold isolatedProcess. Left out several Chromium mitigations. The comparison was incomplete.

Some of it was less fair. Calling the paper dishonest and unethical for getting technical details wrong. Those are different things.

I'm keeping them separate. Technical corrections in the next sections. The personal stuff later.

0.3 2. What I Still Stand By

Most of the original paper held up. Here's what survived review.

0.3.1 2.1 Firefox's Rust Bet Is Paying Off

Firefox has rewritten major subsystems in Rust. CSS engine (Stylo). Renderer (WebRender). Library sandboxing (RLBox). Parts of the networking stack. Rust eliminates whole bug classes at compile time. Use-after-free. Buffer overflows. Null pointer dereferences. Chromium has mitigations for these and they're good mitigations. But mitigations manage symptoms. Rust eliminates the root cause.

Roughly 70% of Chromium's critical-severity bugs are memory safety issues [14]. Firefox's Rust components have produced zero such CVEs since they shipped [15]. That tracks with what the language guarantees.

The data backs up both approaches. Firefox aims to prevent bugs altogether. The Rust rewrite of CSS, rendering, and library sandboxing has produced zero memory safety CVEs so far. Chromium takes a different path. CFI, PartitionAlloc, V8 sandbox make bugs harder to weaponize even when they exist.

0.3.2 2.2 Content Blocking != Privacy Theater

GrapheneOS called extension-based content blocking "privacy theater." Equated it with antivirus scanning. That analogy doesn't hold.

Blocking a request at the network layer before the payload reaches your device is structurally different from scanning a file after it's on disk. uBlock Origin stops exploits from being delivered before they reach the renderer.

Filter lists can't block novel zero-days. That's a real limitation. Zero-day delivery in practice relies on known malicious infrastructure. Compromised ad networks. C2 domains. Exploit kit landing pages.

Filter lists block those. The "enumeration is futile" argument assumes attackers spin up fresh infrastructure for every target at zero cost. Not how mass-market exploitation works.

0.3.3 2.3 Monoculture Risk Is Real

Chromium's mobile market share is overwhelming. When the Chromium engine has a critical vuln, billions of devices are affected at once. That concentration of value creates attacker incentives that don't exist for Firefox's minority codebase.

A Chromium zero-day is worth more to an exploit broker than a Firefox zero-day. Purely because of market share. Firefox being ignored by attackers is itself a security property. Not a flattering one but it's real.

The dual-engine criticism keeps coming up. "Firefox adds a second engine to your device." Android mandates a system WebView. On GrapheneOS that WebView is Vanadium regardless of your browser choice. Running Firefox on top of that adds marginal attack surface. It also reduces monoculture risk. Different users weigh that differently.

0.3.4 2.4 Pick Your Threat Model

The original paper's core claim held up. Browser choice is a threat-model alignment. Not a binary "secure vs insecure" judgment.

Targeted state sponsored attacks. Chrome/Chromium makes sense there. Post-compromise containment is the priority and isolatedProcess delivers that. Mass surveillance and drive-by exploits targeting the Chromium monoculture. Firefox makes more sense there. A renderer that's harder to compromise reduces the probability that any exploit succeeds. Tracking and fingerprinting as primary concern. Firefox's Total Cookie Protection, ETP, and container isolation are ahead of anything in the Chromium ecosystem.

Depends on what you're up against.

0.3.5 2.5 Android vs. Desktop Matters

I keep coming back to this because it's where the debate gets muddled. The sandbox gap is almost entirely Android-specific.

On Windows, Firefox's sandbox is at parity with Chrome's. Both use AppContainer for kernel-level process isolation [13]. Both restrict Win32k syscalls from renderer processes [9]. Both use broker process models for privileged operations.

The isolatedProcess thing is an Android limitation. Android apps can't create namespaces or use seccomp-bpf. The Android Runtime needs too many syscalls. isolatedProcess is the only sandboxing option on Android and Firefox doesn't use it.

Firefox's critics treat this as a universal Firefox deficiency. I see it as a platform-specific gap. If Firefox were categorically less secure you'd expect the gap to exist everywhere. On Windows it doesn't. On Android it does. Weighed against Firefox's advantages that apply on all platforms.

Picking Chrome on Android because you prioritize mobile security is rational. Claiming "Firefox has no sandbox" or "Firefox is much more vulnerable" without qualifying the platform is overreach.

0.4 3. What I Got Wrong

Post-publication review caught several things. Here they are.

0.4.1 3.1 I Left Out Half of Chromium's Mitigations

My first paper spent a lot of time on Firefox's Rust adoption and barely mentioned Chromium's mitigations. Made the comparison look one-sided. Here's what I missed:

Mitigation	What It Does
V8 Sandbox	Locks JIT code to a reserved memory region so a JS exploit can't touch anything outside it
Oilpan GC	Garbage collector for DOM objects that eliminates use-after-free in rendering code
Mojo IPC	Type-checked message passing between processes so sandbox-escape exploits have fewer holes
PartitionAlloc	Hardened heap allocator with partition isolation, freelist entropy, MiraclePtr
Type-based CFI	Validates indirect function calls at runtime to block control-flow hijacking
MTE	Memory tagging at the hardware level (integrated into PartitionAlloc)

Chromium doesn't prevent memory bugs at the source the way Rust does. It makes them significantly harder to exploit. I should have included that. Ignoring it made my comparison incomplete.

0.4.2 3.2 The isolatedProcess Thing

I categorized GrapheneOS's "Firefox has no internal sandboxing" claim as "partially outdated." That was wrong. If you define sandboxing as kernel-level UID isolation, the claim is fully accurate. GeckoView doesn't use isolatedProcess.

GrapheneOS defines sandboxing as isolatedProcess at the kernel level. I was using a broader definition that includes process-level privilege separation. Both definitions are consistent. I should have been clearer from the beginning.

0.4.3 3.3 Fission Isn't Really Shipping on Android

My first paper mentioned Fission shipping on Android then rolling back. But the abstract and conclusion referenced Fission like it was a current mitigation. It's not. Fission is disabled on release and beta channels as of Firefox 152. Nightly and developer builds have it at a reduced level (ISOLATE_HIGH_VALUE rather than full origin isolation). Should have been more careful about this.

0.4.4 3.4 Other Things I Missed

SpiderMonkey has no CFI. V8 has Clang's Cross-DSO Control Flow Integrity for indirect call validation [16]. SpiderMonkey doesn't. JS engines are the densest source of critical vulns in both browsers and this is a real gap. Combined with V8's memory sandbox, Chromium's JS engine mitigation layer is genuinely stronger. Rust doesn't help here because JIT-generated code isn't subject to Rust's safety guarantees.

I claimed hardware mitigations like MTE, PAC, and BTI are "enforced at the OS and hardware level, not by the browser vendor." That was wrong. These features need allocator modifications, compiler support, and codebase validation. Chromium has done the work. MTE in PartitionAlloc, PAC/BTI validation. Firefox hasn't.

The CVE count framing in my first paper was sloppy. Chromium invests way more in fuzzing, AI-guided auditing, and security research. Finding more bugs is a sign of more testing, not less security [18]. Residual risk after mitigation is what matters and that's not directly measurable from public data.

Extensions break site isolation in both browsers. I implied Firefox's extension model was more contained. It's not. Extensions in both browsers run with cross-origin access that bypasses site isolation [19]. The trade-off is between extensions and built-in content filtering (Brave's approach). Neither is clearly better.

0.5 4. Where We Still Disagree

Even after corrections, some disagreements come down to how you weigh the evidence rather than the evidence itself.

0.5.1 4.1 "Much More Vulnerable" vs "Differently Vulnerable"

GrapheneOS says Firefox is "much more vulnerable to exploitation." I don't think the evidence supports that even after accounting for everything I got wrong.

Chrome has stronger post-compromise containment. Firefox has stronger pre-compromise prevention. These are different priorities. Calling one "much more vulnerable" treats containment as the only metric that matters. That's a value judgment. My position is both are rational depending on what you're protecting against. Nothing I've seen changes that.

0.5.2 4.2 Advisory Latency

GrapheneOS's advisory on Firefox hasn't changed significantly since it was written. Firefox's architecture has evolved. The Fission rollback is an example of this working both ways. Needed to correct my own framing. The advisory also doesn't acknowledge that Firefox on Windows has reached sandbox parity. Documentation latency is a known problem in security engineering [6]. Not unique to any project.

0.6 5. The Personal Stuff

GrapheneOS called my paper "dishonest" and "unethical" in their Discord. On Reddit they said it wasn't "based on real research or factual information" and that I was driven by "biases."

I'm separating this from the technical corrections because it's a different category. Factual errors and dishonesty are different things. An error means peer review worked. Calling someone dishonest for making errors raises the cost of doing independent research.

I don't have institutional backing. Not affiliated with Mozilla, Google, or any vendor. The goal is the same with both papers. Improve the quality of publicly available evidence about mobile browser security. Made mistakes. Fixed them. Documented everything transparently.

GrapheneOS's technical contributions to Android security are real. Disagreeing with specific claims in their advisory doesn't change that. I can acknowledge my own errors and still think the core argument holds.

0.7 6. Bottom Line

What I got wrong and fixed: - isolatedProcess framing was imprecise - Left out Chromium's V8 sandbox, Oilpan, CFI, MTE - Didn't catch SpiderMonkey's missing CFI - Wrongly claimed hardware mitigations are automatic - Poor CVE count framing - Implied Firefox extensions have better site isolation properties

What still holds: - Firefox's Rust advantage is real and widening - Content blocking at the network layer is genuine pre-delivery prevention

- Chromium monoculture is a systemic risk - Browser choice is a

threat-model alignment - The sandbox gap is Android-specific. On Windows, Firefox is at parity

The mistakes made the original paper weaker than it should have been. Fixing them makes the argument stronger because now it accounts for the best counterarguments.

0.8 References

[1] Independent Security Research, "Comparative Analysis of Sandboxing and Mitigation Philosophies in Mobile User-Agent Architectures," Jul. 2026. [Online]. Available: https://spiritofstar.github.io/blog.html

[2] Security researchers, personal communication, Jul. 2026. Technical criticisms of [1] raised via public discussion.

[3] GrapheneOS, "Usage: Web Browsing." [Online]. Available: https://grapheneos.org/usage#web-browsing. Accessed: Jul. 2026.

[4] M. Geer et al., "CyberInsecurity: The Cost of Monopoly," 2003. [Online]. Available: https://cryptome.org/cyberinsecurity.htm

[5] B. Schneier, "Schneier on Security," various entries on security theater, security trade-offs, and adversarial thinking. [Online]. Available: https://www.schneier.com/

[6] Google Project Zero, "The State of 0-Day Mitigations," 2021. [Online]. Available: https://googleprojectzero.blogspot.com/2021/02/debunking-myths-about-0-days.html

[7] Chromium Security, "V8 Sandbox," Chromium Docs. [Online]. Available: https://chromium.googlesource.com/chromium/src/+/main/v8/src/sandbox/README.md

[8] Chromium Security, "Oilpan GC Design." [Online]. Available: https://chromium.googlesource.com/chromium/src/+/main/third_party/blink/renderer/platform/heap/BlinkGCDesign.md

[9] Mozilla, "Firefox Security Sandbox," Mozilla Wiki. [Online]. Available: https://wiki.mozilla.org/Security/Sandbox. Accessed: Jul. 2026.

[10] Mozilla, "Integrating Project Fission (Site Isolation) in Firefox," Mozilla Wiki. [Online]. Available: https://wiki.mozilla.org/Project_Fission. Accessed: Jul. 2026.

[11] M. Miller, "Site Isolation: A New Defense-in-Depth Security Architecture for the Web," Chromium Blog, 2018. [Online]. Available: https://blog.chromium.org/2018/07/site-isolation-new-defense-in-depth.html

[12] Apple, "WebKit Sandboxing." [Online]. Available: https://webkit.org/blog/14040/webkit-sandboxing/

[13] V. Nijim, "Building a More Secure Firefox with AppContainer," Mozilla Security Blog, 2019. [Online]. Available: https://blog.mozilla.org/security/2019/05/22/building-a-more-secure-firefox-with-appcontainer/

[14] Chromium Project, "Memory Safety." [Online]. Available: https://www.chromium.org/Home/chromium-security/memory-safety/. Accessed: Jul. 2026.

[15] Mozilla Security Blog, "Security Advisories." [Online]. Available: https://www.mozilla.org/en-US/security/advisories/. Accessed: Jul. 2026.

[16] Chromium Project, "Control Flow Integrity," Chromium Docs. [Online]. Available: https://chromium.googlesource.com/chromium/src/+/main/docs/security/control-flow-integrity.md. Accessed: Jul. 2026.

[17] Chromium, “PartitionAlloc Design,” Chromium Docs. [Online]. Available: https://chromium.googlesource.com/chromium/src/+/main/base/allocator/partition_allocator/PA_README.md. Accessed: Jul. 2026.

[18] Google, “OSS-Fuzz: Continuous Fuzzing for Open Source Software.” [Online]. Available: https://google.github.io/oss-fuzz/. Accessed: Jul. 2026.

[19] Chrome Developers, “Extension Runtime and Process Model.” [Online]. Available: https://developer.chrome.com/docs/extensions/mv3/process_model/. Accessed: Jul. 2026.