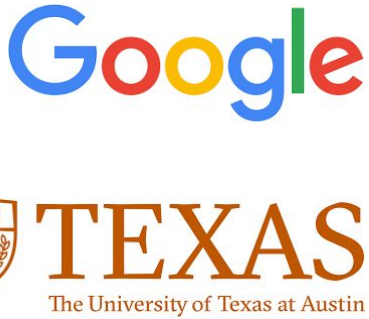
Lightweight Fault Isolation (LFI): LLVM Support for Efficient Native Code Sandboxing

Zachary Yedidia and Tal Garfinkel

Shout out

Nathan Egge, Sharjeel Khan, Daniel Moghimi, Shravan Narayan, Taehyun Noh, Abhishek Sharma, Pirama Arumuga Nainar, Derek Schuff, Colin Cross, Matthew Maurer, Dan Behrendt, Kris Adler, Wonsik Kim, Urs Hölzle, Cory Baker, Matthew Sotoudeh, Eli Friedman, MaskRay, Alexis Engelke



Memory Safety is a Big Problem

Google Chrome: ~70% of bugs (2015–2020)

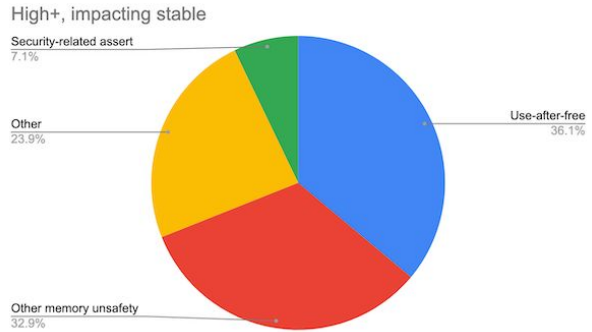


Image from the Chromium project blog
https://www.chromium.org/Home/chromium-security/memory-safety/

Microsoft Windows: ~70% of bugs (2006–2018)

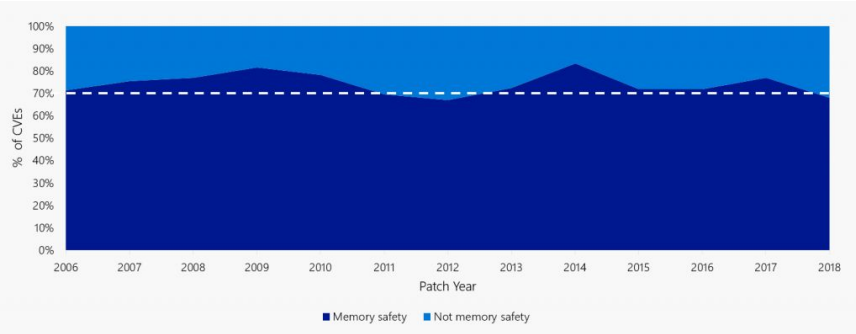


Image from the Microsoft security response center blog
https://msrc-blog.microsoft.com/2019/07/16/a-proactive-approach-to-more-secure-code/

Limited Options

Unsound Mitigations: ASLR, stack canaries, CFI, bounds checks.

+Works with existing code!

-Often bypassed

-Hard to reason about benefits + overheads

Rewrite code in safe language (Rust)?

+Sound security properties

-Huge engineering cost and time: rewrite, retest, support

-Lose benefits of Cooperation: expertise, shared ownership

Sound Security + Works with Existing Code?

Process-based Sandboxing?

The Good:

+ Sound Security Properties!
+ Works with existing library code!

The Bad: Expensive:

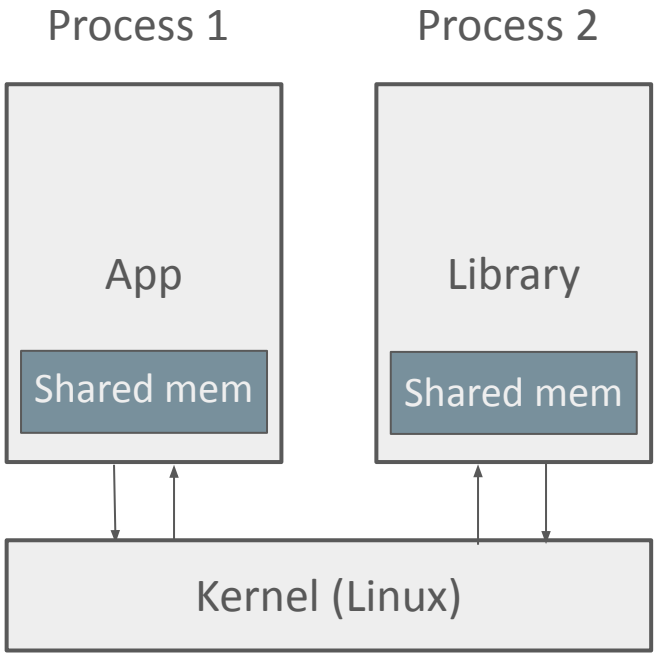
Slow context switches (IPC), Sandbox Creation, Memory Overheads

Higher Latency, Lower Throughput, Poor scalability

The Ugly: Awkward + Complex

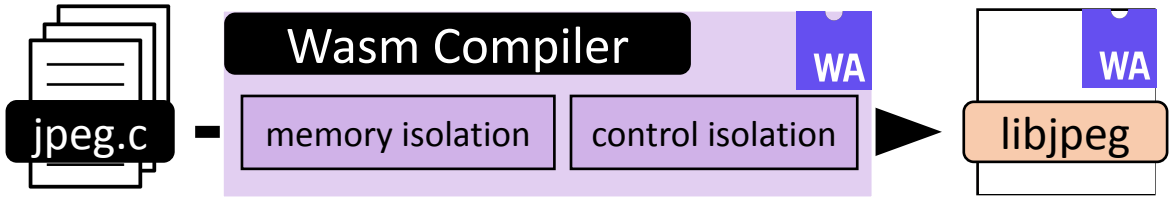
Refactoring, Developer experience 😞

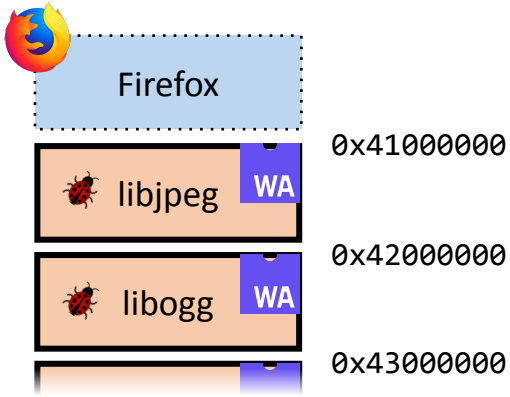
application => distributed system



Is there another option?

Sandboxing in Firefox Render with WebAssembly





100x faster startup

1000x faster context switches

Less memory

Securing Firefox with WebAssembly

By Nathan Froyd

Posted on February 25, 2020 in Featured Article, Firefox, Rust, Security, and WebAssembly

Protecting the security and privacy of individuals is a central tenet of Mozilla's mission, and so we constantly endeavor to make our users safer online. With a

https://www.usenix.org/conference/usenixsecurity20/presentation/narayan

Deployed in Firefox since 2021

Feb 2020

Mac, Linux

Font rendering

Audio playback

Decompression

XML parsing

Spell checking

Dec 2021

All platforms

(~200 million users)

Wasm has many limitations…

Compatibility ❌

Very limited SIMD/intrinsics

No hand-written assembly

No dynamic code generation (JIT)

libc compat (WASI-libc – no glibc, bionic, etc.)

Limited POSIX

ABI differences (different pointer sizes)

Break's existing tools: debuggers, perf, sanitizers,...

Performance ❌

Wasm2c on ARM64:

37.5% Geomean on SPEC 2017

Limits optimization (😞):

e.g. Precise memory traps => no truncating SFI

Breaks key library optimizations (😞)

- SIMD, Hand-written Assembly, JIT

Wasm - Good for the Web, LFI - Good for library sandboxing

High-level SFI – WebAssembly

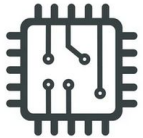
>> target sandboxed IR <<

Platform Independence

Low-level SFI – LFI, (original SFI, NaCl)

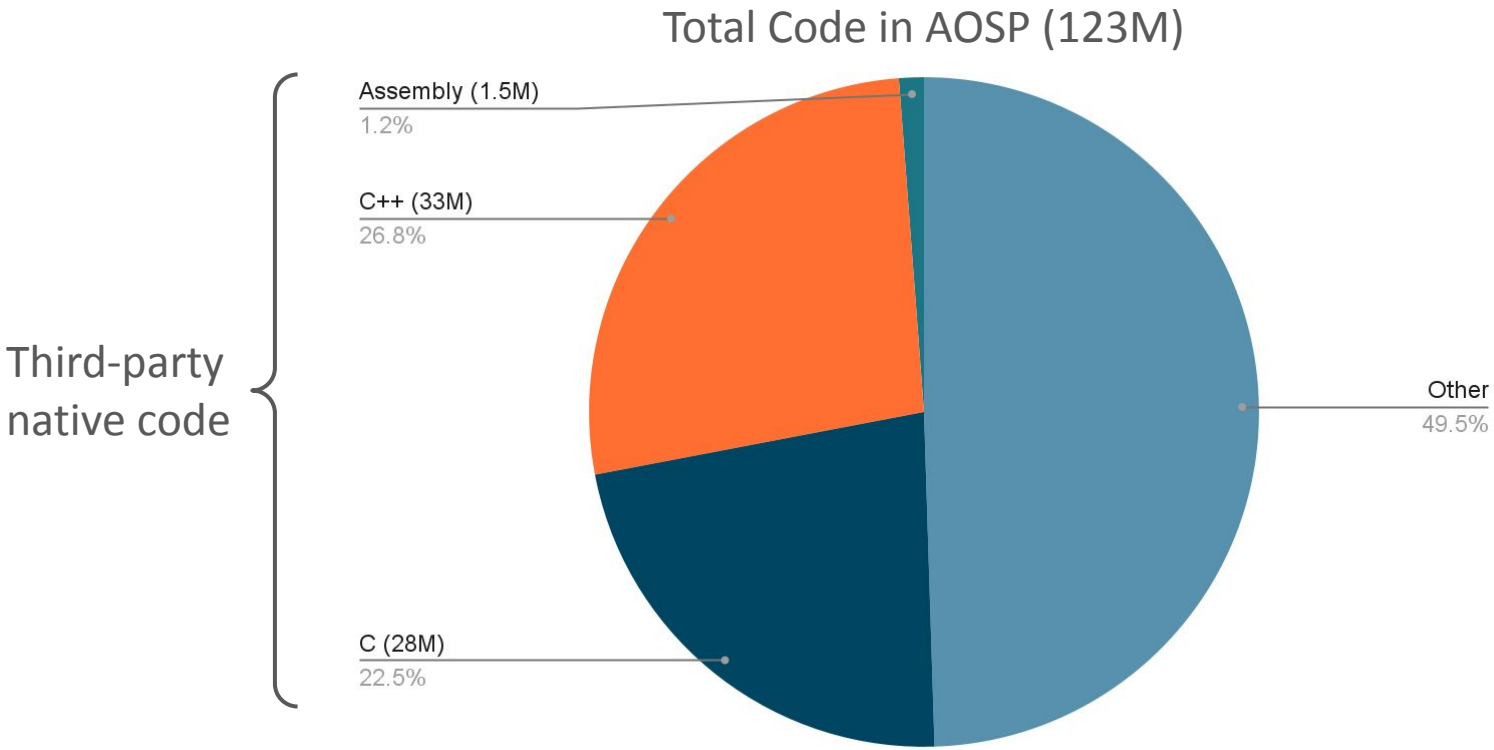
>> low level instruction rewrites <<

Compatibility, Performance, Security, Simplicity



Android

Half of AOSP is Unsafe Third-Party Code



Lightweight Fault Isolation (LFI)

Low-level SFI for isolating buggy or malicious C/C++/Asm.

Performance: fully exploit architecture and compiler

Compatibility: Linux API, 64-bit ABI, hand-written Asm, existing tools (gdb, perf, etc.)

Security: simple, verifiable => high assurance isolation.

Usability: easy retrofitting in existing code; minimal change to dev experience

…Another tool in the LLVM memory safety toolbox… RFC on discourse

Lightweight Fault Isolation

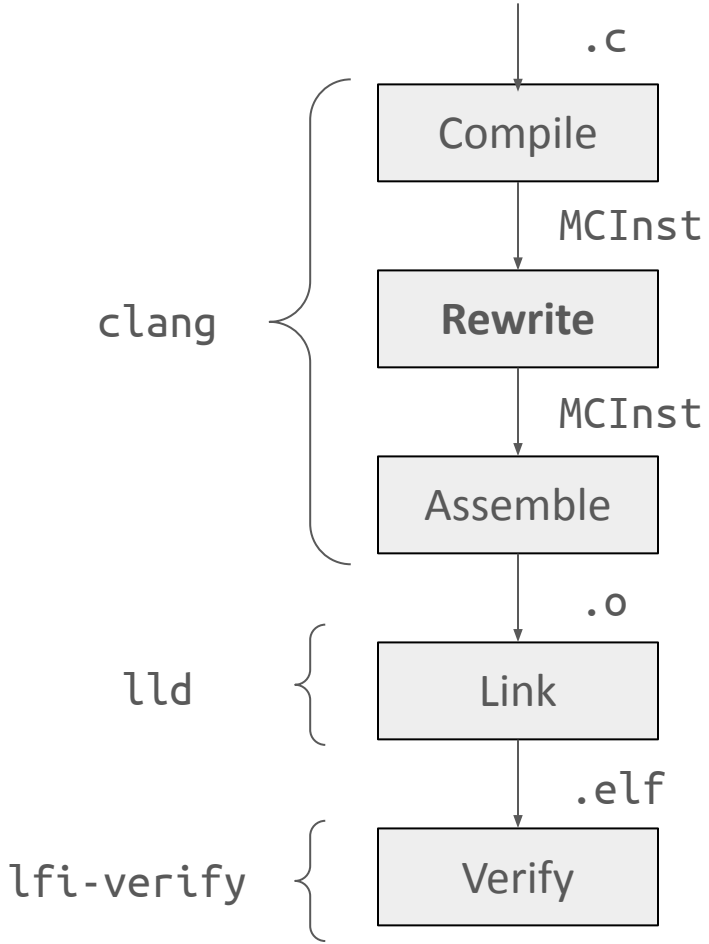
LFI Compiler Pipeline

New architecture target: `aarch64_lfi`.
(example: `aarch64_lfi-linux-musl`)

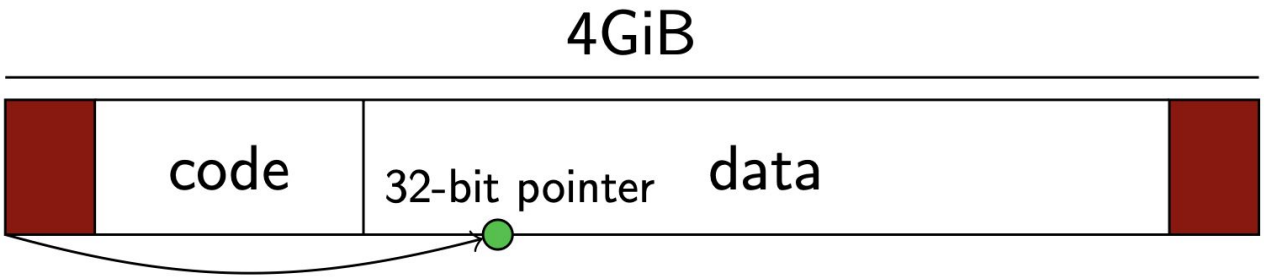
Uses custom LFI MCStreamer to apply rewrites.
Goal: automatically handle hand-written asm.
Alternative: external `.s`/`.o` rewriter.

Other compiler stages remain unchanged*.
*only need to reserve a few registers.

All sandboxed code goes through the rewriter:
Libc, dynamic linker, libc++, compiler-rt, …



LFI Execution Environment



Runtime handles syscalls (indirect branches) and enforces strict W^X.

Dynamic codegen can be supported: Run verifier before `mprotect(PROT_EXEC)`.

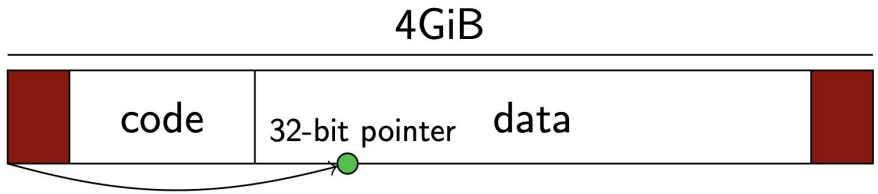
48-bit address space: supports up to 64K sandboxes (80KiB guard pages).

64-bit ABI: matches host ABI, compatible with existing code.

Rewriting Memory Accesses

Reserve x27: sandbox base.

Reserve x28: any sandbox address.




```
ldr x0, [x1]
```
→
```
ldr x0, [x27, w1, uxtw]
```
Mask with addressing mode

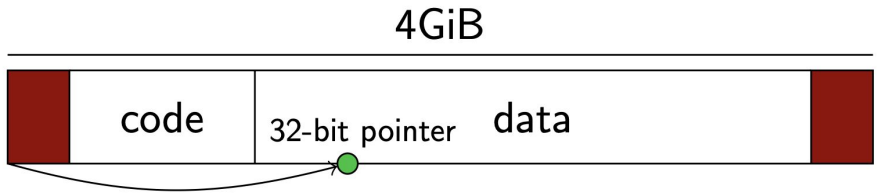
```
ldp x0, x1, [x2]
```
→
```
add x28, x27, w2, uxtw
ldp x0, x1, [x28]
```
Mask with guard instruction

For experts: Arm64 fixed-width instructions → no bundling necessary. See paper for details.

Rewriting Control Flow

Mask all modifications to x28, x30.

Rewrite indirect branches to target x28.




```
br x0
```

```
add x28, x27, w0, uxtw
br x28
```

```
ret
```

```
ret
```

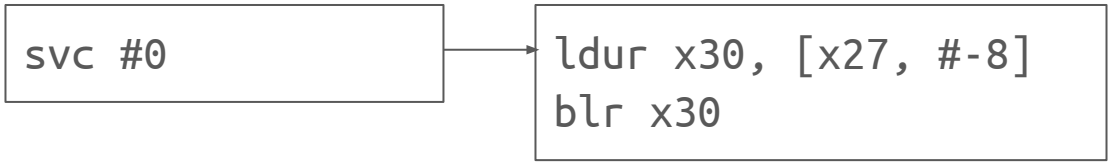
Key: Arm64 instructions are fixed-width!

Safe as long as each individual instruction is safe.

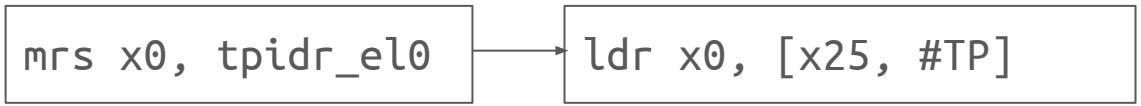
Rewriting System Call Instructions

Runtime entrypoints: placed before the first page of the sandbox (read-only).

→ x27 already points here!

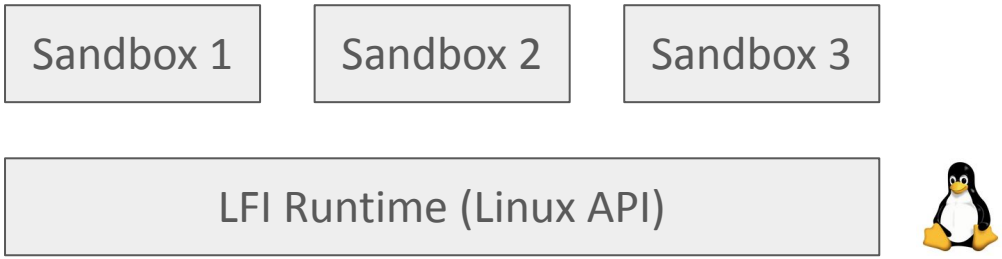


Thread-local storage: reserve a register (x25), or alternatively rewrite to runtime call.



LFI Runtime

LFI runtime: virtualizes user mode



LFI runtime: Loads sandbox, and handles system calls that come from the sandbox.

Implements a **Linux API** (similar to WASI, QEMU-user).

Many system calls are passed through with simple checks.

Putting it all together

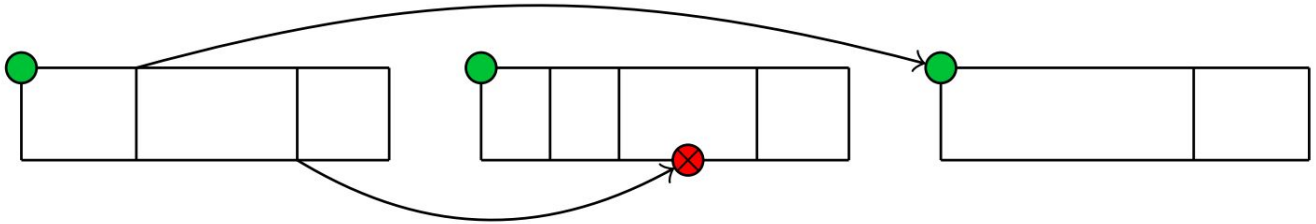
```
$ git clone https://github.com/lua/lua
$ cd lua
$ make CC=aarch64_lfi-linux-musl-clang
...
$ lfi-run ./lua
Lua 5.5.0  Copyright (C) 1994-2025 Lua.org, PUC-Rio
> print('hello world')
hello world
```

```
0000000000023900 <lua_xmove>:
   23900: eb01001f    cmp     x0, x1
   23904: 54000360    b.eq    0x23970 <lua_xmove+0x70>
   23908: 8b2042b2    add     x28, x27, w0, uxtw
   2390c: f9400a48    ldr     x8, [x28, #0x10]
   23910: 7100045f    cmp     w2, #0x1
   23914: cb22d108    sub     x8, x8, w2, sxtw #4
   23918: f9000a48    str     x8, [x28, #0x10]
```

Support for x86-64: Control Flow

Primary challenge: variable-width instructions.

- Simple solution: instruction bundles (32-byte chunks).

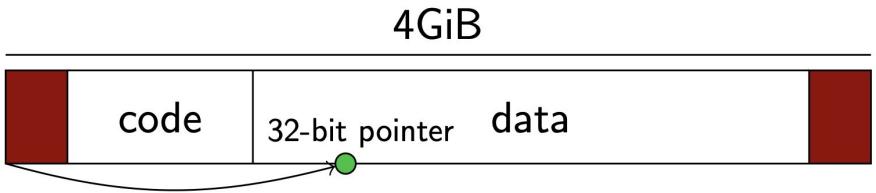


```
jmpq *%rax
```

```
andl $0xfffffffe0, %eax
addq %r14, %rax
jmpq *%rax
```

Support for x86-64: Segmentation

Problem: no `%rX + %eX` addressing mode!



But there is `%gs + %eX`…

Store sandbox base in `%gs`, and use segment-relative addressing.

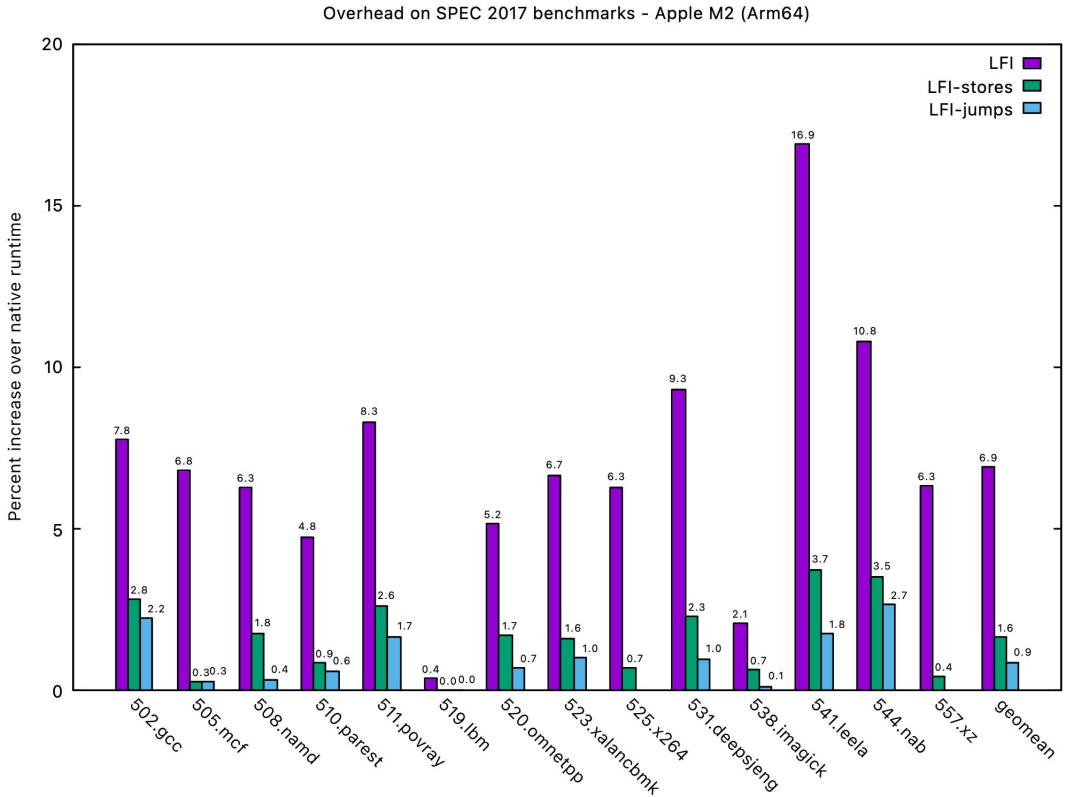
```
movq %rax, (%rdi)   →   movq %rax, %gs:(%edi)
```

Cuts performance overhead from ~15% to ~7%!

See “Segue” paper: https://shravanrn.com/pubs/seguecg.pdf

Performance

Performance: SPEC 2017

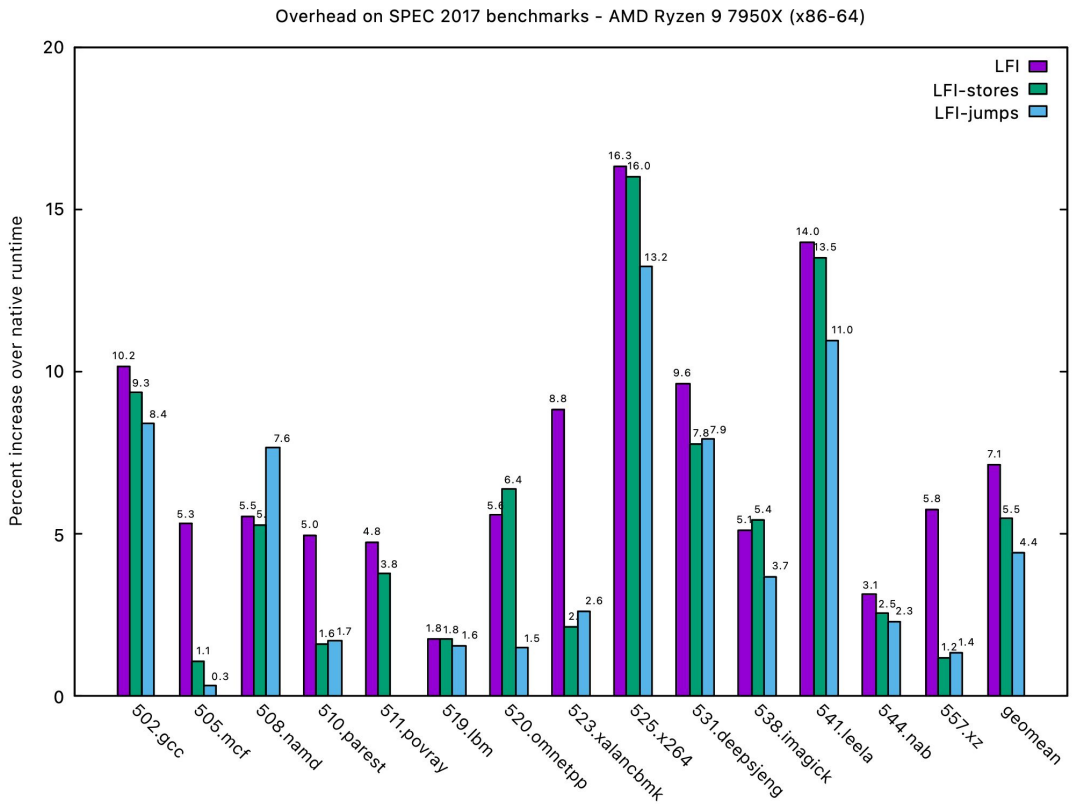


Geometric mean (full): 6.9%

Geometric mean (stores): 1.6%

Geometric mean (jumps): 0.9%

Performance (x86-64): SPEC 2017



Geometric mean (full): 7.1%

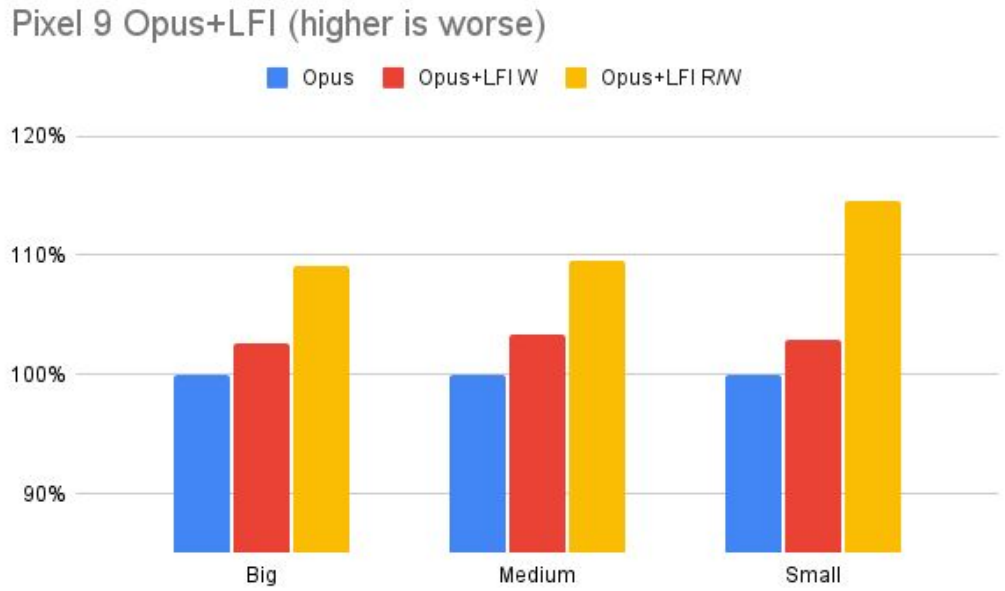
Geometric mean (stores): 5.5%

Geometric mean (jumps): 4.4%

Performance: libopus

Standalone `opus_demo` on Pixel 9, using LFI configured as store-only and load/store.

- Single-threaded, pinned to each of the core sizes.
- Performance delta normalized to unsandboxed decoder.
- Costs higher on A520 core due to in-order execution.

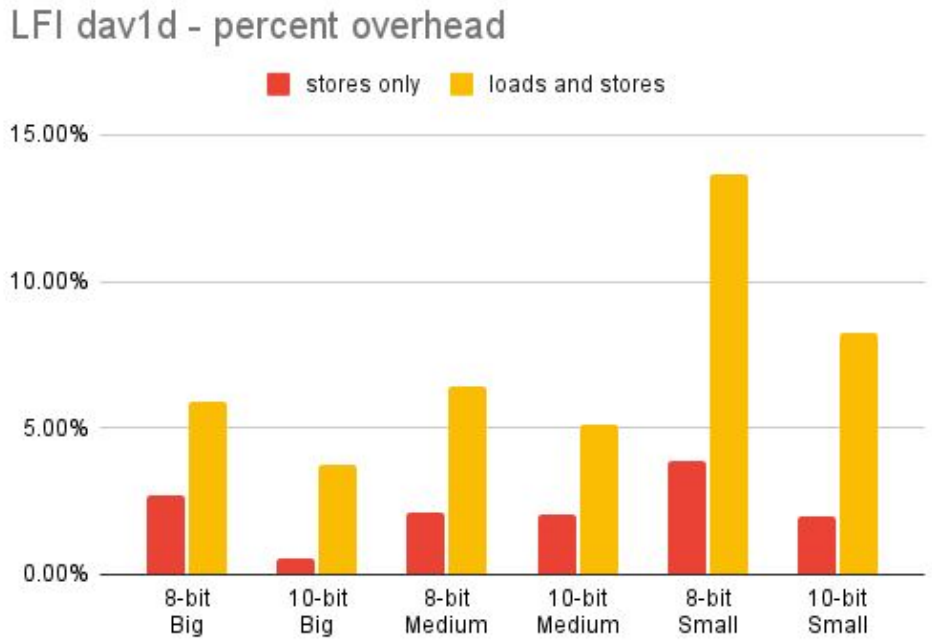


Performance: libdav1d

dav1d is an AV1 decoder:

LOTS of hand-written assembly.

```
Totals grouped by language:
asm:          234001 (85.33%)
ansic:         40075 (14.61%)
sh:              161 (0.06%)
```



Performance critical, lots of asm: “stress test” for LFI!

Microbenchmark: Trampoline Overhead

Measured on Pixel 7

Platform	Cycles	Time
Function call (native)	2	0.7ns
Function call (LFI)	74	23ns
System call (native)	358	130ns
Context switch (native)	15770	6807ns

LFI: much better than inter-process communication (and less noisy).
Future: room for trampoline optimizations.

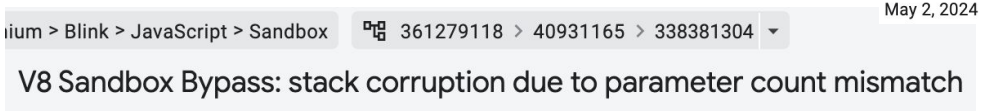
Ongoing Work

SpiderMonkey

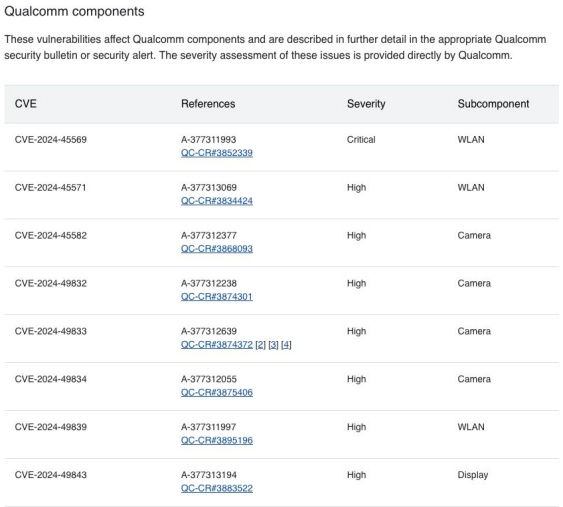
1. LFI-SpiderMonkey: Working on a secure production JIT engine.

Put the entire engine (C++ and generated code) in an LFI sandbox.

Update Gecko to use SpiderMonkey as a sandboxed library.



2. LFI Kernel Modules: Device driver isolation.



Qualcomm components

These vulnerabilities affect Qualcomm components and are described in further detail in the appropriate Qualcomm security bulletin or security alert. The severity assessment of these issues is provided directly by Qualcomm.

CVE	References	Severity	Subcomponent
CVE-2024-45569	A-377311993 QC-CR#3852339	Critical	WLAN
CVE-2024-45571	A-377313069 QC-CR#3834424	High	WLAN
CVE-2024-45582	A-377312377 QC-CR#3868093	High	Camera
CVE-2024-49832	A-377312238 QC-CR#3874301	High	Camera
CVE-2024-49833	A-377312639 QC-CR#3874372 [2] [3] [4]	High	Camera
CVE-2024-49834	A-377312055 QC-CR#3875406	High	Camera
CVE-2024-49839	A-377311997 QC-CR#3895196	High	WLAN
CVE-2024-49843	A-377313194 QC-CR#3883522	High	Display

Conclusion

Lightweight Fault Isolation:

- Fast, Compatible, Simple, Secure.
- Coming soon to LLVM and Android!

How you can help!

LFI

- LFI Pilot Studies
- LFI FFI support: Java, Swift, Python,...
- LLVM implementation: optimizations, test suite, AArch64 MCInst info…

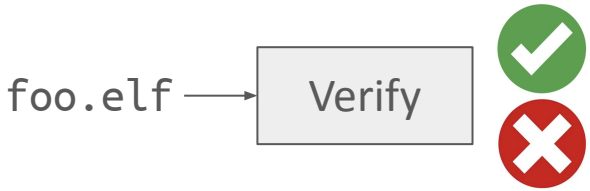
Check out RFC on Discourse!

https://lfi-project.org

Extra Slides

Simple Verification

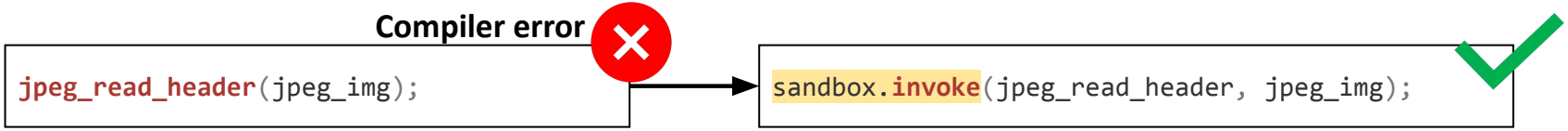
Enables: supply chain (build) safety, safe closed-source libraries, verifies rewriter.



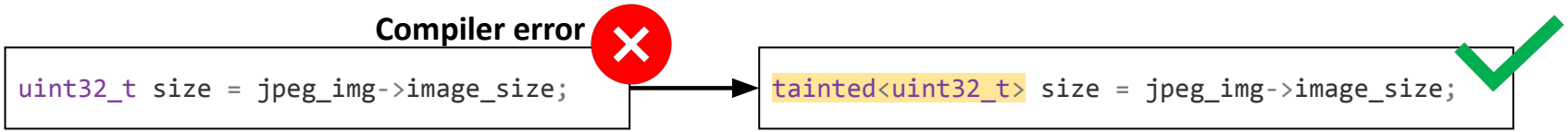
Fast (500+ MiB/s), **Simple** (~400 LoC), **Easy to fuzz/formally verify**

RLBox for Retrofitting Safely in Existing Code

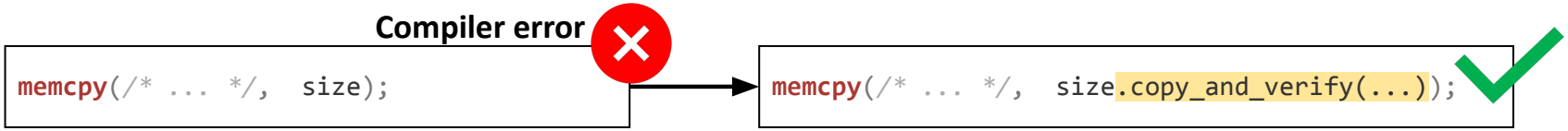
1. RLBox forces control flow to be explicit



2. RLBox forces data from the sandbox to be marked **tainted**



3. Tainted data must be checked before use



Android…

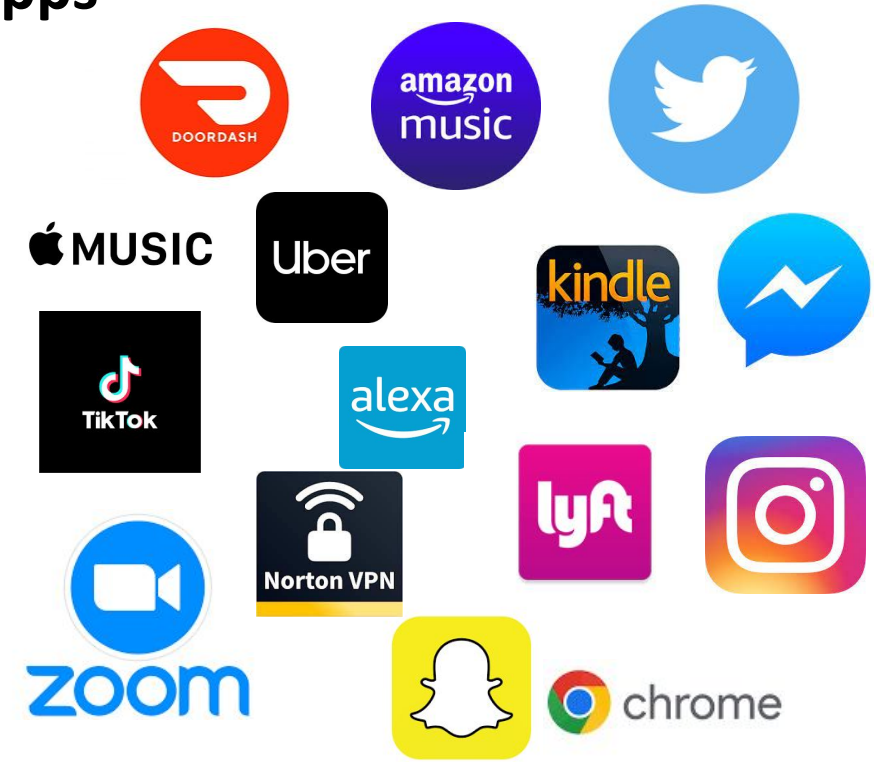
3.5 billion devices and growing

45% of global OS market

72% of global smartphone market (over 3B devices)

>45% of tablets

Lots of (Unpatched) third-party native libraries in popular Apps



Frequent vulnerabilities

App Name	Vul Lib Version	Vul Announced	TTRP (Days)	TTAF (Days)
Xbox	XML2-2.7.7	2014-11-04	12	1956
Apple Music	XML2-2.7.7	2014-11-04	12	1704
TikTok	GIFLib-5.1.1	2015-12-21	87	1429
Zoom Meetings	OpenSSL-1.0.0a	2010-08-17	91	1323
Amazon Alexa	OpenSSL-1.0.1s	2016-05-04	12	1086
Amazon Kindle	Libpng-1.6.34	2017-01-30	330	1019
StarMaker	FFmpeg-3.2	2016-12-23	4	1001
eBay	OpenCV-2.4.13	2017-08-06	41	905
Fitbit	SQlite3-3.20.1	2017-10-12	12	902
Uber	OpenCV-2.4.13	2017-08-06	41	830
Snapchat	SQlite3-3.20.1	2017-10-12	12	670
Discord	GIFLib-5.1.1	2015-12-21	87	665
Lyft	OpenCV-2.4.11	2017-08-06	41	662
Twitter	GIFLib-5.1.1	2015-12-21	87	457
Instagram	FFmpeg-2.8.0	2017-01-23	2	267

Too Quiet in the Library: An Empirical Study of Security Updates in Android Apps' Native Code