Optimism - Fix Review: MIPS EventFd and Syscall Error Handling

Security Review

Solo review by:
Zigtur, Lead Security Researcher

June 21, 2025

Contents

1 Introduction **2**
1.1 About Cantina 2
1.2 Disclaimer 2
1.3 Risk assessment 2
1.3.1 Severity Classification 2

2 Security Review Summary **3**

3 Missing `EventFd2` Support **4**

4 Incorrect Syscall Error Handling **4**

5 Closing Remarks **5**

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

A security review is a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While the review endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that a security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Optimism is a fast, stable, and scalable L2 blockchain built by Ethereum developers, for Ethereum developers. Built as a minimal extension to existing Ethereum software, Optimism's EVM-equivalent architecture scales your Ethereum apps without surprises. If it works on Ethereum, it works on Optimism at a fraction of the cost.

From Jun 13th the security researchers conducted a review of optimism on commit hash 7f7b9abb. The particular scope of the review is the following set of pull requests aiming to fix issues found internally by OP Labs:

- PR 16341.
- PR 16346.
- PR 16384.

The researchers conclude that the fixes implemented for the security issues described in the aforementioned scope are accurate. The MIPS64 virtual machine now correctly handles syscall errors and implements event file descriptor capabilities to support Go 1.23 runtime.

3 Missing `EventFd2` Support

Context: (*No context files provided by the reviewer*)

Description: The Go 1.23 runtime uses the `eventfd2` syscall in the `netpollinit` function. As a no-op operation in the Optimism MIPS64 virtual machine, this syscall was returning `0` as the file descriptor `fd`. The `fd = 0` file descriptor maps to the standard input `stdin`.

The Go runtime would then attempt to write to this `0` event file descriptor in the `netpollBreak` function. This specific `write` syscall resulted in an `EBADF` error because writing to `fd = 0` is not supported in the VM.

```
func HandleSysWrite(...) (...) {
    // ...
    default:
        v0 = ^Word(0)
        v1 = MipsEBADF
```

Zigtur: This issue was fixed by implementing support for the `eventfd2` syscall and returning a specific event file descriptor `FdEventFd = 100`. Writing to this event file descriptor is now supported and will return the non-blocking error `EAGAIN` to avoid panics.

```
func HandleSysWrite(...) (...) {
    // ...
    case FdEventFd:
        // Always report that the write could not be completed
        // This acts as if the counter has already reached the maximum value
        v0 = MipsEAGAIN
        v1 = SysErrorSignal
```

4 Incorrect Syscall Error Handling

Context: (*No context files provided by the reviewer*)

Description: In the syscall handling logic, the `v0` value was set to the error signal and `v1` to the error code. These values were then respectively set in the `V0` and `A3` registers (through `HandleSyscallUpdates`).

```
func HandleSysWrite(...) (...) {
    // ...
    default:
        v0 = ^Word(0)
        v1 = MipsEBADF

func HandleSyscallUpdates(cpu *mipsevm.CpuScalars, registers *[32]Word, v0, v1 Word) {
    registers[register.RegSyscallRet1] = v0
    registers[register.RegSyscallErrno] = v1

    cpu.PC = cpu.NextPC
    cpu.NextPC = cpu.NextPC + 4
}
```

The error signal in the `v0` variable would be set in the `V0` register and the error code in `v1` would be set in the `A3` register. However, syscalls for the MIPS architecture expect the error signal to be set in the `A3` register and the error code to be set in the `V0` register. This incorrect handling causes syscall errors to be misinterpreted, as the calling code expects to find the error flag in `A3` and the actual error value in `V0`, but instead finds them in the opposite registers.

Zigtur: This issue was fixed by adding a `SysErrorSignal` constant that is set in `v1` instead of `v0`. The error code is now set in `v0`:

```
const (
    SysErrorSignal = ^Word(0)
)

func HandleSysWrite(...) (...) {
    // ...
    default:
        v0 = MipsEBADF      // Error code in v0 (goes to V0 register)
        v1 = SysErrorSignal // Error signal in v1 (goes to A3 register)
```

This fix ensures that:

- The error code is properly placed in the `V0` register .
- The error signal flag is correctly set in the `A3` register.
- The syscall error handling complies with the `syscall` specifications for MIPS architecture.

With this correction, the MIPS VM can now properly detect and handle syscall errors according to the expected calling convention.

In addition, the support for the `eventfd2` syscall used by the Go 1.23 runtime enforces the `EFD_NONBLOCK` flag on the file descriptor. If the flag is not set, the MIPS VM will return an `EINVAL` error.

5 Closing Remarks

Both fixes are accurate and ensure that the issue will not be triggerable again.

It is worth mentioning that the `Fcntl` syscall does not support setting flags on the event file descriptor. This should be possible according to the `eventfd2` specifications:

> EFD_NONBLOCK (since Linux 2.6.27)
> Set the O_NONBLOCK file status flag on the open file description (see open(2)) referred to by the new file descriptor. **Using this flag saves extra calls to fcntl(2) to achieve the same result**.

This behavior was acknowledged by Optimism on Slack:

> That's intentional. Go programs shouldn't be setting flags in an eventfd descriptor.