

# Follow-on for Wasmtime's Feb 24, 2026 Security Advisory

Pat Hickey

F5

# The Advisory

<https://github.com/bytecodealliance/wasmtime/security/advisories/GHSA-852m-cvvp-9p4w>

The screenshot shows a GitHub repository page for 'bytecodealliance / wasmtime'. The 'Security' tab is selected, showing a single advisory. The advisory title is 'Guest-controlled resource exhaustion in WASI implementations'. It was published by alexcrichton 3 hours ago and has 17 comments. The advisory details section includes a table for affected and patched versions, a severity rating of Moderate (6.9 / 10), and detailed CVSS v4 base metrics. The 'Impact' section describes how Wasmtime's implementation of WASI host interfaces can lead to guest-controlled resource exhaustion, causing denial-of-service. A list of specific impact points is provided, such as allocating large amounts of host memory or causing host crashes. The advisory also notes that Wasmtime's security bug policy considers these behaviors as vulnerabilities.

Package: [wasmtime \(Rust\)](#)

Affected versions	Patched versions
< 24.0.6, >= 25.0.0, < 36.0.6, >= 37.0.0, < 40.0.4, >= 41.0.0, < 41.0.0, < 41.0.4	24.0.6, 36.0.6, 40.0.4, 41.0.4, 42.0.0

Severity: Moderate (6.9 / 10)

CVSS v4 base metrics:

Exploitability Metrics	Network
Attack Vector	Network
Attack Complexity	Low
Attack Requirements	Present
Privileges Required	Low
User interaction	Passive

Vulnerable System Impact Metrics:

Confidentiality	None
Integrity	None
Availability	High

Subsequent System Impact Metrics:

Confidentiality	None
Integrity	None
Availability	High

Learn more about base metrics

CVSS:4.0/AV:N/AC:L/AT:P/PR:L/UI:P/VC:N/VI:N/VA:H/SC:N/SI:N/SA:H

alexchrichton opened last week · edited by pchickey · ...

Description

Impact

Wasmtime's implementation of WASI host interfaces are susceptible to guest-controlled resource exhaustion on the host. Wasmtime did not appropriately place limits on resource allocations requested by the guests. This serves as a denial-of-service vector where a guest can induce a range of crashing behaviors on the host such as:

- Allocating arbitrarily large amounts of host memory.
- Causing an allocation failure on the host, which in Rust defaults to aborting the process.
- Causing a panic on the host due to over-large allocations being performed.
- Cause degradation in performance of the host by holding excessive host memory alive.

Wasmtime's [security bug policy](#) considers all of these behaviors a security vulnerability. Wasmtime's implementation of WASI has a number of different ways that resource exhaustion could happen, and fixing any one of them is insufficient from solving this vulnerability. A number of individual issues are grouped within this advisory and as a whole represent the known ways that guests can exhaust resources on the host.

# Credit

- New contributor @mbund reported memory leak in wasmtime-wasi's p1 impl. Thank you!!!
- Alex investigated and realized it was a much bigger issue
- Alex did most of the remediation, I helped

bytecodealliance / wasmtime

Code Issues Pull requests 91 Agents Discussions Actions Security 27 Insights Settings

⚠ mbund requested your review on this pull request.

## Fix wasip1 memory leak #12599

[Open](#) mbund wants to merge 2 commits into `bytecodealliance:main` from `mbund:wasip1-resource-table-memory-leak-fix`

Conversation 0 Commits 2 Checks 45 Files changed 1

mbund commented last week

I use wasmtime to host a long lived wasip1 program, and I discovered a memory leak.

The issue is in async wasip1 contexts. The minimal code to reproduce (is taken from the existing `wasip1-async` example):

# Guest input can consume Host memory

- By creating resources – *new limit on how many, traps*
- By calling **any** import function accepting lists or strings with large lists and strings – *new limit on total size, traps*
- By calling **particular** import functions with arguments that cause allocations, some of which are long-lived
  - Stream or file read takes a size argument: allocates a BytesMut. Now caps at maximum read size, which is allowed by spec
  - Get random bytes takes a size argument: allocates a Vec<u8>. *New limit, traps*
  - HTTP Fields gives arbitrary Vec<u8> to a resource to own forever. *New limit, traps*
    - Bonus: when this exceeded 32k it would panic the host in http::HeaderMap. Fixed!

# Security mitigations, not long-term solutions

- We broke userland
  - There may be valid programs which now trap
- We did so unilaterally
  - Because it was a security issue, we couldn't seek community input
- If the host changes a limit, the guest can't discover it
  - Writing a guest which doesn't trap today requires reading the wasmtime release notes.
    - wasi-libc and wstd could help, but do not (yet)
  - Writing a guest which will always respect host limits is impossible

If you maintain a host which isn't just  
wasmtime-{wasi + wasi-http}, you  
*probably have special cases you  
need to discover and fix*

And you might have to break userland too

# This needs to flow back up to the standards

- Amorphous and large design space. First, socialize some actual plans, then figure out the standards
- Does the CM spec take on resource and lifting fuel implementation limits?
- wasip1 and wasip2 spec – need to document these new limits, which requires on settling on them
- wasip3 spec: can we design our way out?
  - It's the 11<sup>th</sup> hour before release, but I don't think it should go out without thinking more about these issues
- Alex filed <https://github.com/WebAssembly/WASI/issues/888>, 889, 890