MAY 11-12

BRIEFINGS

Attacking the WebAssembly Compiler of WebKit

Zong Cao(@P1umer) Zheng Wang(@xmzyshypnc)

Who are we

Zong Cao
(@P1umer)

NeSE of IIE.CAS

Zheng Wang
(@xmzyshypnc)

Tencent Security Xuanwu Lab

Yeqi Fu
(@q1iq)

Peking University

Fangming Gu
(@afang5472)

NeSE of IIE.CAS

Bohan Liu
(@p4nda)

Tencent Security Xuanwu Lab

Why WebAssembly Compiler in WebKit ?

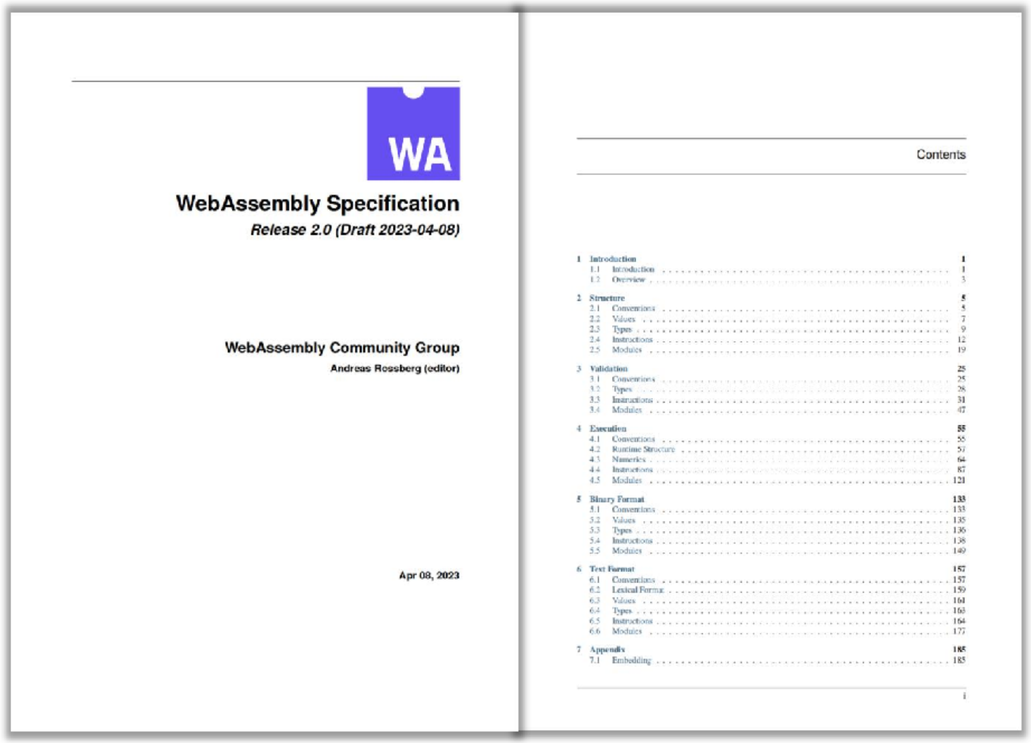
Why WebAssembly Compiler in WebKit?

#1

#2

WASM Compiler in WebKit #1

- **New** features from WebAssembly 2.0 specs
  - Wasm 2.0 compatibility roadmap



WA

WebAssembly Specification

Release 2.0 (Draft 2023-04-08)

WebAssembly Community Group

Andreas Rossberg (editor)

Apr 08, 2023

Contents

	Your browser	Chrome	Firefox	Safari	Wasmtime	Wasmer	Node.js	Deno	wasm2c
Standardized features									
JS BigInt to Wasm i64 integration	✓	85	78	14.1 [d]	N/A	N/A	15.0	1.1.2	N/A
Bulk memory operations	✓	75	79	15	0.20	1.0	12.5	0.4	1.0.30
Extended constant expressions	✗	⚑ [a]	⚑ [c]	✗	✗	✗	⚑ [g]	⚑ [l]	✗
Multi-value	✓	85	78	✓	0.17	1.0	15.0	1.3.2	1.0.24
Mutable globals	✓	74	61	✓	✓	0.7	12.0	0.1	1.0.1
Reference types	✓	96	79	15	0.20	2.0	17.2	1.16	1.0.31
Non-trapping float-to-int conversions	✓	75	64	15	✓	✓	12.5	0.4	1.0.24
Sign-extension operations	✓	74	62	14.1 [d]	✓	✓	12.0	0.1	1.0.24
Fixed-width SIMD	✓	91	89	16.4	0.33	2.0	16.4	1.9	✗
Tail calls	✓	112	✗	✗	✗	✗	⚑ [j]	⚑ [o]	✗
In-progress proposals									
Exception handling	✓	95	100	15.2	✗	✗	17.0	1.16	⚑ [q]
Garbage collection	✗	⚑ [b]	✗	✗	✗	✗	✗	✗	✗
Memory64	✗	⚑ [a]	⚑ [c]	✗	⚑ [e]	✗	⚑ [h]	⚑ [m]	⚑ [r]
Multiple memories	?	✗	✗	✗	⚑ [f]	✗	✗	✗	⚑ [s]
Relaxed SIMD	✗	⚑ [a]	⚑ [c]	✗	✗	✗	⚑ [i]	⚑ [n]	✗
Threads and atomics	✓	74	79	14.1 [d]	N/A	N/A	16.4	1.9	✗
Type reflection	?	⚑ [a]	⚑ [c]	✗	✗	2.0	⚑ [k]	⚑ [p]	✗

WASM Compiler in WebKit #2

- **Shared** security implications
- **Active** on WASM 2.0

WebAssembly SIMD

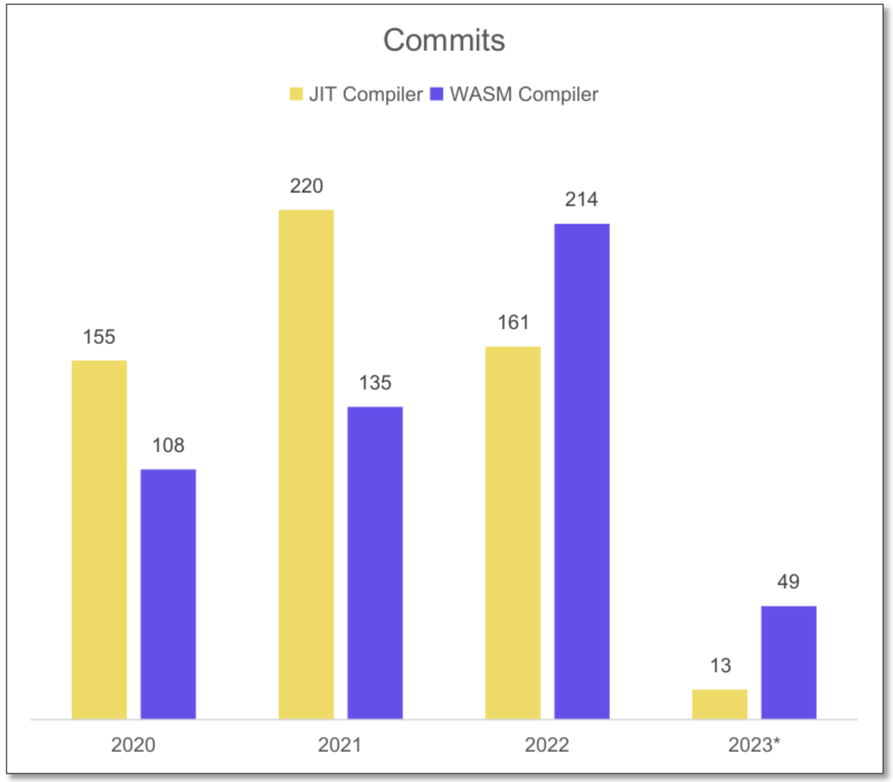
- Enabled WASM SIMD on ARM (257632@main)
- Enabled WASM SIMD on x64 (258309@main)
- Added support for conversions on Intel (257592@main)
- Added support for extended
  (257657@main)
- Added support for floating-p
  (257880@main)
- Added Intel support for load
- Added support for saturating
- Added support for swizzle a
  `replace_lane` (257400@
- Added support for bitwise o
- Added support for integer a
- Added support for vector comparisons on Intel (257532@main)
- Added Intel support for the remaining conversion opcodes (257965@main)
- Made SIM
  (257519@
- Emulated
- Emulated
- Fixed `mi`

WebAssembly

- Added support for `anyref` behind flag (261711@main)
- Implemented `cast` operations behind flag (261445@main)
- Implemented `eqref` and `ref.eq` behind flag (261663@main)
- Implemented initial minimal JS API for Wasm GC behind flag (261544@main)

WebAssembly

- Allowed WASM to use up to 4GB (r284330)
- Implemented the WebAssembly exception handling proposal (r283852)



2023*: 2023.1.1~2023.5.1
Command: git log --pretty=oneline --since=202{n}.1.1 --before=202{n}.12.31 -- ./{dir} |wc -l

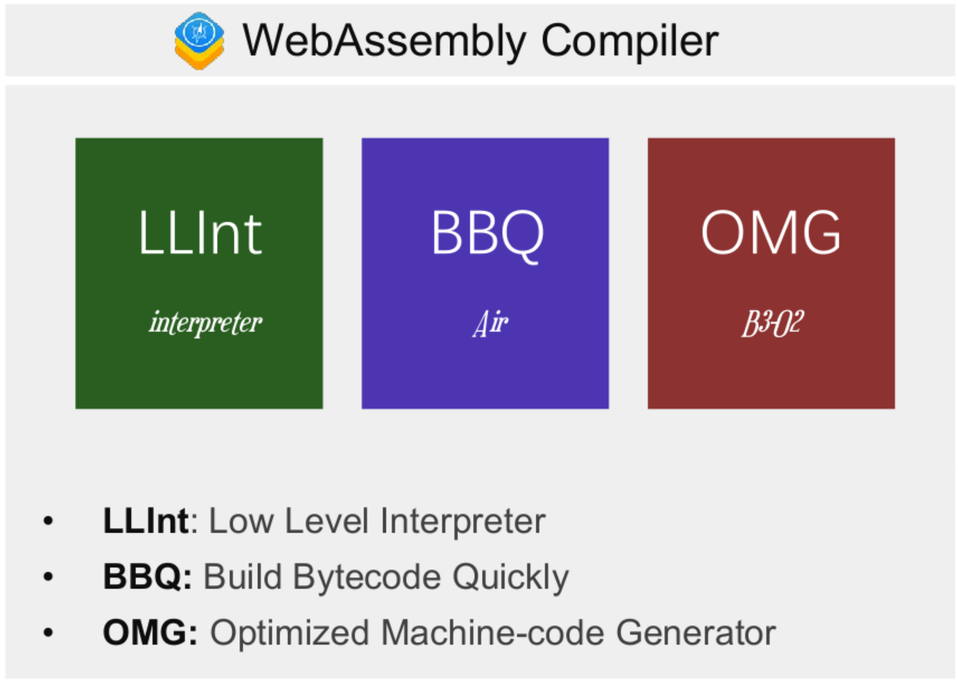
WASM Compiler in WebKit #2

- **Active** on WASM 2.0

➢ https://www.zerodayinitiative.com/blog/2021/4/2/pwn2own-2021-schedule-and-live-results

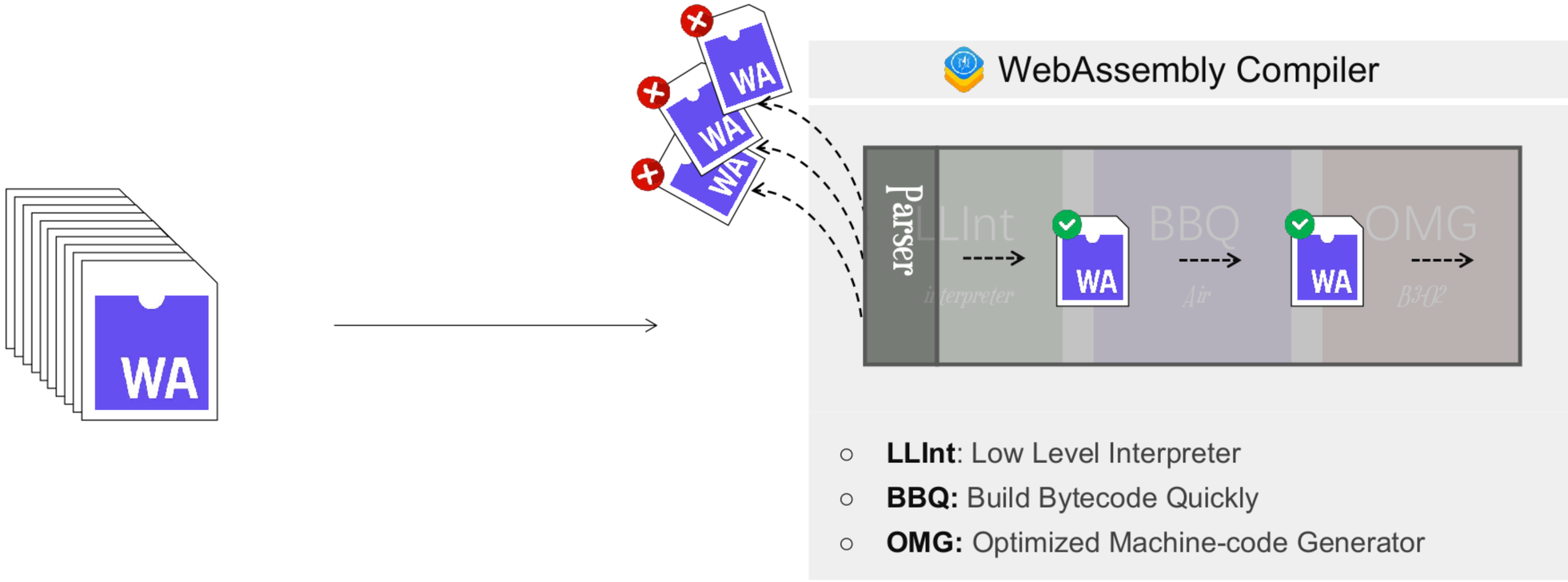
Fuzzing WebAssembly Compiler in WebKit

Fuzzing Overview



Fuzzing Overview

Main challenge: how to make the fuzzer generate **semi-well-formed** samples



- **LLInt**: Low Level Interpreter
- **BBQ:** Build Bytecode Quickly
- **OMG:** Optimized Machine-code Generator

Fuzzing Overview

Main challenge: how to make the fuzzer generate **semi-well-formed** samples

ID	Pri	Type	Component	Status	Summary + Labels	Owner
1427334	2	Bug	Blink>JavaScript>WebAssembly	Assigned	v8_wasm_compile_fuzzer: Fatal error in Exception mismatch! Expected: <RangeError: Maximum call stack si… ClusterFuzz	
1425320	2	Bug	Blink>JavaScript>WebAssembly	Verified	v8_wasm_compile_fuzzer: Crash in v… Reproducible ClusterFuzz	
1424671	1	Bug	Blink>JavaScript>WebAssembly	Duplicate	v8_wasm_compile_fuzzer: Abrt in v8… Reproducible ClusterFuzz	
1421464	2	Bug	Blink>JavaScript>WebAssembly	Verified	v8_wasm_compile_fuzzer: Fatal erro… valid module. Run with --trace-was… ClusterFuzz	
1421303	1	Bug	Blink>JavaScript>WebAssembly	Verified	v8_wasm_compile_fuzzer: CHECK fa… Reproducible ClusterFuzz	
1419637	2	Bug	Blink>JavaScript>WebAssembly	Assigned	v8_wasm_compile_fuzzer: CHECK fa… fuzzer-common.cc Reproducible Clu…	
1419622	1	Bug	Blink>JavaScript>WebAssembly	Verified	v8_wasm_compile_fuzzer: CHECK fa… Reproducible ClusterFuzz	
1417516	1	Bug	Blink>JavaScript>WebAssembly	Verified	v8_wasm_compile_fuzzer: DCHECK … liftoff-assembler-arm.h Reproducibl…	
1405706	1	Bug	Blink>JavaScript>Compiler, Blink>JavaScript>WebAssembly	Duplicate	v8_wasm_compile_fuzzer: Abrt in v8… Reproducible ClusterFuzz	
1404880	1	Bug-Security	Blink>JavaScript>Runtime	Duplicate	v8_wasm_compile_fuzzer: DCHECK … HAS_STRONG_HEAP_OBJECT_TAG… ClusterFuzz allpublic	
1404876	1	Bug-Security	Blink>JavaScript>WebAssembly	Duplicate	v8_wasm_compile_fuzzer: DCHECK failure in feedback_instruction_index_ < type_feedback_.size() in graph-bu… interface.c Reproducible ClusterFuzz allpublic	
1404761	1	Bug	Blink>JavaScript>GarbageCollection, Blink>JavaScript>WebAssembly	Duplicate	v8_wasm_compile_fuzzer: Null-dereference READ in v8::internal::RootScavengeVisitor::ScavengePointer Reproducible ClusterFuzz	
1404712	1	Bug-Security	Blink>JavaScript>GarbageCollection	Duplicate	v8_wasm_compile_fuzzer.exe: Crash in v8::internal::Heap::IterateF… Reproducible ClusterFuzz allpublic	
1404655	1	Bug-Security	Blink>JavaScript>WebAssembly	Duplicate	v8_wasm_compile_fuzzer: DCHECK failure in (address & ::v8::internal::kHeapObjectTagMask) == 0 in heap-object.h Reproducible ClusterFuzz allpublic	clemensb@chromium.org

main binaryen / src / tools / fuzzing /

tlively Remove the ability to construct basic types in a TypeBuilder (#5678) …

..

fuzzing.cpp
heap-types.c
heap-types.h
parameters.h
random.cpp
random.h

wasm-smith

A WebAssembly test case generator.

docs passing | crates.io v0.12.7 | downloads 287k | Rust

- Features
- Usage
  - With `cargo fuzz` and `libfuzzer-sys`
  - As a Command Line Tool

Fuzzing Overview

Main challenge: how to make the fuzzer generate **semi-well-formed** samples

So Good !

ID	Pri	Type	Component		Summary + Labels	Owner
1427334	2	Bug			v8_wasm_compile_fuzzer: Fatal error in Exception mismatch! Expected: <RangeError: Maximum call stack si ClusterFuzz	
					v8_wasm_compile_fuzzer: Crash in v Reproducible ClusterFuzz	
				Duplicate	v8_wasm_compile_fuzzer: Abrt in v8 Reproducible ClusterFuzz	
			Blink>JavaScript>WebAssembly	Verified	v8_wasm_compile_fuzzer: Fatal erro valid module. Run with --trace-was ClusterFuzz	
	1	Bug	Blink>JavaScript>WebAssembly	Verified	v8_wasm_compile_fuzzer: CHECK fa Reproducible ClusterFuzz	
1419637	2	Bug	Blink>JavaScript>WebAssembly	Assigned	v8_wasm_compile_fuzzer: CHECK fa fuzzer-common.cc Reproducible Clu	
1419622	1	Bug	Blink>JavaScript>WebAssembly	Verified	v8_wasm_compile_fuzzer: CHECK fa Reproducible ClusterFuzz	
1417516	1	Bug	Blink>JavaScript>WebAssembly	Verified	v8_wasm_compile_fuzzer: DCHECK liftoff-assembler-arm.h Reproducibl	
1405706	1	Bug	Blink>JavaScript>Compiler, Blink>JavaScript>WebAssembly	Duplicate	v8_wasm_compile_fuzzer: Abrt in v8 Reproducible ClusterFuzz	
1404880	1	Bug-Security	Blink>JavaScript>Runtime	Duplicate	v8_wasm_compile_fuzzer: DCHECK HAS_STRONG_HEAP_OBJECT_TAG ClusterFuzz allpublic	
1404876	1	Bug-Security	Blink>JavaScript>WebAssembly	Duplicate	v8_wasm_compile_fuzzer: DCHECK failure in feedback_instruction_index_ < type_feedback_.size() in graph-bu interface.c Reproducible ClusterFuzz allpublic	
1404761	1	Bug	Blink>JavaScript>GarbageCollection, Blink>JavaScript>WebAssembly	Duplicate	v8_wasm_compile_fuzzer: Null-dereference READ in v8::internal::RootScavengeVisitor::ScavengePointer Reproducible ClusterFuzz	
1404712	1	Bug-Security	Blink>JavaScript>GarbageCollection	Duplicate	v8_wasm_compile_fuzzer.exe: Crash in v8::internal::Heap::IterateF Reproducible ClusterFuzz allpublic	
1404655	1	Bug-Security	Blink>JavaScript>WebAssembly	Duplicate	v8_wasm_compile_fuzzer: DCHECK failure in (address & ::v8::internal::kHeapObjectTagMask) == 0 in heap-object.h Reproducible ClusterFuzz allpublic	clemensb@chromium.org

main binaryen / src / tools / fuzzing /

tlively Remove the ability to construct basic types in a TypeBuilder (#5678) ...

..

fuzzing.cpp

heap-types.c

heap-types.h

parameters.h

random.cpp

random.h

wasm-smith

A WebAssembly test case generator.

docs passing | crates.io v0.12.7 | downloads 287k | Rust

- Features
- Usage
  - With `cargo fuzz` and `libfuzzer-sys`
  - As a Command Line Tool

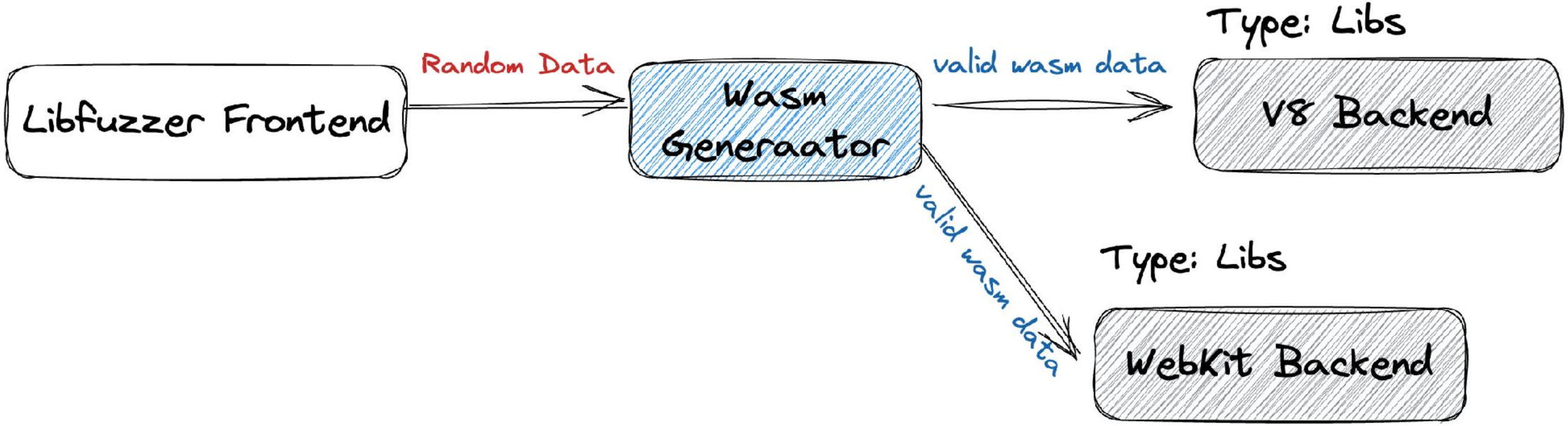
Our Approach: Inspiration

- *v8_wasm_compile_fuzzer*: 3 parts with strong binding

- **LibFuzzer Frontend**: Generate random data
- **Wasm Generator:** Convert ramdom data to valid & general wasm module
- **V8 Backend:** Embeded V8 as harness

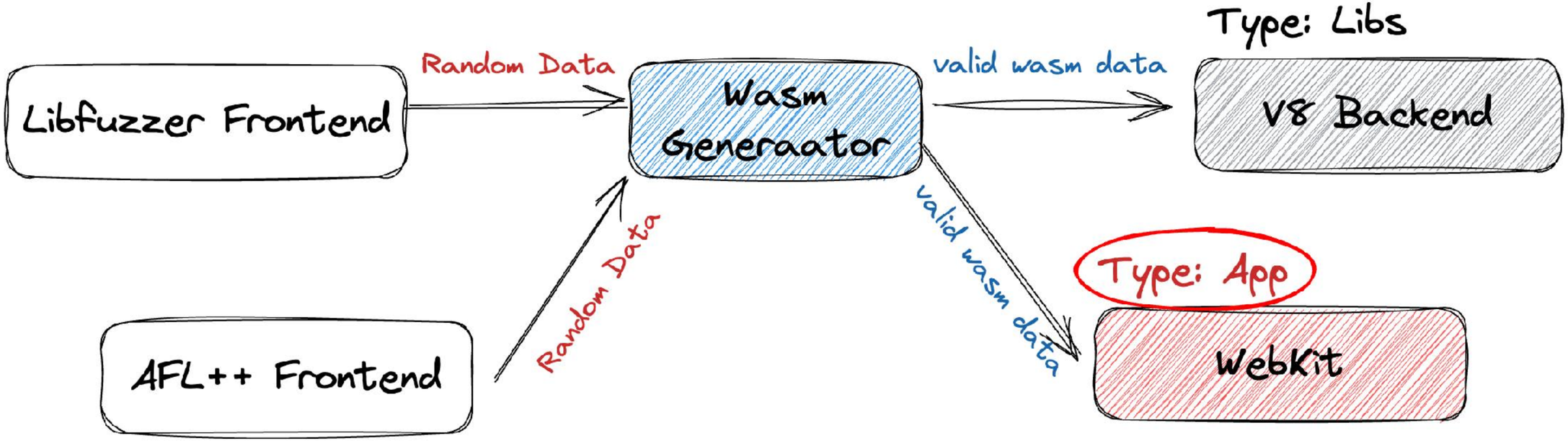
Our Approach: Inspiration

- **Original idea:** Port WebKit backend, but there are some issues:
  - Code complexity
  - Integration effort



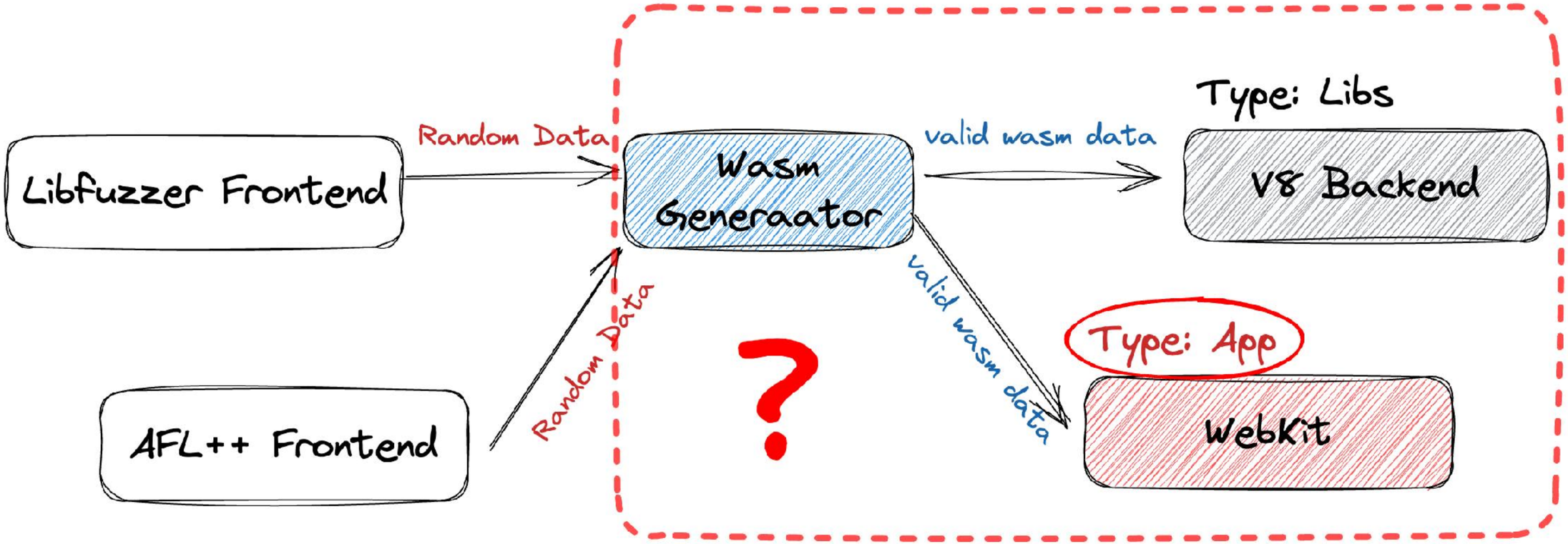
Our Approach: Inspiration

- Use AFL++ for complete WebKit application fuzzing
- Goal shifts: Port WebKit to *wasm_compile_fuzzer* ⇨ enable AFL++ with wasm generator



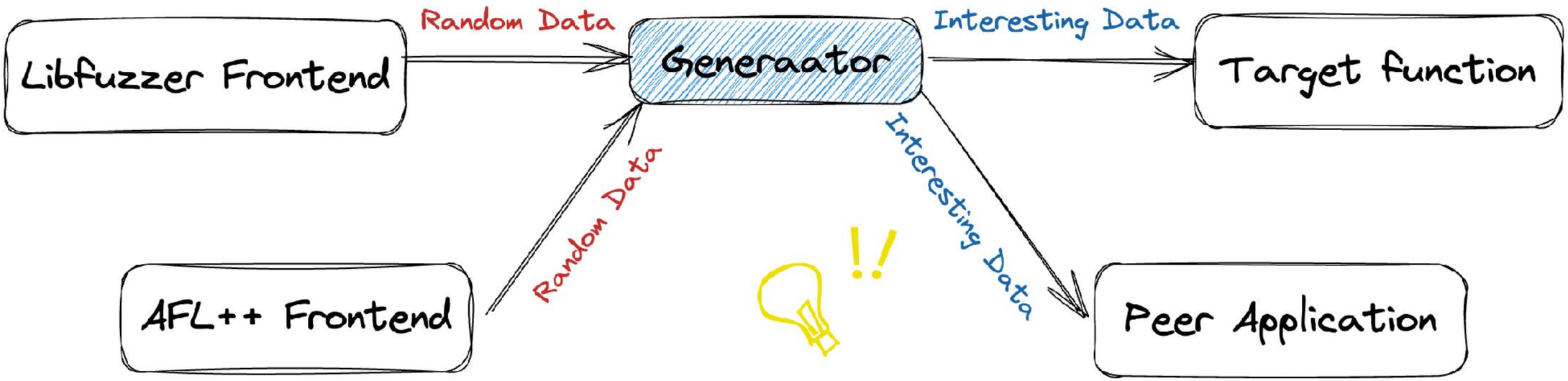
Our Approach: Inspiration

- **Generator Reuse:** not only between V8 & WebKit



Our Approach: Inspiration

- **Generator Reuse:** applicable across peer-applications



Our Approach: Inspiration

- **Where are Peer Applications?**
  - Different JS engines/compilers
  - Similar libraries & frameworks
  - Different protocol implementations

```
// The actual fuzz target that consumes the PNG data.
extern "C" int FuzzPNG(const uint8_t* data, size_t size);

DEFINE_PROTO_FUZZER(const PngProto &png_proto) {
  auto s = ProtoToPng(png_proto);
  FuzzPNG((const uint8_t*)s.data(), s.size());
}
```

```
DEFINE_PROTO_FUZZER(const json_proto::JsonParseAPI &json_proto) {
  json_proto::JsonProtoConverter converter;
  std::string data_str = converter.Convert(json_proto.object_value());
  int32_t hash_settings = json_proto.settings();
  FuzzJson(data_str, hash_settings);
```

```
DEFINE_PROTO_FUZZER(const xmlProtoFuzzer::XmlDocument& xmlDocument) {
    std::string xmlData = xmlProtoFuzzer::ProtoConverter().protoToString(xmlDocument);
    parseInMemory((const uint8_t *)xmlData.c_str(), xmlData.size());
}
```

Our Approach: AFL++ Extractor

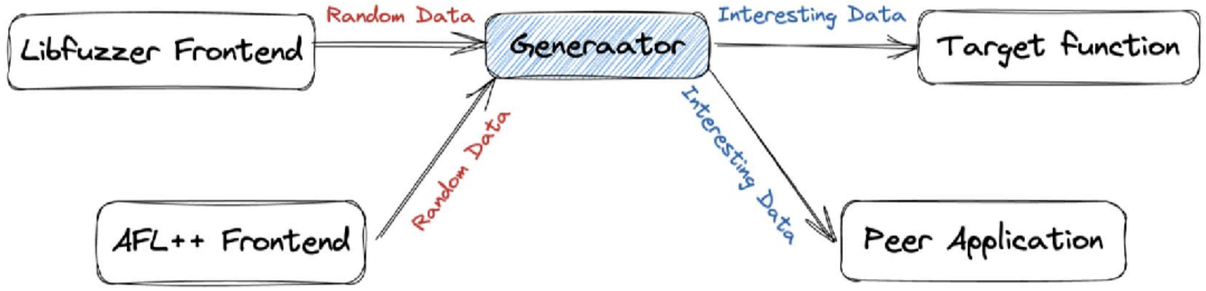
- An AFL++ plugin with:
  - **Concise** code
  - **Easy** usage
  - **Remarkable** results

AFL++ Plugin: Data-Generator Extractor

This plugin allows AFL++ to use LibFuzzer's data generators for fuzzing. The plugin extracts generators from LibFuzzer and integrates them into AFL++.

Motivation

他山之石，可以攻玉

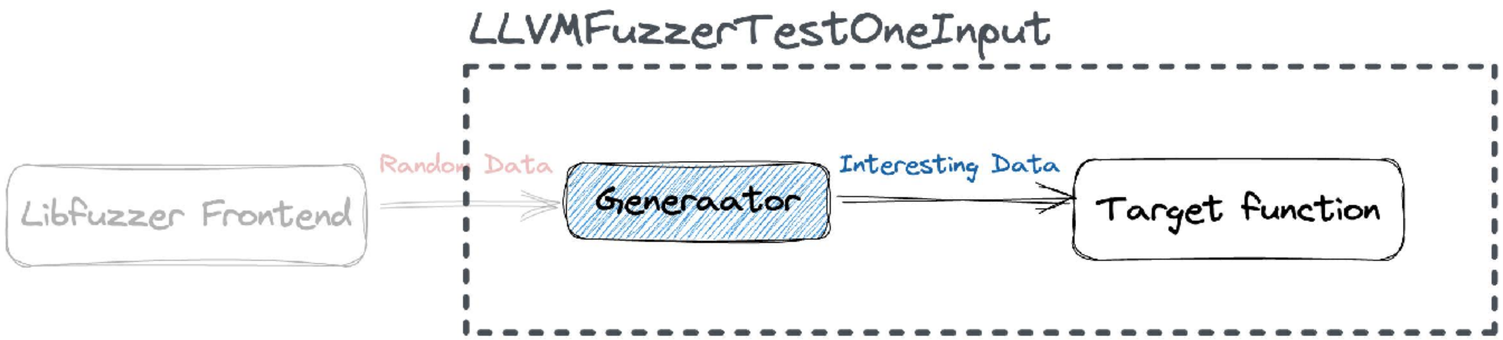


The primary motivation behind this plugin is to enhance the fuzzing capabilities of AFL++ by utilizing the powerful data generators present in specific LibFuzzers. These data generators can construct complex inputs that meet certain structures, syntax, or semantics, making AFL++ more effective when testing PEER applications. This idea is inspired by a saying from the Chinese classic "Book of Songs" (诗经): "The stones from other hills can be used to polish jade" (他山之石, 可以攻玉), emphasizing the value of learning from others to improve oneself.

➢ **GitHub:** https://github.com/P1umer/AFLplusplus-Extractor/tree/libfuzzer_extractor/utils/aflpp_extractor

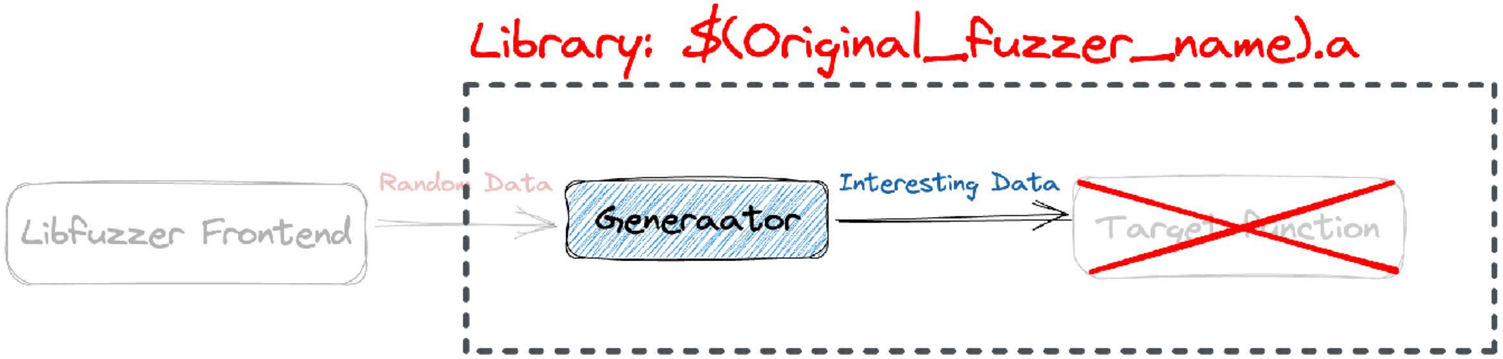
Our Approach: AFL++ Extractor

1. LibFuzzer divided into Generator and Harness logically



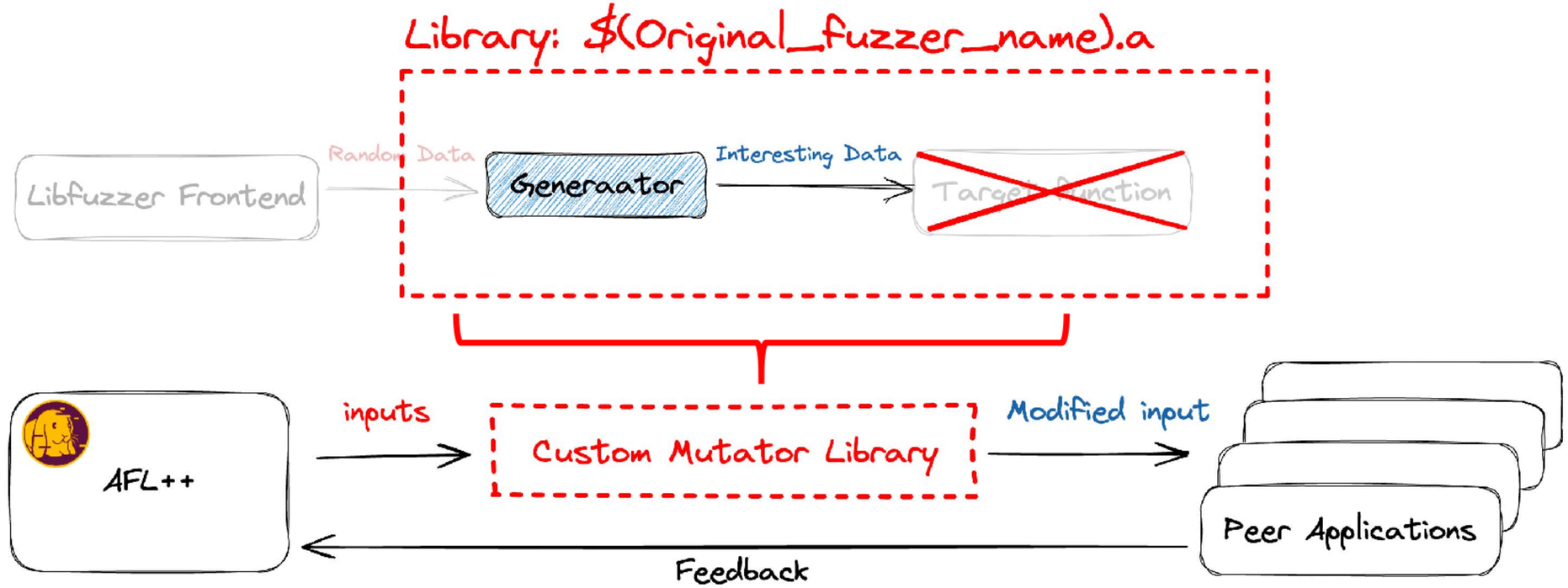
Our Approach: AFL++ Extractor

2. Remove the invocation of the target functions via patching.



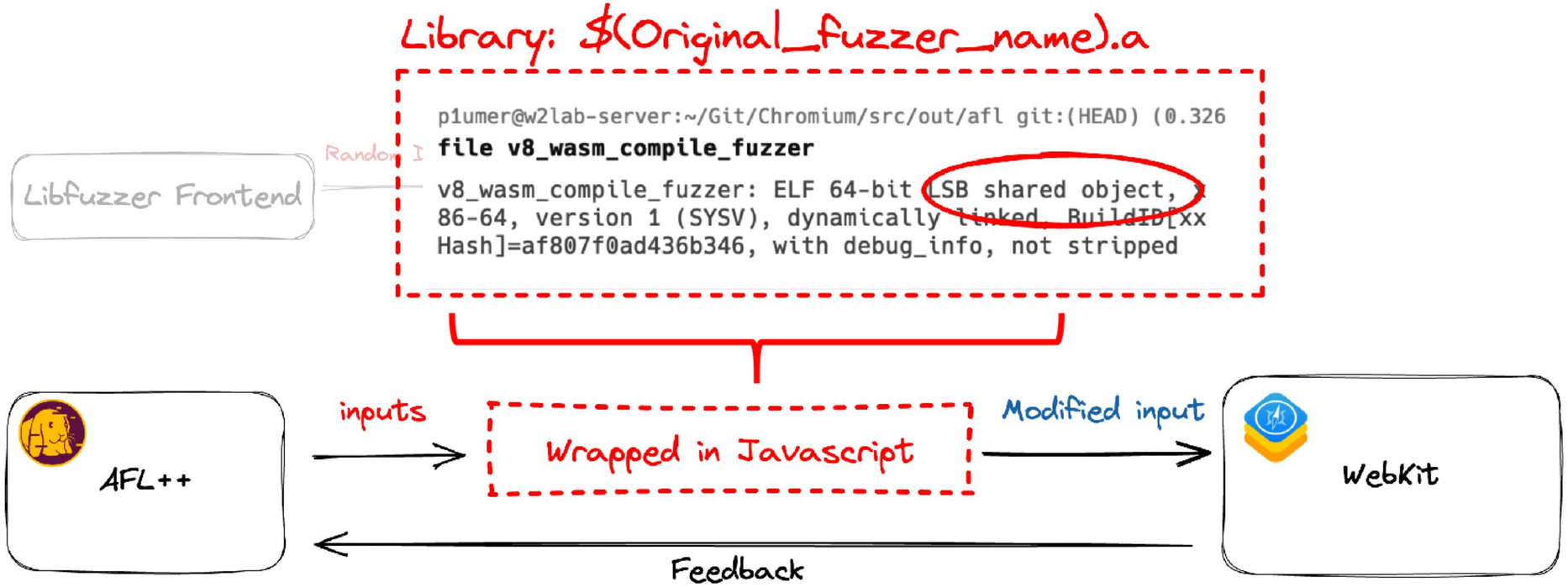
Our Approach: AFL++ Extractor

3. Compile libfuzzer into a shared library using ***afl-cc*** modified by AFL++ Extractor
   - Make an AFL++ custom mutator based on this shared library



Our Approach: AFL++ Extractor

✓ Apply **AFL++ Extractor** to *v8_wasm_compile_fuzzer*

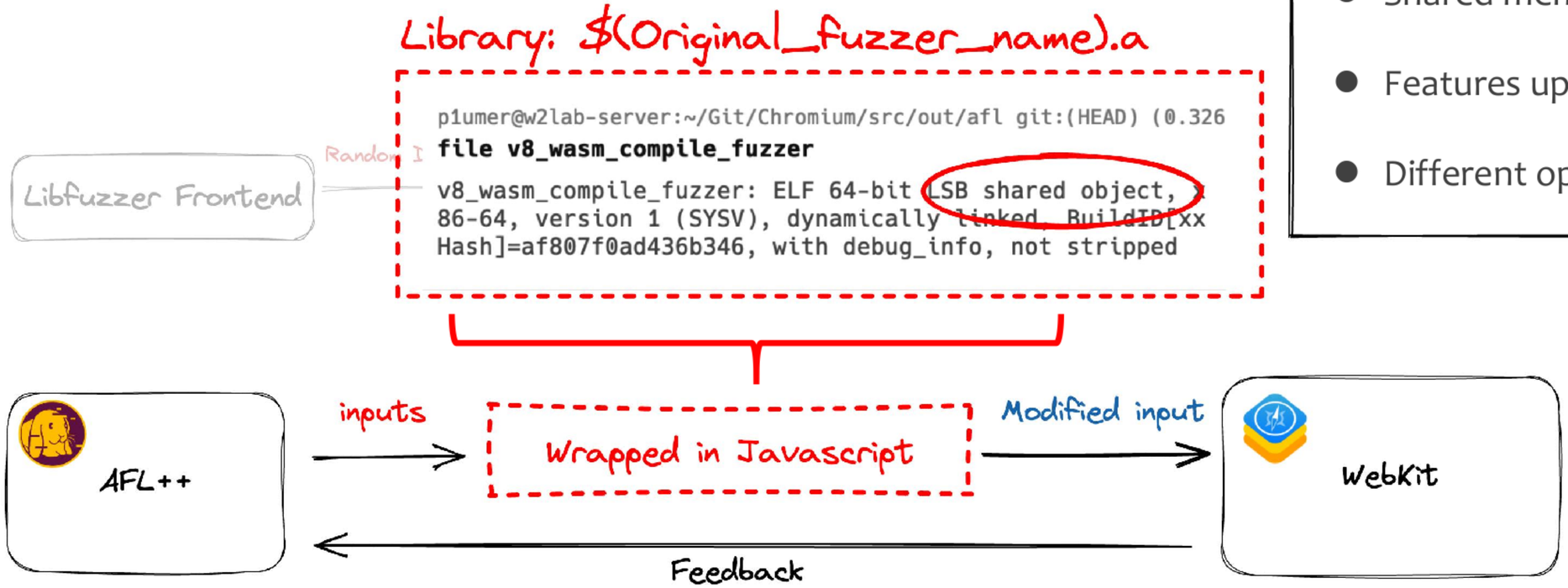


Our Approach: AFL++ Extractor

✓ Apply **AFL++ Extractor** to *v8_wasm_compile_fuzzer*

Optimize:

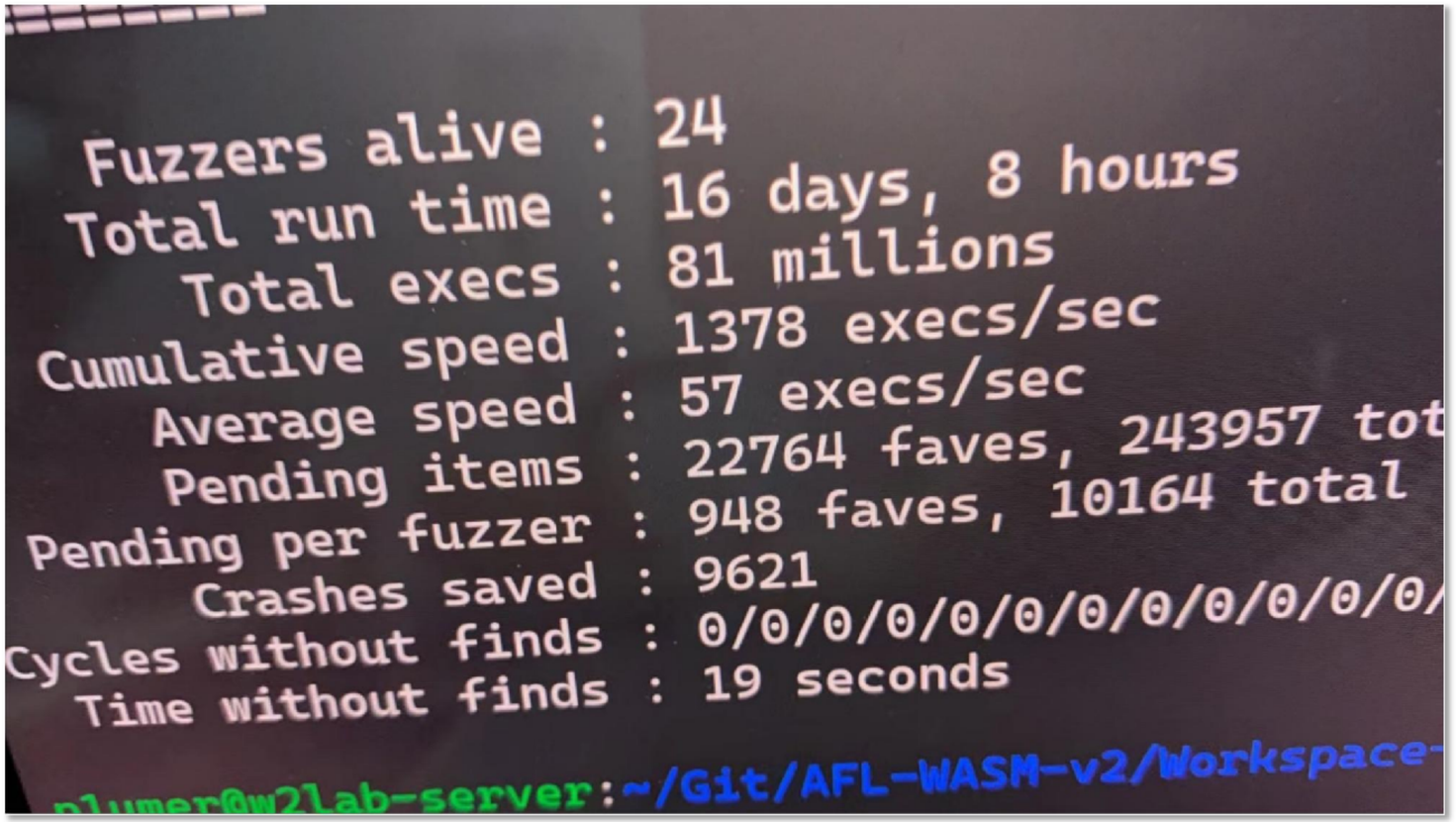
- Persistent mode
- Shared memory fuzzing
- Features update
- Different options



Result

- 13 Security-Related Issues
- 4 CVEs & Acknowledgements
- 3 Tiers of Pipeline
- 2 arch: x64 + arm64

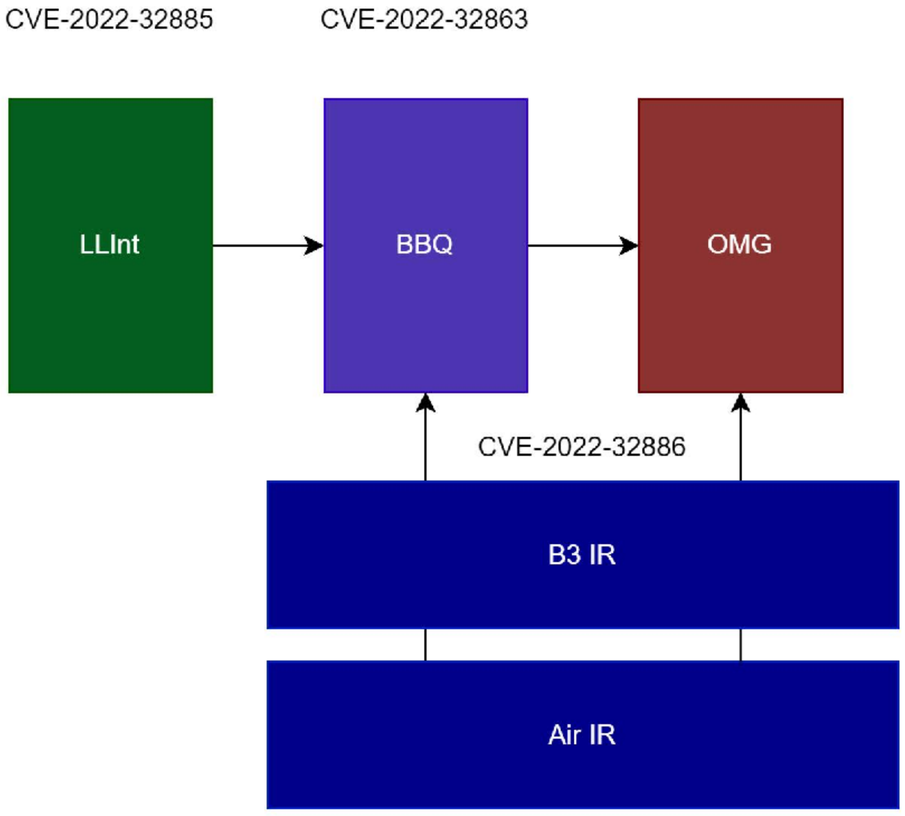
Still Progressing ...



Cases Study

WebKit Wasm Compilers

1. **LLInt** : Interpreter
2. **B3 and Air** : Low-level optimizer, IRGenerator
3. **BBQ** : Fast in compiling
4. **OMG** : Fast in executing



WebKit Wasm Compilers

LLInt Interpreter

- Parse
- Bytecode Generation
- Execute

```
(module
  (func (export "add") (param i32 i32) (result i32)
    local.get 0
    local.get 1
    i32.add
    block (param i32) (result i32)
      i32.const 1337
      i32.add
    end
    return
  )
)
```

LLInt/WebAssembly.asm

```
wasmOp(i32_add, WasmI32Add, macro(ctx)
    mloadi(ctx, m_lhs, t0)
    mloadi(ctx, m_rhs, t1)
    addi t0, t1, t2
    returni(ctx, t2)
end)
```

- src: JavaScriptCore/llint/WebAssembly.asm

WebKit Wasm Compilers

BBQ

- BBQ omits many optimizations in the B3 compiler

B3 IR

Double-To-Float
Simplify (folding, CFG, etc)
Legalization
Constant Motion
Lower to Air (isel)

Air IR

Simplify CFG
Macro Lowering
DCE
Linear Scan Reg+Stack Alloc
Fix Partial Register Stalls
Lower Multiple Entrypoints
Select Block Order
Emit Machine Code

➢ Ref: 《All About JavaScriptCore's Many》 - Filip Pizlo

WebKit Wasm Compilers

OMG

- OMG uses as many optimizations as possible to generate code that executes quickly.

B3 IR
Double-To-Float
Simplify (folding, CFG, etc)
LICM
Global CSE
Switch Inference
Tail Duplication
Path Constants
Macro Lowering
Legalization
Constant Motion
Lower to Air (isel)

Air IR
Simplify CFG
Macro Lowering
DCE
Graph Coloring Reg Alloc
Spill CSE
Graph Coloring Stack Alloc
Report Used Registers
Fix Partial Register Stalls
Lower Multiple Entrypoints
Select Block Order
Emit Machine Code

➢ Ref: 《All About JavaScriptCore's Many》 - Filip Pizlo

CVE-2022-32863

Vulnerability analysis

- Vulnerability exists in the **BBQ Air**.
- Inappropriate implementation in Wasm stackoverflow check
- Type Confusion of **JSWebAssemblyInstance**

CVE-2022-32863

Vulnerability analysis

[0] : Calculate rsp(in x86_64) by rbp

[1] : Init |this| slot in call frame

Uninitialized Value !

```
AirIRGenerator::AirIRGenerator(...)
{
    // [...]
    m_prologueGenerator = createSharedTask<B3::Air::PrologueGeneratorFunction>([=, this] (CCallHelpers& jit,
B3::Air::Code& code) {
        // [...]
        {
            if (needsOverflowCheck) {
                // […]
                jit.addPtr(CCallHelpers::TrustedImm32(-checkSize), GPRInfo::callFrameRegister, scratch);
                MacroAssembler::JumpList overflow;
                if (UNLIKELY(needUnderflowCheck))
                    overflow.append(jit.branchPtr(CCallHelpers::Above, scratch, GPRInfo::callFrameRegister));
                    overflow.append(jit.branchPtr(CCallHelpers::Below, scratch,
CCallHelpers::Address(m_prologueWasmContextGPR, Instance::offsetOfCachedStackLimit())));
                    jit.addLinkTask([overflow] (LinkBuffer& linkBuffer) {
                      linkBuffer.link(overflow,
CodeLocationLabel<JITThunkPtrTag>(Thunks::singleton().stub(throwStackOverflowFromWasmThunkGenerator).code()));
                });
            } // […]
            if (m_catchEntrypoints.size()) {
                GPRReg scratch = wasmCallingConvention().prologueScratchGPRs[0];
                jit.loadPtr(CCallHelpers::Address(m_prologueWasmContextGPR, Instance::offsetOfOwner()),
scratch);
                jit.store64(scratch, CCallHelpers::Address(GPRInfo::callFrameRegister,
CallFrameSlot::thisArgument * sizeof(Register)));
            }
        }
    });
}
```

[0]

[1]

➢ src: JavaScriptCore/wasm/WasmAirIRGenerator.cpp

CVE-2022-32863

How to trigger

1. Exception handler exists in wasm code
2. Hit a StackOverflow exception :
   operationWasmToJSException
   ->genericUnwind
   ->Interpreter::unwind
   ->StackVisitor::visit

```
static void visit(CallFrame* startFrame, VM& vm, const Functor& functor)
{
        StackVisitor visitor(startFrame, vm);
        if (action == TerminateIfTopEntryFrameIsEmpty && visitor.topEntryFrameIsEmpty())
            return;
        while (visitor->callFrame()) {
            IterationStatus status = functor(visitor);
            if (status != IterationStatus::Continue)
                break;
            visitor.gotoNextFrame();
        }
}
```

➢ src: JavaScriptCore/interpreter/StackVisitor.h

CVE-2022-32863

Vulnerability analysis

Fetch |this| as jsInstance

Type confusion happened

```
IterationStatus operator()(StackVisitor &visitor) const
{
  // [...]

#if ENABLE(WEBASSEMBLY)
  CalleeBits callee = visitor->callee();
  if (callee.isCell())
  {
    if (auto *jsToWasmICCallee = jsDynamicCast<JSToWasmICCallee *>(callee.asCell()))
      m_vm.wasmContext.store(jsToWasmICCallee->function()->previousInstance(m_callFrame), m_vm.softStackLimit());
  }

  if (m_catchableFromWasm && callee.isWasm())
  {
    Wasm::Callee *wasmCallee = callee.asWasmCallee();
    if (wasmCallee->hasExceptionHandlers())
    {
      JSWebAssemblyInstance *jsInstance = jsCast<JSWebAssemblyInstance *>(m_callFrame->thisValue());
      unsigned exceptionHandlerIndex = m_callFrame->callSiteIndex().bits();
      m_handler = {wasmCallee->handlerForIndex(jsInstance->instance(), exceptionHandlerIndex, m_wasmTag), wasmCallee};
      if (m_handler.m_valid)
        return IterationStatus::Done;
    }
  }
#endif
  // [...]
}
```

[2]

➢ src: JavaScriptCore/interpreter/Interpreter.cpp

CVE-2022-32863

Patch

- Setup wasm stack |this| first if there is exception handler in wasm code

```
diff --git a/Source/JavaScriptCore/wasm/WasmAirIRGenerator.cpp b/Source/JavaScriptCore/wasm/WasmAirIRGenerator.cpp
index 5255b8d71e21..53ca908b4949 100644
--- a/Source/JavaScriptCore/wasm/WasmAirIRGenerator.cpp
+++ b/Source/JavaScriptCore/wasm/WasmAirIRGenerator.cpp
@@ -1014,14 +1014,21 @@ AirIRGenerator::AirIRGenerator(const ModuleInformation& info, B3::Procedure& pro
             bool needUnderflowCheck = static_cast<unsigned>(checkSize) > Options::reservedZoneSize();
             bool needsOverflowCheck = m_makesCalls || wasmFrameSize >= static_cast<int32_t>(minimumParentCheckSize) ||
needUnderflowCheck;

+           if ((needsOverflowCheck || m_usesInstanceValue) && Context::useFastTLS())
+               jit.loadWasmContextInstance(m_prologueWasmContextGPR);
+
+           // We need to setup JSWebAssemblyInstance in |this| slot before checking stack overflow. Otherwise, we
will fail to get it when unwinding
+           // if we throw an error from the stack overflow check.
+           if (m_catchEntrypoints.size()) {
+               GPRReg scratch = wasmCallingConvention().prologueScratchGPRs[0];
+               jit.loadPtr(CCallHelpers::Address(m_prologueWasmContextGPR, Instance::offsetOfOwner()), scratch);
+               jit.store64(scratch, CCallHelpers::Address(GPRInfo::callFrameRegister, CallFrameSlot::thisArgument *
sizeof(Register)));
+           }
+
            // This allows leaf functions to not do stack checks if their frame size is within
            // certain limits since their caller would have already done the check.
            if (needsOverflowCheck) {
                GPRReg scratch = wasmCallingConvention().prologueScratchGPRs[0];
                // […]
                });
            }
        }
    });
```

➢ Fix: e49129f82a16a9b5ef9c951c1743a33057d07d12

CVE-2022-32885

- Vulnerability exists in the LLInt Parser.
- Inappropriate implementation on parsing delegate bytecode
- Break the stack frame balance/StackOverflow

CVE-2022-32885

LLInt Parser Overview

- **Wasm function :** Highly-structured
- **m_expressionStack :** Track the value of expressions
- **m_controlStack :** Stack of expression stacks
- **enclosedExpressionStack :** Store parsed expressions

CVE-2022-32885

Unreachable in Wasm

- The code located behind **Br/Brtable/Return**

```
(module
  (func (export "add") (param i32 i32) (result i32)
    local.get 0
    local.get 1
    i32.add
    block (param i32) (result i32)
      i32.const 1337
      i32.add
    end
    return
    i32.const 0xdeadbeef
    i32.add
    return
  )
)
```

CVE-2022-32885

Delegate in Wasm

- **Delegate(label_x) :**
  handle over exception handling to label_x

```
(module
  try $l0
    try
    call $foo
  delegate $l0  ;; (= delegate 0)
catch
  ...
catch_all
  ...
end
)
```

➢ Ref: wasm exception-handling

CVE-2022-32885

Vulnerability analysis

- parseUnreachableExpression: parse expressions when encountering unreachable blocks

```
template<typename Context>
auto FunctionParser<Context>::parseUnreachableExpression() -> PartialResult
{
    ASSERT(m_unreachableBlocks);
#define CREATE_CASE(name, ...) case OpType::name:
    switch (m_currentOpcode) {
      // [...]
      case End: {
        if (m_unreachableBlocks == 1) {
            ControlEntry data = m_controlStack.takeLast();
            if (ControlType::isIf(data.controlData)) {
                WASM_TRY_ADD_TO_CONTEXT(addElseToUnreachable(data.controlData));
                m_expressionStack = WTFMove(data.elseBlockStack);
                WASM_FAIL_IF_HELPER_FAILS(unify(data.controlData));
                WASM_TRY_ADD_TO_CONTEXT(endBlock(data, m_expressionStack));
            } else {
                Stack emptyStack;
                WASM_TRY_ADD_TO_CONTEXT(addEndToUnreachable(data, emptyStack));
            }

            m_expressionStack.swap(data.enclosedExpressionStack);
        }
        m_unreachableBlocks--;
        return { };
    }
    // [...]
    }
}
```

➢ src: JavaScriptCore/wasm/WasmFunctionParser.h

CVE-2022-32885

Vulnerability analysis

- Delegate operator should be handled the same way as the End operator :

  try … end

  v.s.

  try … delegate x

```
template <typename Context>
auto FunctionParser<Context>::parseUnreachableExpression() -> PartialResult
{
  ASSERT(m_unreachableBlocks);
#define CREATE_CASE(name, ...) case OpType::name:
  switch (m_currentOpcode)
  {
  // [...]
  case Delegate:
  {
    WASM_PARSER_FAIL_IF(!Options::useWebAssemblyExceptions(), "wasm exceptions are not enabled");

    WASM_PARSER_FAIL_IF(m_controlStack.size() == 1, "can't use delegate at the top-level of a function");

    uint32_t target;
    WASM_FAIL_IF_HELPER_FAILS(parseBranchTarget(target));

    ControlEntry controlEntry = m_controlStack.takeLast();
    WASM_VALIDATOR_FAIL_IF(!ControlType::isTry(controlEntry.controlData), "delegate isn't associated to a
try");

    ControlType &data = m_controlStack[m_controlStack.size() - 1 - target].controlData;
    WASM_VALIDATOR_FAIL_IF(!ControlType::isTry(data) && !ControlType::isTopLevel(data), "delegate target is-
n't a try block");

    WASM_TRY_ADD_TO_CONTEXT(addDelegateToUnreachable(data, controlEntry.controlData));
    Stack emptyStack;
    WASM_TRY_ADD_TO_CONTEXT(addEndToUnreachable(controlEntry, emptyStack));
    m_expressionStack.swap(controlEntry.enclosedExpressionStack);
    return {};
  }
    // [...]
  }
}
```

➢ src: JavaScriptCore/wasm/WasmFunctionParser.h

CVE-2022-32885

POC

- Add a Delegate statement after unreachable code.

```
(module
  (type $t0 (func (param i32 i32 i32) (result i32)))
  (type $t1 (func))
  (func $main (export "main") (type $t0) (param $p0 i32) (param $p1 i32) (param $p2 i32) (result i32)
    (local $l3 f64)
    (try  ;; label = @1
      (do
        (try  ;; label = @2
          (do
            (try  ;; label = @3
              (do
                (drop
                  (call $main
                    (i32.mul
                      (i32.const 0)
                      (i32.const 0))
                    (i32.const 0)
                    (i32.const 0))))
              (catch $e0)
              (catch_all))
            (br 0 (;@2;)))
          (delegate 0)))
      (catch_all))
    (i32.const 0))
  (table $T0 1 2 funcref)
  (memory $M0 16 32)
  (tag $e0 (type $t1))
  (global $g0 (mut i64) (i64.const 0))
  (elem $e0 (i32.const 0) func $main))
```

[1]

[2]

CVE-2022-32885

Vulnerability analysis

- parseUnreachableExpression was mistakenly used while parsing the CatchAll operator

```
template<typename Context>
auto FunctionParser<Context>::parseBody() -> PartialResult
{
    m_controlStack.append({ { }, { }, m_context.addTopLevel(&m_signature) });
    uint8_t op = 0;
    while (m_controlStack.size()) {
        // [...]
        if (m_unreachableBlocks)
            WASM_FAIL_IF_HELPER_FAILS(parseUnreachableExpression());
        else {
            WASM_FAIL_IF_HELPER_FAILS(parseExpression());
        }
    }
    WASM_FAIL_IF_HELPER_FAILS(m_context.endTopLevel(&m_signature, m_expressionStack));

    ASSERT(op == OpType::End);
    return { };
}
```

➢ src: JavaScriptCore/wasm/WasmFunctionParser.h

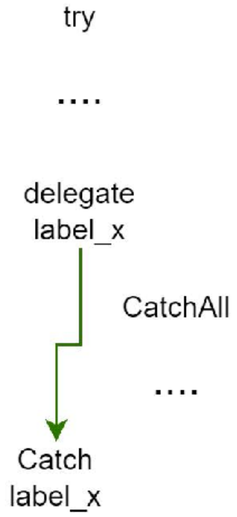
CVE-2022-32885

Vulnerability analysis

- In normal case, a pass-jmp instruction will be emitted

```
template <typename Context>
auto FunctionParser<Context>::parseExpression() -> PartialResult
{
  switch (m_currentOpcode)
  {
  // [...]
  case CatchAll:
  {
    // [...]

    ResultList results;
    Stack preCatchStack;
    m_expressionStack.swap(preCatchStack);
    WASM_TRY_ADD_TO_CONTEXT(addCatchAll(preCatchStack, controlEntry.con-
trolData));
    return {};
  }
    // [...]
  }
}
auto LLIntGenerator::addCatchAll(Stack& expressionStack, ControlType& data) ->
PartialResult
{
    finalizePreviousBlockForCatch(data, expressionStack);
    WasmJmp::emit(this, data.m_continuation->bind(this));
    return addCatchAllToUnreachable(data);
}
```



➢ src: JavaScriptCore/wasm/WasmFunctionParser.h

CVE-2022-32885

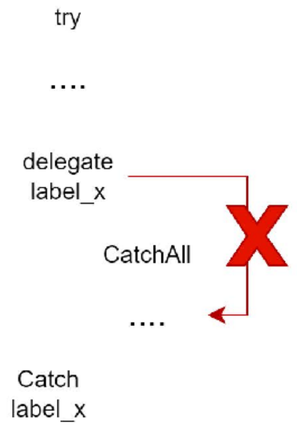
Vulnerability analysis

- Under the vulnerability, it won't emit such instruction and will fall into CatchAll's handler code directly

```
template <typename Context>
auto FunctionParser<Context>::parseUnreachableExpression() -> PartialResult
{
  ASSERT(m_unreachableBlocks);
#define CREATE_CASE(name, ...) case OpType::name:
  switch (m_currentOpcode)
  {
  // [...]
  case CatchAll:
  {
    WASM_PARSER_FAIL_IF(!Options::useWebAssemblyExceptions(), "wasm exceptions are not enabled");

    if (m_unreachableBlocks > 1)
      return {};

    ControlEntry &data = m_controlStack.last();
    m_unreachableBlocks = 0;
    m_expressionStack = {};
    WASM_VALIDATOR_FAIL_IF(!isTryOrCatch(data.controlData), "catch block isn't associated to a
try");
    WASM_TRY_ADD_TO_CONTEXT(addCatchAllToUnreachable(data.controlData));
    return {};
  }
    // [...]
  }
}
```



➢ src: JavaScriptCore/wasm/WasmFunctionParser.h

CVE-2022-32885

Patch

- Handle control stack in the same way as the End opcode
- Decrement m_unreachableBlocks in the same way as the End opcode

```
diff --git a/Source/JavaScriptCore/wasm/WasmFunctionParser.h b/Source/JavaScriptCore/wasm/WasmFunctionParser.h
index f74c800c923f..edad1bb01f6b 100644
--- a/Source/JavaScriptCore/wasm/WasmFunctionParser.h
+++ b/Source/JavaScriptCore/wasm/WasmFunctionParser.h

    FunctionParser(Context&, const uint8_t* functionStart, size_t functionLength, const TypeDefinition&, const ModuleInformation& );
@@ -1728,16 +1729,19 @@ auto FunctionParser<Context>::parseUnreachableExpression() -> PartialResult
        uint32_t target;
        WASM_FAIL_IF_HELPER_FAILS(parseBranchTarget(target));

-       ControlEntry controlEntry = m_controlStack.takeLast();
-       WASM_VALIDATOR_FAIL_IF(!ControlType::isTry(controlEntry.controlData), "delegate isn't associated to a try");
+       if (m_unreachableBlocks == 1) {
+           ControlEntry controlEntry = m_controlStack.takeLast();
+           WASM_VALIDATOR_FAIL_IF(!ControlType::isTry(controlEntry.controlData), "delegate isn't associated to a try");

-       ControlType& data = m_controlStack[m_controlStack.size() - 1 - target].controlData;
-       WASM_VALIDATOR_FAIL_IF(!ControlType::isTry(data) && !ControlType::isTopLevel(data), "delegate target isn't a try block");
+           ControlType& data = m_controlStack[m_controlStack.size() - 1 - target].controlData;
+           WASM_VALIDATOR_FAIL_IF(!ControlType::isTry(data) && !ControlType::isTopLevel(data), "delegate target  isn't a try block");

-       WASM_TRY_ADD_TO_CONTEXT(addDelegateToUnreachable(data, controlEntry.controlData));
-       Stack emptyStack;
-       WASM_TRY_ADD_TO_CONTEXT(addEndToUnreachable(controlEntry, emptyStack));
-       m_expressionStack.swap(controlEntry.enclosedExpressionStack);
+           WASM_TRY_ADD_TO_CONTEXT(addDelegateToUnreachable(data, controlEntry.controlData));
+           Stack emptyStack;
+           WASM_TRY_ADD_TO_CONTEXT(addEndToUnreachable(controlEntry, emptyStack));
+           m_expressionStack.swap(controlEntry.enclosedExpressionStack);
+       }
+       m_unreachableBlocks--;
        return { };
    }
```

➢ Fix: 27d302ab3481407e8746f57729f0961058235e33

CVE-2022-32886

- BBQ & OMG
- Uninitialized value in callSiteIndex
- Wrong exception handler

CVE-2022-32886

Callsite Index

- Used for exception handler
- Store the position of Call/Try/Catch/Throw
- StackMap : keep callsite_index and used_values key-value pair

CVE-2022-32886

Vulnerability analysis

- The handle is initialized only when the call instruction is enclosed within a try-catch block

[1]

```
PatchpointExceptionHandle B3IRGenerator::preparePatchpointForExceptions(BasicBlock* block, PatchpointValue* patch)
{
    ++m_callSiteIndex;
    if (!m_tryCatchDepth)
        return { };

    Vector<Value*> liveValues;
    Origin origin = this->origin();
    for (Variable* local : m_locals) {
        Value* result = block->appendNew<VariableValue>(m_proc, B3::Get, origin, local);
        liveValues.append(result);
    }
    for (unsigned controlIndex = 0; controlIndex < m_parser->controlStack().size(); ++controlIndex) {
        ControlData& data = m_parser->controlStack()[controlIndex].controlData;
        Stack& expressionStack = m_parser->controlStack()[controlIndex].enclosedExpressionStack;
        for (Variable* value : expressionStack)
            liveValues.append(get(block, value));
        if (ControlType::isAnyCatch(data))
            liveValues.append(get(block, data.exception()));
    }

    patch->effects.exitsSideways = true;
    patch->appendVectorWithRep(liveValues, ValueRep::LateColdAny);

    return PatchpointExceptionHandle { m_callSiteIndex, static_cast<unsigned>(liveValues.size()) };
}
```

[2]

➢ src: JavaScriptCore/wasm/WasmB3IRGenerator.cpp

CVE-2022-32886

Vulnerability analysis

- BBQ and OMG will ultimately call the generate function to get the optimized code
- The generate function omits the storing operation. The callsite index isn't kept neither in slot nor in stack map

[3]

[4]

```
struct PatchpointExceptionHandle {
    template <typename Generator>
    void generate(CCallHelpers& jit, const B3::StackmapGenerationParams& params, Generator* generator) const
    {
        if (m_callSiteIndex == s_invalidCallSiteIndex)
            return;

        StackMap values(m_numLiveValues);
        unsigned paramsOffset = params.size() - m_numLiveValues;
        unsigned childrenOffset = params.value()->numChildren() - m_numLiveValues;
        for (unsigned i = 0; i < m_numLiveValues; ++i)
            values[i] = OSREntryValue(params[i + paramsOffset], params.value()->child(i + childrenOffset)->type());

        generator->addStackMap(m_callSiteIndex, WTFMove(values));
        jit.store32(CCallHelpers::TrustedImm32(m_callSiteIndex), CCallHelpers::tagFor(CallFrameSlot::argumentCountIn-
cludingThis));
    }

    static constexpr unsigned s_invalidCallSiteIndex = std::numeric_limits<unsigned>::max();

    unsigned m_callSiteIndex { s_invalidCallSiteIndex };
    unsigned m_numLiveValues;
}
```

➢ src: JavaScriptCore/wasm/WasmIRGeneratorHelpers.h

CVE-2022-32886

Vulnerability analysis

- This results in an incorrect handler due to the presence of a dirty value

```
IterationStatus operator()(StackVisitor &visitor) const
{
  // [...]

#if ENABLE(WEBASSEMBLY)
  CalleeBits callee = visitor->callee();
  if (callee.isCell())
  {
    if (auto *jsToWasmICCallee = jsDynamicCast<JSToWasmICCallee *>(callee.asCell()))
      m_vm.wasmContext.store(jsToWasmICCallee->function()->previousInstance(m_callFrame), m_vm.softStackLimit());
  }

  if (m_catchableFromWasm && callee.isWasm())
  {
    Wasm::Callee *wasmCallee = callee.asWasmCallee();
    if (wasmCallee->hasExceptionHandlers())
    {
      JSWebAssemblyInstance *jsInstance = jsCast<JSWebAssemblyInstance *>(m_callFrame->thisValue());
      unsigned exceptionHandlerIndex = m_callFrame->callSiteIndex().bits();
      m_handler = {wasmCallee->handlerForIndex(jsInstance->instance(), exceptionHandlerIndex, m_wasmTag), wasmCallee};
      if (m_handler.m_valid)
        return IterationStatus::Done;
    }
  }
#endif
  // [...]
}
```

[5]

➢ src: JavaScriptCore/interpreter/Interpreter.cpp

CVE-2022-32886

Vulnerability analysis

- Before executing error-handling code, the stack map is obtained by calling the buildEntryBufferForCatch function.

```
static inline void buildEntryBufferForCatch(Probe::Context& context)
{
    CallFrame* callFrame = context.fp<CallFrame*>();
    CallSiteIndex callSiteIndex = callFrame->callSiteIndex();
    OptimizingJITCallee* callee = bitwise_cast<OptimizingJIT-
Callee*>(callFrame->callee().asWasmCallee());
    const StackMap& stackmap = callee->stackmap(callSiteIndex);
    VM* vm = context.gpr<VM*>(GPRInfo::regT0);
    uint64_t* buffer = vm->wasmContext.scratchBufferForSize(stackmap.size() * 8);
    loadValuesIntoBuffer(context, stackmap, buffer);

    context.gpr(GPRInfo::argumentGPR0) = bitwise_cast<uintptr_t>(buffer);
}
```

➢ src: JavaScriptCore/wasm/WasmIRGeneratorHelpers.h

CVE-2022-32886

Vulnerability analysis

- Since the key-value pair was not stored, an assert failure is triggered

```
const StackMap& OptimizingJITCallee::stackmap(CallSiteIndex callSiteIndex) const
{
    auto iter = m_stackmaps.find(callSiteIndex);
    if (iter == m_stackmaps.end()) {
        for (auto pair : m_stackmaps) {
            dataLog(pair.key.bits(), ": ");
            for (auto value : pair.value)
                dataLog(value, ", ");
            dataLogLn("");
        }
    }
    RELEASE_ASSERT(iter != m_stackmaps.end());
    return iter->value;
}
```

➢ src: JavaScriptCore/wasm/WasmCallee.cpp

CVE-2022-32886

Patch

- Store CallSiteIndex for calls in Air and B3 if there are exception handlers present

```
diff --git a/Source/JavaScriptCore/wasm/WasmIRGeneratorHelpers.h
b/Source/JavaScriptCore/wasm/WasmIRGeneratorHelpers.h
index bfb21da023ad..21d5eda6ed4e 100644
--- a/Source/JavaScriptCore/wasm/WasmIRGeneratorHelpers.h
+++ b/Source/JavaScriptCore/wasm/WasmIRGeneratorHelpers.h
@@ -40,11 +40,24 @@
 namespace JSC { namespace Wasm {

 struct PatchpointExceptionHandle {
     template <typename Generator>
     void generate(CCallHelpers& jit, const B3::StackmapGenerationParams&
params, Generator* generator) const
     {
-        if (m_callSiteIndex == s_invalidCallSiteIndex)
+        if (m_callSiteIndex == s_invalidCallSiteIndex) {
+            if (!m_hasExceptionHandlers || m_hasExceptionHandlers.value())
+                jit.store32(CCallHelpers::TrustedImm32(m_callSiteIndex),
CCallHelpers::tagFor(CallFrameSlot::argumentCountIncludingThis));
             return;
+        }

 };
```

➢ Fix: 97b340447c9fc3e934a8045e059b3fac71e0ab4d

Q & A

Thanks

Feel free to contact us at @P1umer and @xmzyshypnc1 in Twitter