CSCI 599: Software Engineering for Security

Project Update 1

March 22

Group II
Sara Baradaran, Jungkyu Kim, Ritu Pravakar, Yutian Yan

Vulnerability Study in WebAssembly Runtimes/Virtual Machines

Presentation Outline

1. Project Overview

2. Selected Runtimes

3. CVE Analysis & Reproduction

4. Next Step

Project Overview

- The goal of this project is to analyze, categorize, and reproduce vulnerabilities detected in WebAssembly runtimes
- Comprehensively understanding detected vulnerabilities can help developers know vulnerable code patterns and functions/APIs and avoid them when developing analogous software
- Since many vulnerabilities may share similar root causes, understanding root causes for known vulnerabilities may reveal undiscovered vulnerabilities
- Also, finding vulnerable products and the root causes of vulnerabilities can help developers to decide which products should be used for a specific purpose
    - For example, if a runtime does not validate the input WebAssembly files, the products that internally use it must validate the codes before passing them into the runtime

Preliminary Study

- We have collected a set of 114 CVEs related to WebAssembly from 2016 to 2024 using keywords, such as WebAssembly, Wasm, etc.
- We have collected dataset from four sources
  - www.nist.gov, cve.mitre.org, www.cvedetails.com, www.exploit-db.com
- We have extracted CVEs features such as CVE-ID, CWE (type of weakness), severity scores, vulnerable product name and its version, descriptions, vulnerability patch, vulnerable code snippets, commits related to vulnerability fix
- Most of the vulnerabilities are related to a specific group of products, for example there are 15 CVEs for Wasmtime, but there is only one for Wasmer
- Most of the severity scores are medium and high, there are a few number of critical and low severity scores

Data Sample

1	CVE ID	CWE	CNA: SCORE	NIST: SCORE	PRODUCT	LINK	DESCRIPTION
15	CVE-2022-31146	CWE-416: Use After Free	6.4 MEDIUM	8.8 HIGH	wasmtime	Link	There is a bug in the Wasmtime's code generator, Cranelift, where functio runtime garbage collection. This means that if a GC happens at runtime th to GC'd values, reclaiming them and deallocating them. The function will t leading later to a use-after-free. This bug was introduced in the migration t on 2022-05-20. This bug has been patched and users should upgrade to V the reference types proposal by passing `false` to `wasmtime::Config::was
16	CVE-2022-31104	CWE-682: Incorrect Calculation	4.8 MEDIUM	5.6 MEDIUM	wasmtime	Link	In affected versions wasmtime's implementation of the SIMD proposal for ' implemented in Cranelift. The aarch64 implementation of the simd propos WebAssembly instructions. The `select` instruction is only affected when t were `swizzle` and `select`. The `swizzle` instruction lowering in Cranelift value, for example. This means that future uses of the same constant may Cranelift wasn't correctly implemented for vector types that are 128-bits wi correct input to the output of the instruction meaning that only the low 32 b register previously contained (instead of the input being moved from). The bug in Wasmtime's implementation of these instructions on x86_64 repres according to the WebAssembly specification. The impact of this is benign execution of a guest program. For example a WebAssembly program coul the risk of exposing the program itself to other related vulnerabilities which cranelift-codegen (and other associated cranelift crates) 0.85.1 which cont upgrading is not an option for you at this time, you can avoid the vulnerabi x86_64 hosts. Other aarch64 hosts are not affected. Note that s390x hosts
17	CVE-2023-52284	CWE-415: Double Free	N/A	5.5 MEDIUM	WAMR	Link	Bytecode Alliance wasm-micro-runtime (aka WebAssembly Micro Runtime WebAssembly module because push_pop_frame_ref_offset is mishandlec
18	CVE-2023-48105	CWE-787: Out-of-bounds Write	N/A	7.5 HIGH	WAMR	Link	An heap overflow vulnerability was discovered in Bytecode alliance wasm- the wasm_loader_prepare_bytecode function in core/iwasm/interpreter/wa
19	CVE-2023-51661	CWE-284: Improper Access Control	8.6 HIGH	8.6 HIGH	Wasmer	Link	Wasmer is a WebAssembly runtime that enables containers to run anywhe can access the filesystem outside of the sandbox. Service providers runni filesystem. This vulnerability has been patched in version 4.2.4.

[Link to CVEs dataset]

Analyzed Runtimes

- **Wasm3**: A lightweight and efficient WebAssembly runtime that focuses on quick startup times. It is suitable for resource-constrained environments and is used by different products such as wasmcloud (A platform for writing portable business logic) and WowCube (A console and the gaming platform)
- **Wasmtime**: A fast and secure runtime for WebAssembly that supports multiple programming languages and provides a standalone command-line interface. It is developed by Bytecode Alliance and supports JIT (Just-In-Time) compilation
- Special: Non-runtimes
  - **Binaryen**: provides a suite of compiler and toolchain components for WebAssembly development, offering optimizations, code generation, and transformation capabilities
  - **WABT**: an open-source suite of tools and libraries for working with WebAssembly, which provides a range of utilities and components, including binary manipulation, validation, and disassembly

Wasmtime Vulnerabilities Overview

- There exist 15 CVEs for known vulnerabilities detected in Wasmtime
- Wasmtime vulnerabilities have detailed description and commits that patch the vulnerabilities are clearly identified
- Almost half of the bugs (6 CVEs) originate from Wasmtime's code generator, Cranelift
- Many vulnerabilities share the same root causes such as erroneously handling zero-extension of addresses
- Known vulnerabilities are mostly fixed by developers. Most vulnerabilities are no longer alive in the latest version of Wasmtime

Wasmtime Vulnerabilities Root Causes

- Handling support of externrefs is a common root cause of 5 CVEs in Wasmtime
- WebAssembly supports a value type called externref that aims to enable more efficient communication between the host (JS) and the wasm module
- Without the reference types, to take and return references to the host's objects, cooperating glue code on the host side is required. Using large glue code causes WebAssembly to become less attractive as a universal binary format
- The proposal for reference types has three main parts; First, a new externref type representing an opaque, unforgeable reference to a host object. Second, an extension to WebAssembly tables, allowing them to hold externrefs in addition to function references. Third, new instructions for manipulating tables and their entries
- With these new capabilities, Wasm modules can talk about host references directly, rather than requiring external glue code running in the host

Challenges for Handling Reference Types

- The externrefs are opaque meaning that a Wasm module cannot observe an externref value's bit pattern
- Since externrefs are completely opaque from the module perspective, the only way to use them is to send an externref back to the host as an argument of an imported function. Passing a reference to a host object into a Wasm module also does not reveal any information about the host's address space and the layout of host objects within it
- The externrefs are also unforgeable, meaning that a Wasm module cannot create a fake host reference. The Wasm module cannot pretend an arbitrary integer value is a valid host reference
- An externref cannot be dereferenced by the module, thus, the module cannot directly access or modify the data behind the reference. It can only return either a reference you already gave it or a null reference

CVE Examples with Common Root Causes

- **CVE-2021-39216:** To trigger the bug, you have to explicitly pass multiple externrefs from the host to a Wasm instance at the same time, either by passing multiple externrefs as arguments from host code to a Wasm function, or returning multiple externrefs to Wasm from a multi-value return function defined in the host
- **CVE-2021-39218**: To trigger this bug, Wasmtime needs to be running Wasm that uses externrefs, the host creates non-null externrefs, Wasmtime performs a garbage collection (GC), and there has to be a Wasm frame on the stack that is at a GC safepoint where there are no live references at this safepoint, and there is a safepoint with live references earlier in this frame's function
- **CVE-2022-23636**: There exists a bug in the pooling instance allocator in Wasmtime where a failure to instantiate an instance for a module that defines an externref will result in an invalid drop of a VMExternRef via an uninitialized pointer

CVE Examples with Common Root Causes

- **CVE-2022-24791**: There is a use after free vulnerability in Wasmtime when both running Wasm that uses externrefs and enabling epoch interruption in Wasmtime
- **CVE-2022-31146**: There is a bug in the Wasmtime's code generator, Cranelift, where functions that use reference types may be incorrectly missing metadata required for runtime garbage collection. This means that if a GC happens at runtime then the GC pass will mistakenly think these functions do not have live references to GC'd values, reclaiming them and deallocating them. The function will then subsequently continue to use the values assuming they had not been GC'd, leading later to a use-after-free
- All 5 vulnerabilities can be avoided by disabling reference types through passing *false* to wasmtime::Config::wasm_reference_types

Wasmtime Vulnerabilities Root Causes

- Implementing SIMD proposal is another root cause of 3 CVEs in Wasmtime
- SIMD stands for *single instruction, multiple data*. SIMD instructions are a special class of instructions that exploit data parallelism in applications by simultaneously performing the same operation on multiple data elements
- Computationally intensive applications such as audio/video codecs and image processors take advantage of SIMD instructions to accelerate performance
- The high-level goal of the WebAssembly SIMD proposal is to introduce vector operations to the WebAssembly specification, in a way that guarantees portable performance. It defines a portable, performant subset of SIMD operations that are available across most modern architectures
- The current proposal is limited to standardizing fixed-width 128-bit SIMD operations. It introduces a new v128 value type, used to represent different types of packed data, and a number of new operations that operate on this type (on packed data)

CVE Examples with Common Root Causes

- **CVE-2022-31104**: Wasmtime's implementation of the SIMD proposal for Wasm on x86_64 contained two distinct bugs in the instruction lowerings implemented in Cranelift. The bugs were presented in the *i8x16.swizzle* and *select* WebAssembly instructions
- **CVE-2023-41880**: Contains a miscompilation of the WebAssembly *i64x2.shr_s* instruction on x86_64 platforms when the shift amount is a constant value that is larger than 32. The primary impact of this issue is that any WebAssembly programs that use the *i64x2.shr_s* with a constant shift amount larger than 32 may produce an incorrect result
- **CVE-2023-27477**: Wasmtime's code generation backend, Cranelift, has a bug on x86_64 platforms for the WebAssembly *i8x16.select* instruction which will produce the wrong results when the same operand is provided to the instruction and some of the selected indices are greater than 16

Wasm3 Vulnerabilities Overview

- There exist 9 CVEs for known vulnerabilities detected in Wasm3. We also found 3 more vulnerabilities which are not assigned CVE IDs (unofficial ones)
- Unlike Wasmtime, Wasm3 vulnerabilities do not have detailed description and commits that patch the vulnerabilities are not clearly identified. In the NVD website, each CVE is described in a single line
- All 12 vulnerabilities can be classified into two categories: 4 detected by oss-fuzz and reported by developers of this tool, and 8 reported by independent users
- Almost all vulnerabilities share the same root cause. Currently, Wasm3 assumes that input wasm files are validated. It performs some checks, but the Wasm3 validation process is incomplete. As a result, Wasm3 gets several fuzzer errors, the majority of which are produced by invalid WebAssembly inputs

Incomplete input validation in Wasm3

An example for invalid WebAssembly input that results in a crash in Wasm3.

```
sara@sara:~/WasmRuntimes/wasm3/build$ ./wasm3 --version
Wasm3 v0.5.0 on x86_64
Build: Mar 21 2024 23:18:42, GCC 11.4.0
sara@sara:~/WasmRuntimes/wasm3/build$ ./wasm3 ../../op_Select_i32_srs.wasm
AddressSanitizer:DEADLYSIGNAL
=================================================================
==3595928==ERROR: AddressSanitizer: SEGV on unknown address 0x630efbcab954 (pc 0x55b3bef284f4 bp 0x7ffced1363e0 sp 0x7ffced1363a0 T0)
==3595928==The signal is caused by a READ memory access.
    #0 0x55b3bef284f4 in op_Select_i32_srs /home/sara/WasmRuntimes/wasm3/source/m3_exec.h:1075
    #1 0x55b3bef15ff9 in op_f64_Ceil_s /home/sara/WasmRuntimes/wasm3/source/m3_exec.h:269
    #2 0x55b3bef1286c in op_i32_Divide_rs /home/sara/WasmRuntimes/wasm3/source/m3_exec.h:228
    #3 0x55b3bef2b876 in op_f32_Load_f32_s /home/sara/WasmRuntimes/wasm3/source/m3_exec.h:1338
    #4 0x55b3bef36ca7 in op_i32_Store_i32_ss /home/sara/WasmRuntimes/wasm3/source/m3_exec.h:1446
    #5 0x55b3bef26e6c in op_SetSlot_i32 /home/sara/WasmRuntimes/wasm3/source/m3_exec.h:938
    #6 0x55b3bef25ad5 in op_MemGrow /home/sara/WasmRuntimes/wasm3/source/m3_exec.h:701
    #7 0x55b3bef171b8 in op_i32_EqualToZero_s /home/sara/WasmRuntimes/wasm3/source/m3_exec.h:279
    #8 0x55b3bef2660e in op_Entry /home/sara/WasmRuntimes/wasm3/source/m3_exec.h:805
    #9 0x55b3beef6133 in Call /home/sara/WasmRuntimes/wasm3/source/m3_exec.h:114
    #10 0x55b3beefbee9 in m3_CallArgv /home/sara/WasmRuntimes/wasm3/source/m3_env.c:923
    #11 0x55b3bee6e538 in repl_call /home/sara/WasmRuntimes/wasm3/platforms/app/main.c:274
    #12 0x55b3bee7137f in main /home/sara/WasmRuntimes/wasm3/platforms/app/main.c:625
    #13 0x7fd38c029d8f in __libc_start_call_main ../sysdeps/nptl/libc_start_call_main.h:58
    #14 0x7fd38c029e3f in __libc_start_main_impl ../csu/libc-start.c:392
    #15 0x55b3bee6d214 in _start (/home/sara/WasmRuntimes/wasm3/build/wasm3+0x2e214)

AddressSanitizer can not provide additional info.
SUMMARY: AddressSanitizer: SEGV /home/sara/WasmRuntimes/wasm3/source/m3_exec.h:1075 in op_Select_i32_srs
==3595928==ABORTING
sara@sara:~/WasmRuntimes/wasm3/build$
```

Given WebAssembly input

```
asm
``#Tasi_snapshrt_preèiE1f˘˜ritt
_start
*(
```

Wasm3 Vulnerabilities (Analysis & Reproduction)

- We analyzed all 12 vulnerabilities including those that are not assigned CVE IDs
- We also tried to reproduce 8 vulnerabilities reported by independent users by using the PoC (Proof of Concept) files provided by the people who detected them
- Out of 8 vulnerabilities, we could exactly reproduce 4 vulnerabilities still alive in the latest version of Wasm3. We also reproduced 1 fixed vulnerability by using previous versions of Wasm3. For 3 remaining, we observed crashes when executing PoCs, but the crashes are not exactly the same as those reported
- Out of 12, only one reported CVE has been fixed in Wasm3 and the corresponding commit for fixing it is clearly identified. There is no other commit clearly identified as a patch for a legacy vulnerability
- The errors reported in GitHub issues are quickly closed by developers and ignored. They claim that since the input validation is not complete, the fuzzer errors have no place for discussion and analysis

Wasm3 Fixed CVEs

The only CVE that is fixed in Wasm3 is CVE-2022-28990 detected in WASI API, which uses iovs without checking the iovs buffer length and the buffer address. To fix it, the function `m3ApiCheckMem(addr,len)` is used to properly manage iov buffers.

In WebAssembly, `iov` stands for "I/O Vector" and refers to a mechanism used to efficiently transfer data between the WebAssembly module and the host environment.

```
39  39    static inline
40      - void copy_iov_to_host(void* _mem, __wasi_iovec_t* host_iov, __wasi_iovec_t*
          wasi_iov, int32_t iovs_len)
      40  + const void* copy_iov_to_host(IM3Runtime runtime, void* _mem, __wasi_iovec_t*
          host_iov, __wasi_iovec_t* wasi_iov, int32_t iovs_len)
41  41    {
42  42        // Convert wasi memory offsets to host addresses
43  43        for (int i = 0; i < iovs_len; i++) {
44  44            host_iov[i].buf = m3ApiOffsetToPtr(wasi_iov[i].buf);
45  45            host_iov[i].buf_len  = wasi_iov[i].buf_len;
      46  +         m3ApiCheckMem(host_iov[i].buf,     host_iov[i].buf_len);
46  47        }
      48  +     m3ApiSuccess();
47  49    }
```

Wasm3 CVE Reproduction

Since CVE-2022-28990 is fixed in the latest version of Wasm3, when we tried to reproduce it using v0.5.0, instead of crash the error is trapped.

```
sara@sara:~/WasmRuntimes/wasm3/build$ ./wasm3 --version
Wasm3 v0.5.0 on x86_64
Build: Mar 21 2024 18:19:53, GCC 11.4.0
sara@sara:~/WasmRuntimes/wasm3/build$ ./wasm3 ../../poc.wasm
Error: [trap] out of bounds memory access
```

We could reproduce the vulnerability by using versions before v0.4.9, where buffer overflow results in memory leakage.

```
sara@sara:~/WasmRuntimes/wasm3/build$ ./wasm3 ../../poc.wasm
 ?@?G?UP??G?U??G?Ux?FI?Upy?G?Upy?G?Upy?G?U0?G?U0?G?U0?G?U
                                                        0?G?U
???G?U??GI?U
?A?G?Uq?Result: <Empty Stack>
sara@sara:~/WasmRuntimes/wasm3/build$ ./wasm3 --version
Wasm3 v0.4.7 (Mar 21 2024 18:46:06, GCC 11.4.0, x86_64)
sara@sara:~/WasmRuntimes/wasm3/build$
```

```
sara@sara:~/WasmRuntimes/wasm3/build$ ./wasm3 ../../poc.wasm
=================================================================
==3589473==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x631000024818 at pc 0x7ff1d763fe57 bp 0x7fff9d22dad0 sp 0x7fff9d22d288
READ of size 45056 at 0x631000024818 thread T0
```

Non-Runtime CVEs

Although some CVEs are reported from Wasm compilation toolchain, Wasm tools, and Wasm decompilers, part of these CVEs are related to Wasm code execution and we also analyzed them.

- Binaryen
  - Optimizer and compiler/toolchain library for WebAssembly, part of Emscripten
  - 8 CVEs, 2021 to 2022
- WABT
  - The WebAssembly Binary Toolkit
  - 9 CVEs, 2022 to 2023

Binaryen - CVE-2021-46053

- Binaryen version 103, wasm-ctor-eval
- wasm-ctor-eval executes functions, or parts of them, at compile time
- Example

wasm-ctor-eval log:
trying to eval main
… partial evalling successful, but stopping since could not eval: call import: import.import
… stopping

Before:

```
(module
 (global $global (mut i32) (i32.const 0))
 (import "import" "import" (func $import))
 (func "main"
  (global.set $global (i32.const 1))
  (call $import)
  (global.set $global (i32.const 2)
)))
```

After:

```
(module
 (import "import" "import" (func $import))
 (global $global (mut i32) (i32.const 1))
 (export "main" (func $0_0))
 (func $0_0
  (call $import)
  (global.set $global (i32.const 2)
)))
```

CVE-2021-46053 (cont.)

- CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer
- Binaryen version 103, wasm-ctor-eval
- Given a fuzzer-generated program, wasm-ctor-eval will be terminated without any reasonable output and being killed because of OOM
- Reproducible: memory usage went to 100% and program terminated
- Not fixed: no related commits reported
- Reason: The fuzzer generated invalid programs, and wasm-ctor-eval did not validate the code before the execution
  - Node: report “SyntaxError: Invalid or unexpected token” and stops execution, no OOM
  - Wasm-decompile: report “error: unable to read u32 leb128: type count”

Other Binaryen Vulnerabilities

- All 8 Binaryen CVEs are found by fuzzing tests
- CVE-2021-46054: Reachable Assertion
  - When wasm-ctor-eval detects a crashed program, an assertion triggered and program terminated without any explanations
  - Fixed by replacing the assertion with throwing an error with error information
  - Such fix method also fixed CVE-2021-46048, CVE-2021-46052, CVE-2021-46055, CVE-2021-45290, and CVE-2021-45293
- CVE-2021-46055: Allocation of Resources Without Limits or Throttling
  - wasm-ctor-eval crashes after handling 6,625 nested try-catches
  - **Not fixed** because the developers “**don't know much** about stack overflow mitigations in native builds”

WABT Vulnerabilities

- WABT has a tool named **wasm-interp** that decode and run a WebAssembly binary file using a stack-based interpreter, and 5 CVEs are related to it
- CVE-2023-46331: Out-of-bounds Read
  - When wasm-interp tries to init the memory and checks whether the Data segment has a valid range, an out-of-bound memory read in DataSegment::IsValidRange()
  - Not fixed, and no related discussion
  - CVE-2023-46332 reports an out-of-bounds write in DataSegment::Drop(), same situation

WABT Vulnerabilities

- WABT has a tool named **wasm-interp** that decode and run a WebAssembly binary file using a stack-based interpreter, and 5 CVEs are related to it
- CVE-2022-43281: Out-of-bounds Write
  - Also CVE-2022-43280 and CVE-2022-43282, all fixed
  - We are still checking them and try to understand. So far:
  - wasm-interp has bugs in tail call implementation
  - Developer: “the tail-call proposal tests would have caught these … bugs ... But we're not running the tail-call proposal tests because (I guess?) the interpreter doesn't support tail calls yet”
  - However, the README mentions that wasm-interp should support tail call at that time
  - Possible reason: Tests deployed later than functionality

Next Step

- We are going to analyze the remaining CVEs in our dataset
- We intend to classifies the whole CVEs based on their common features and finds the most challenging WebAssembly proposals for the implementation from a security perspective
- We will also try to reproduce the remaining vulnerabilities

Thank You!