patriksimek / **vm2** Public

Code Issues 10 Pull requests 1 Actions Projects Wiki Secur

Commit `2b5f3e3`

patriksimek committed last week

```
test(GHSA-grj5-jjm8-h35p): add descriptor-chain history coverage (PoCs #1-5)

The advisory thread documents seven escape attempts iterating through
descriptor-based bypasses of the `__lookupGetter__` + `Buffer.apply`
constructor leak. PoC #6 (the Symbol.species self-return chain) is the
headline case covered in repro.js. PoCs #1-5 are closed by intermediate
fixes already on public main; this file ensures those defenses do not
regress. Each test exercises the verbatim PoC with host-side hostMark
for ground truth.
```

main · v3.11.2 ••• v3.11.0

1 parent 7395c3a commit 2b5f3e3

1 file changed +137

Top

Filter files...

test/ghsa/GHSA-grj5-jjm8-h35p

descriptor-chain-history.js

Search within code

...-grj5-jjm8-h35p/descriptor-chain-history.js

```
•••   @@ -0,0 +1,137 @@
1   + 'use strict';
2   +
3   + /**
4   +  * GHSA-grj5-jjm8-h35p — descriptor-chain history coverage (PoCs #1–5)
5   +  *
6   +  * The advisory thread documents seven escape attempts iterating through
```

```
+ * bypasses. PoC #6 (the `cwu` / Symbol.species
+ * self-return chain) is the headline case, closed by the fix on this branch
+ * and covered in `repro.js`. PoCs #1-5 are an earlier descriptor-based
+ * family that progressively bypassed each tightening of the
+ * `__lookupGetter__` + `Buffer.apply` constructor leak.
+ *
+ * They are CLOSED by intermediate fixes that have already landed on public
+ * `main` (e.g. `9084cd6`, `c17c27e`). This file ensures those defenses do
+ * not regress: each test exercises the exact PoC verbatim from the report
+ * with a host-side `hostMark` for ground truth.
+ *
+ * Coverage matrix (verified on Node 25.9.0):
+ *
+ *   PoC #1 (Object.getOwnPropertyDescriptor → .value)            blocked
+ *   PoC #2 (ho.getOwnPropertyDescriptor → .value)                blocked
+ *   PoC #3 (ho.entries(cd).find('value') chain)                  blocked
+ *   PoC #4 (getOwnPropertyDescriptors double indirection)        blocked
+ *   PoC #5 (find('constructor') variant)                         blocked
+ *
+ * RCE payload is replaced with `process.pid` extraction so the test is
+ * observation-only.
+ */
+
+ const assert = require('assert');
+ const { VM } = require('../../../lib/main.js');
+
+ describe('GHSA-grj5-jjm8-h35p — descriptor chain history (PoCs #1–5)', function
  () {
+
+     it('PoC #1 (Object.getOwnPropertyDescriptor): host process.pid not
  extractable', function () {
+         const hostMark = {pid: null, err: null};
+         const vm = new VM({sandbox: {hostMark}});
+         vm.run(`
+             try {
+                 const g = ({}).__lookupGetter__;
+                 const a = Buffer.apply;
+                 const p = a.apply(g, [Buffer, ['__proto__']]);
+                 hostMark.pid =
  Object.getOwnPropertyDescriptor(p.call(a),'constructor').value('return
```

```
process.pid')();
+            } catch (e) { hostMark.err = e.message; }
+        `);
+        assert.strictEqual(hostMark.pid, null,
+            'host process.pid was extracted: hostMark.pid=' + hostMark.pid + ',
host pid=' + process.pid);
+    });
+
+    it('PoC #2 (ho.getOwnPropertyDescriptor): host process.pid not
extractable', function () {
+        const hostMark = {pid: null, err: null};
+        const vm = new VM({sandbox: {hostMark}});
+        vm.run(`
+            try {
+                const g = ({}).__lookupGetter__;
+                const a = Buffer.apply;
+                const p = a.apply(g, [Buffer, ['__proto__']]);
+                const fp = p.call(a);
+                const op = p.call(fp);
+                const ho = op.constructor;
+                hostMark.pid =
ho.getOwnPropertyDescriptor(fp,'constructor').value('return process.pid')();
+            } catch (e) { hostMark.err = e.message; }
+        `);
+        assert.strictEqual(hostMark.pid, null,
+            'host process.pid was extracted: hostMark.pid=' + hostMark.pid + ',
host pid=' + process.pid);
+    });
+
+    it('PoC #3 (ho.entries+a.apply chain): host process.pid not extractable',
function () {
+        const hostMark = {pid: null, err: null};
+        const vm = new VM({sandbox: {hostMark}});
+        vm.run(`
+            try {
+                const g = ({}).__lookupGetter__;
+                const a = Buffer.apply;
+                const p = a.apply(g, [Buffer, ['__proto__']]);
+                const fp = p.call(a);
+                const op = p.call(fp);
```

```
78  +                const ho = op.constructor;
79  +                const cd = ho.getOwnPropertyDescriptor(fp,'constructor');
80  +                const e = ho.entries(cd).find(v => v[0] === 'value');
81  +                e.shift();
82  +                e.push([undefined, ['return process.pid']]);
83  +                hostMark.pid = a.apply(a, e)();
84  +            } catch (e) { hostMark.err = e.message; }
85  +        `);
86  +        assert.strictEqual(hostMark.pid, null,
87  +            'host process.pid was extracted: hostMark.pid=' + hostMark.pid + ',
      host pid=' + process.pid);
88  +    });
89  +
90  +    it('PoC #4 (getOwnPropertyDescriptors double indirection): host process.pid
      not extractable', function () {
91  +        const hostMark = {pid: null, err: null};
92  +        const vm = new VM({sandbox: {hostMark}});
93  +        vm.run(`
94  +            try {
95  +                const g = ({}).__lookupGetter__;
96  +                const a = Buffer.apply;
97  +                const p = a.apply(g, [Buffer, ['__proto__']]);
98  +                const fp = p.call(a);
99  +                const op = p.call(fp);
100 +                const ho = op.constructor;
101 +                const cd =
      ho.getOwnPropertyDescriptor(ho.getOwnPropertyDescriptors(fp,'constructor'),'con
      structor');
102 +                const ee = ho.entries(cd).find(v => v[0] === 'value');
103 +                ee.shift();
104 +                const e = ho.entries.apply(null, ee).find(v => v[0] ===
      'value');
105 +                e.shift();
106 +                e.push([undefined, ['return process.pid']]);
107 +                hostMark.pid = a.apply(a, e)();
108 +            } catch (e) { hostMark.err = e.message; }
109 +        `);
110 +        assert.strictEqual(hostMark.pid, null,
111 +            'host process.pid was extracted: hostMark.pid=' + hostMark.pid + ',
      host pid=' + process.pid);
```

```
112 +     });
113 +
114 +     it('PoC #5 (find(\'constructor\') variant): host process.pid not extractable', function () {
115 +         const hostMark = {pid: null, err: null};
116 +         const vm = new VM({sandbox: {hostMark}});
117 +         vm.run(`
118 +             try {
119 +                 const g = ({}).__lookupGetter__;
120 +                 const a = Buffer.apply;
121 +                 const p = a.apply(g, [Buffer, ['__proto__']]);
122 +                 const fp = p.call(a);
123 +                 const op = p.call(fp);
124 +                 const ho = op.constructor;
125 +                 const cd = ho.getOwnPropertyDescriptors(fp,'constructor');
126 +                 const ee = ho.entries(cd).find(v => v[0] === 'constructor');
127 +                 ee.shift();
128 +                 const e = ho.entries.apply(null, ee).find(v => v[0] === 'value');
129 +                 e.shift();
130 +                 e.push([undefined, ['return process.pid']]);
131 +                 hostMark.pid = a.apply(a, e)();
132 +             } catch (e) { hostMark.err = e.message; }
133 +         `);
134 +         assert.strictEqual(hostMark.pid, null,
135 +             'host process.pid was extracted: hostMark.pid=' + hostMark.pid + ', host pid=' + process.pid);
136 +     });
137 + });
```

Comments 0

Please sign in to comment.