CSCI 599: Software Engineering for Security

Project Proposal

February 2

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

Vulnerability Study in WebAssembly Runtimes/Virtual Machines

Presentation Outline

1. Introduction & Motivation
2. Related Work
3. Preliminary Study
4. Future Plan

What is WebAssembly?

- Binary format executable for a stack-based virtual machine
- A portable compilation target for high-level languages (e.g., C/C++)
- Not designed for manually writing or reading
- Can be executed at a near-native speed

What is our goal in this project?

To comprehensively analyze vulnerabilities found in WebAssembly runtimes/virtual machines, including reproducing exploits, categorizing vulnerabilities, and providing suggestions for the community

What is a WebAssembly runtime?

WebAssembly runtime is the infrastructure for executing WebAssembly code, which is used as an execution engine by web browsers or non-web platforms

- Web runtimes: All major browsers (e.g, Chrome, Safari)
- Non-web (standalone) runtimes: wasmtime, wasmer, etc.

What is the consequence of vulnerability in runtimes?

The presence of vulnerability within WebAssembly runtimes not only leads to unexpected behavior of the Wasm code when executing but can also harm applications that rely on it.

- If a web application is getting exploited by attackers, it can cause serious damage to the users of that web application
- For non-web runtimes like wasmtime, vulnerabilities could break the control flow

What is our motivation?

- 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

CVE Example: CVE-2022-39394

- Weakness type: Out-of-bounds Write (CWE-787)
- CVSS severity: Critical (NVD score 9.8)
- CVE description: There is a vulnerability in Wasmtime's C API implementation where the definition of the **`wasmtime_trap_code`** does not match its declared signature in the **`trap.h`** header file.

`wasmtime_trap_code` declaration in `trap.h`

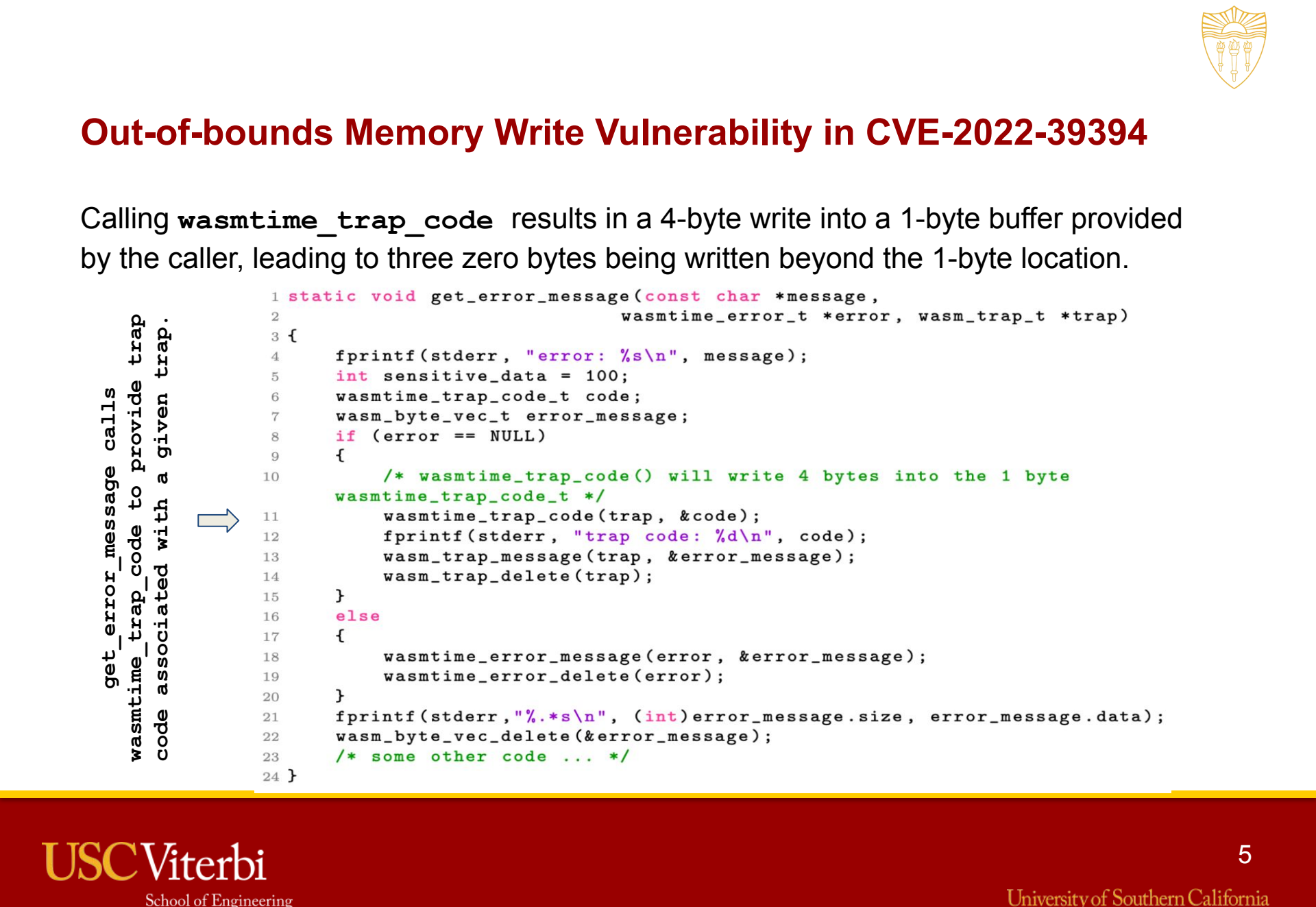
```c
typedef uint8_t wasmtime_trap_code_t;

WASM_API_EXTERN bool wasmtime_trap_code(const wasm_trap_t *,
                                        wasmtime_trap_code_t *code);
```

`wasmtime_trap_code` definition in `crates/c-api/src/trap.rs`

```rust
/* vulnerable function in versions 2.0.1 and earlier */
pub extern "C" fn wasmtime_trap_code(raw: &wasm_trap_t, code: &mut i32) ->
     bool { /* function body */ }

/* patched function in versions 2.0.2 and later */
pub extern "C" fn wasmtime_trap_code(raw: &wasm_trap_t, code: &mut u8) ->
    bool { /* function body */ }
```



Out-of-bounds Memory Write Vulnerability in CVE-2022-39394

Calling **`wasmtime_trap_code`** results in a 4-byte write into a 1-byte buffer provided by the caller, leading to three zero bytes being written beyond the 1-byte location.

`get_error_message` calls `wasmtime_trap_code` to provide trap code associated with a given trap.

```
1  static void get_error_message(const char *message,
2                                wasmtime_error_t *error, wasm_trap_t *trap)
3  {
4      fprintf(stderr, "error: %s\n", message);
5      int sensitive_data = 100;
6      wasmtime_trap_code_t code;
7      wasm_byte_vec_t error_message;
8      if (error == NULL)
9      {
10         /* wasmtime_trap_code() will write 4 bytes into the 1 byte
       wasmtime_trap_code_t */
11         wasmtime_trap_code(trap, &code);
12         fprintf(stderr, "trap code: %d\n", code);
13         wasm_trap_message(trap, &error_message);
14         wasm_trap_delete(trap);
15     }
16     else
17     {
18         wasmtime_error_message(error, &error_message);
19         wasmtime_error_delete(error);
20     }
21     fprintf(stderr,"%.*s\n", (int)error_message.size, error_message.data);
22     wasm_byte_vec_delete(&error_message);
23     /* some other code ... */
24 }
```

Out-of-bounds Memory Write Vulnerability Exploitation

- **Control Hijacking:** Attackers may take advantage of this vulnerability to modify program execution flow so that the next instruction will be pointing to a memory location where a malicious code is injected.
- **Denial-of-Service (DoS):** The out-of-bounds write can be exploited to overwrite key data structures, causing the program to malfunction or crash.

Related Work: Vulnerability/CVEs Classification

- **"A fine-grained classification and security analysis of web-based virtual machine vulnerabilities"**, 2021, F. Yilmaz, M. Sridhar, A. Mohanty, V. Tendulkar, K. W. Hamlen, *Computers & Security*
  - Web-based virtual machines are one of the primary targets of attackers due to number of design flaws they contain and the connectivity provided by the Web.
  - Authors propose Inscription that mitigates web attacks targeting unpatched, legacy Flash VMs and their apps.
  - Inscription works by modifying incoming Flash binaries with extra security programming that self-checks against known VM exploits as the modified binary executes.
  - Authors also re-classify ActionScript CVE vulnerabilities labeled as generic *memory corruption* and *unspecified* into one of more fine-grained sub-classes (e.g., use-after-free)

Related Work: WebAssembly Runtimes Analysis

- **“A Comprehensive Study of WebAssembly Runtime Bugs”**, 2023, Y. Wang, Z. Zhou, Z. Ren, D. Liu and H. Jiang, *IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*
  - Analyzed bugs in V8 and SpiderMonkey which are in two mainstream browsers, and two popular standalone runtimes, Wasmer and Wasmtime
  - RQ1: How are the root causes of WebAssembly runtime bugs distributed? Incorrect Algorithm Implementation, Incorrect Memory Handling, Incorrect Exception Handling, etc
  - RQ2: How are the symptoms of WebAssembly runtime bugs distributed? Crash, Incorrect Functionality, Build Error, Bad Performance, Hang
  - RQ3: What is the connection between WebAssembly runtime bug root causes and symptoms? For example, bugs related to Memory Handling mostly cause Crash

Related Work: WebAssembly Runtimes Analysis

- **“A Comprehensive Study of WebAssembly Runtime Bugs”**, 2023, Y. Wang, Z. Zhou, Z. Ren, D. Liu and H. Jiang, *IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*
  - Analyzed bugs in V8 and SpiderMonkey which are in two mainstream browsers, and two popular standalone runtimes, Wasmer and Wasmtime
  - RQ4: How long do WebAssembly runtime bugs take to fix? Bugs in SpiderMonkey are fixed in 21 days on average and in V8 they took 71 days on average
  - RQ5: How many files and lines of code must be changed to fix a WebAssembly runtime bug? At median, 3 files need to be changed for fixing a single bug
  - RQ6: Do the bugs of different WebAssembly runtimes have anything in common? The four WebAssembly runtimes have moderate and strong correlations in root causes of bugs and strong correlations in symptoms of bugs.

Preliminary Study

- We have collected a set of 113 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]

Our Initial Plan

- Analyzing collected vulnerabilities (CVEs)
  - Reproducing attacks using vulnerable products and exploits
  - Analyze root causes of each CVE
  - Categorize CVEs based on different features such as Type of weakness, Vulnerable functions/APIs, Product, Severity, etc.

- Gaining Insights into automating vulnerability detection
  - Find possible patterns of vulnerabilities in WebAssembly runtimes/VMs

Thank You!