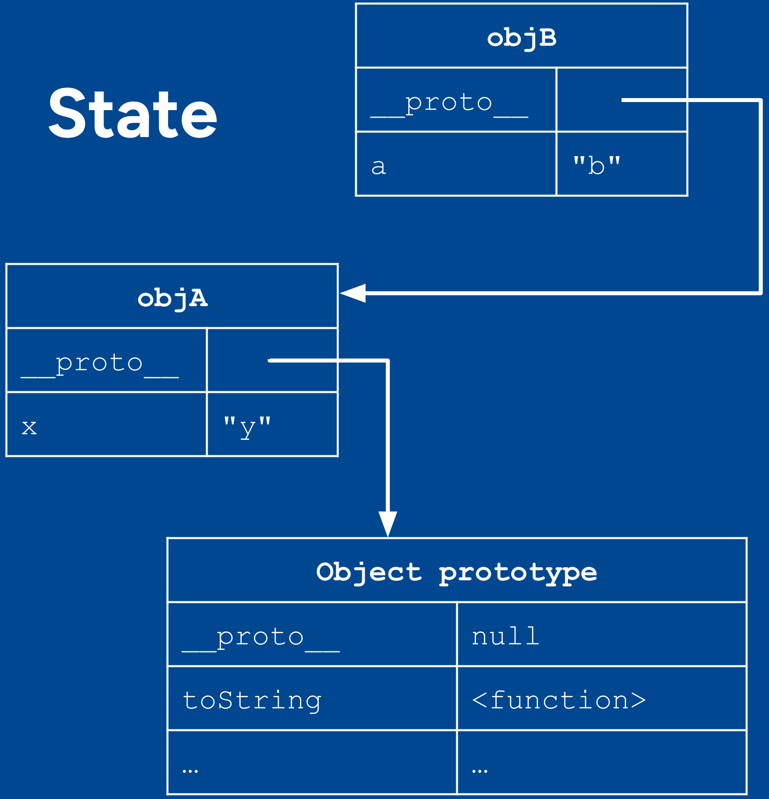
GHunter: Universal Prototype Pollution Gadgets in JavaScript Runtimes

Eric Cornelissen (KTH), Mikhail Shcherbakov (KTH), Musard Balliu (KTH)

Inheritance in JavaScript

- Prototype based: reuse existing objects for inheritance

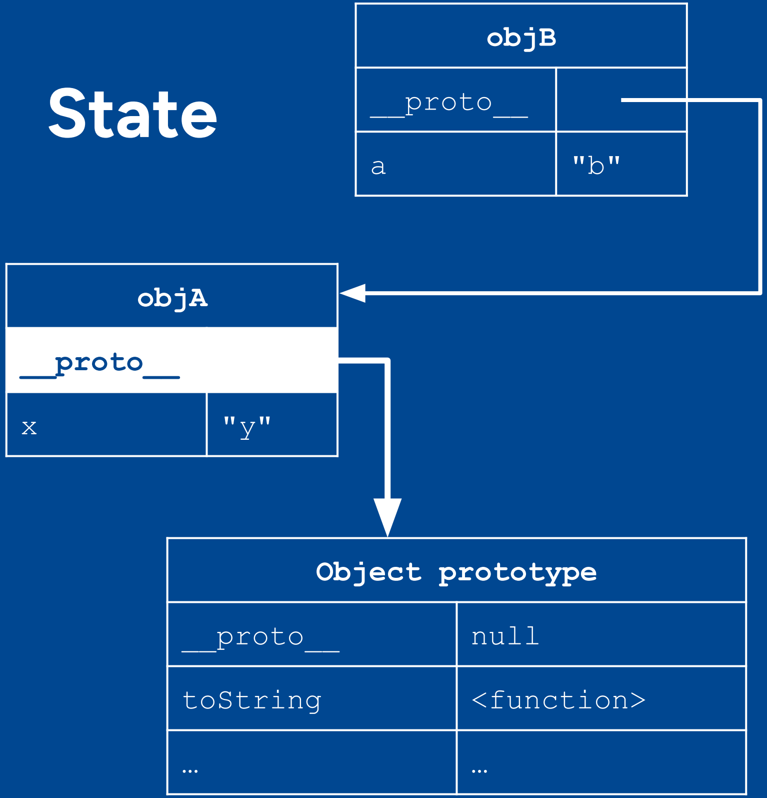
State



Inheritance in JavaScript

- Prototype based: reuse existing objects for inheritance

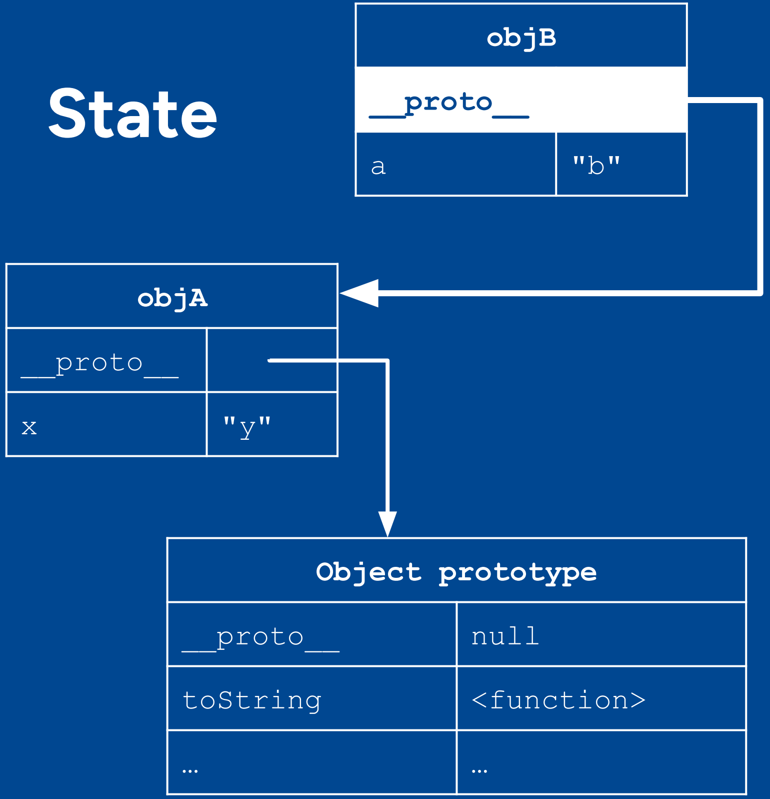
State



Inheritance in JavaScript

- Prototype based: reuse existing objects for inheritance

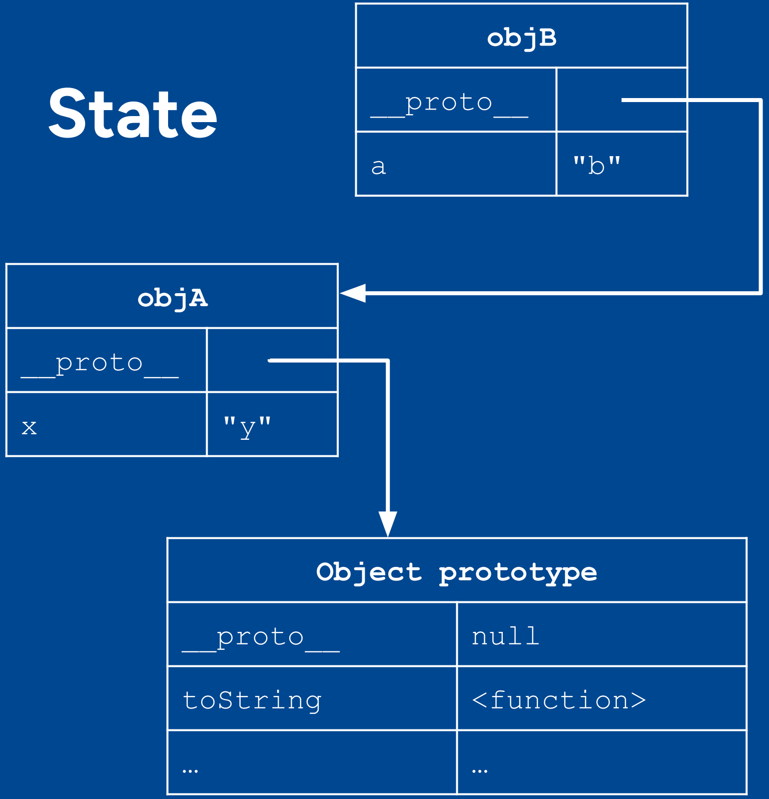
State



Inheritance in JavaScript

- Prototype based: reuse existing objects for inheritance
- No stratification: exposed as regular programming construct

State

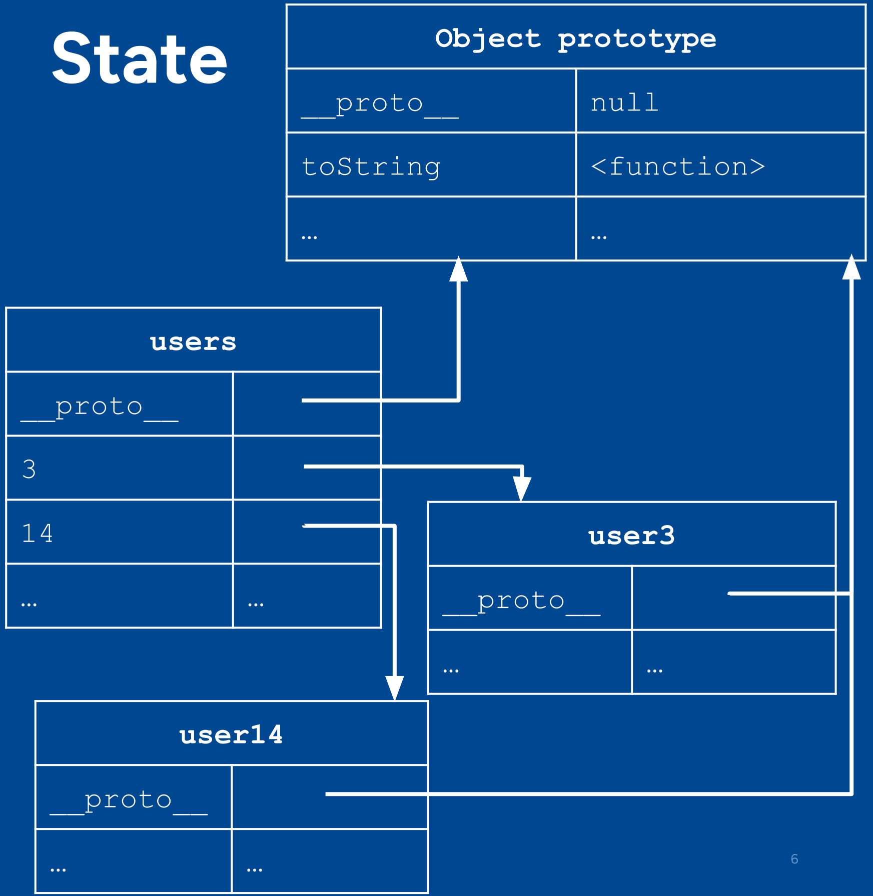


example.com/3
```
{
  "key": "foo",
  "value": "bar"
}
```

Program

```
router.post("/:uid", (req, res) => {
  users[req.uid][req.key]=req.value;
  exec("echo 'A value was stored'");
  res.status(200).send();
});
```

State

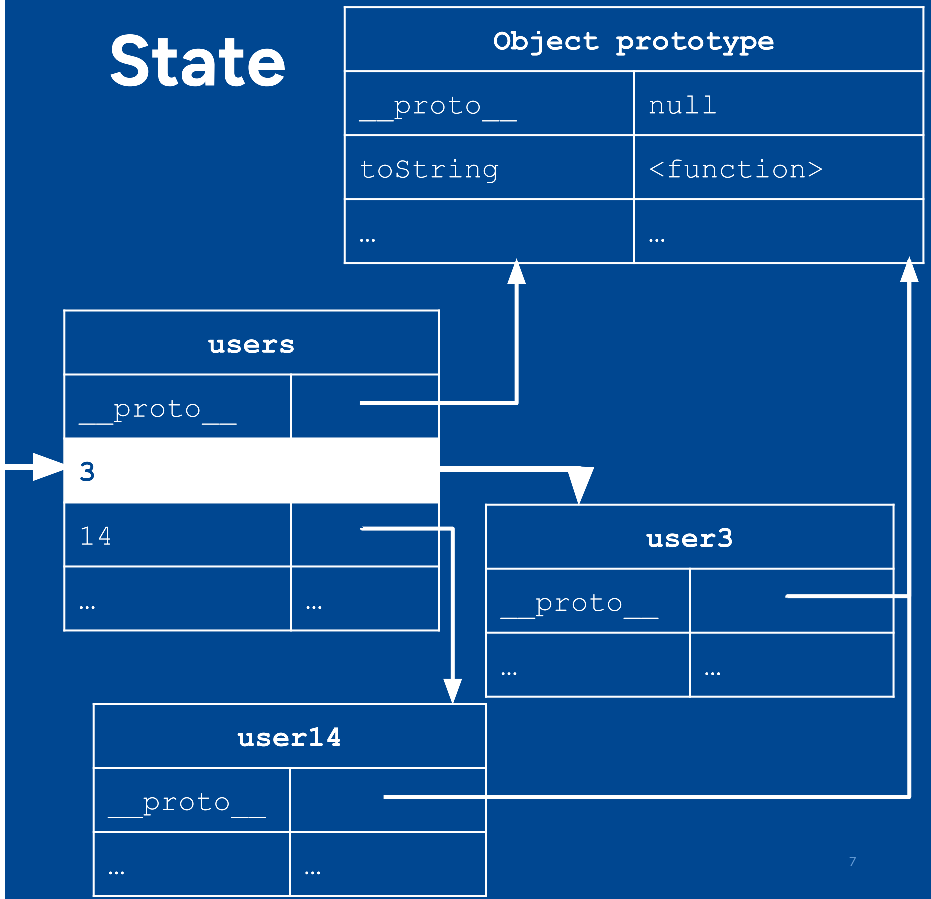


Program

```
example.com/3
{
  "key": "foo",
  "value": "bar"
}
```

```
router.post("/:uid", (req, res) => {
  users[ 3 ][req.key]=req.value;
  exec("echo 'A value was stored'");
  res.status(200).send();
});
```

State

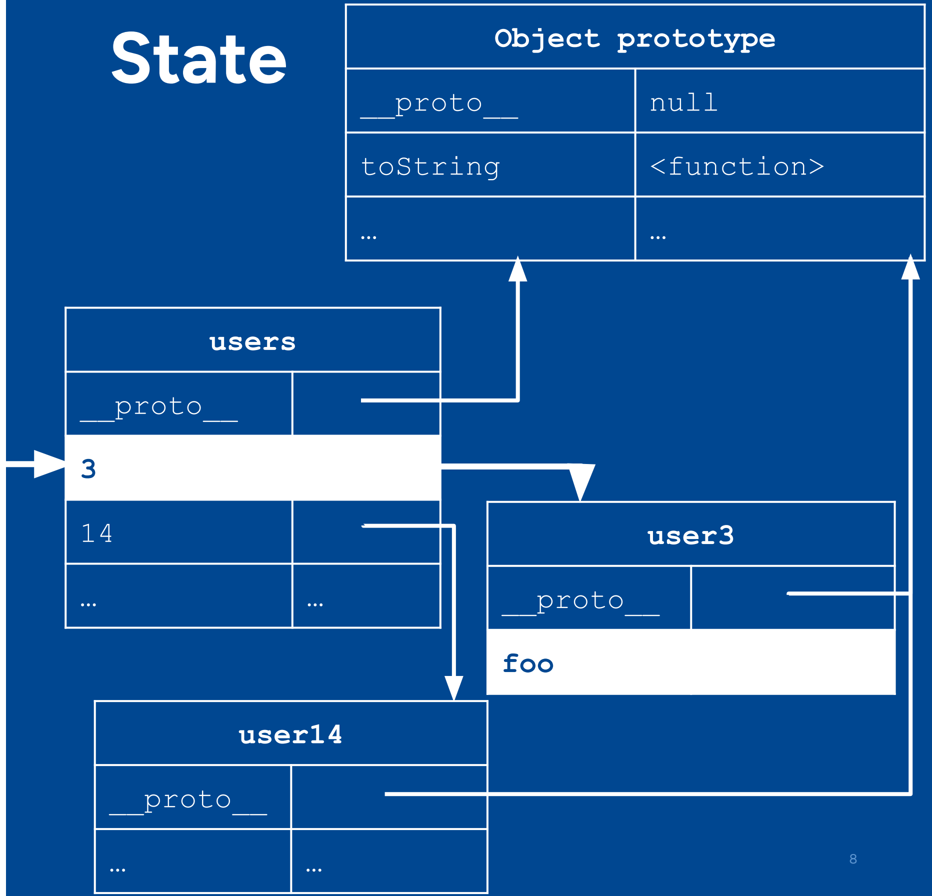


```
example.com/3
{
  "key": "foo",
  "value": "bar"
}
```

Program

```
router.post("/:uid", (req, res) => {
  users[ 3 ][ "foo" ]=req.value;
  exec("echo 'A value was stored'");
  res.status(200).send();
});
```

State

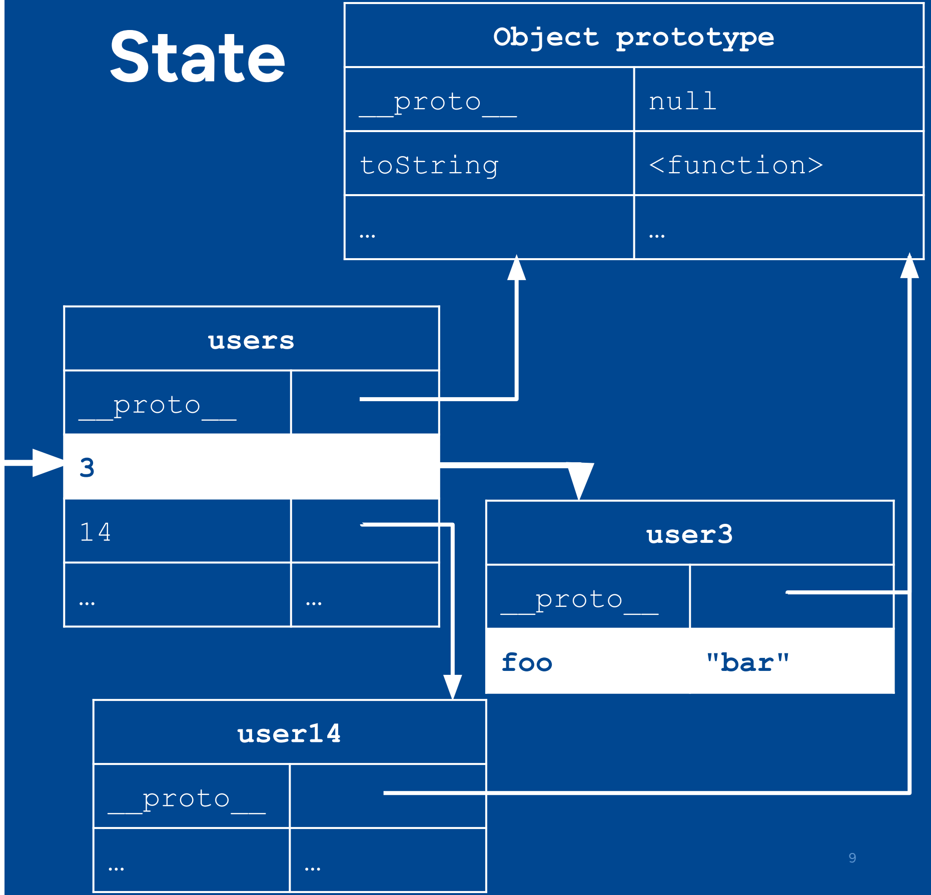


Program

```
example.com/3
{
  "key": "foo",
  "value": "bar"
}
```

```
router.post("/:uid", (req, res) => {
  users[ 3 ][ "foo" ] = "bar" ;
  exec("echo 'A value was stored'");
  res.status(200).send();
});
```

State

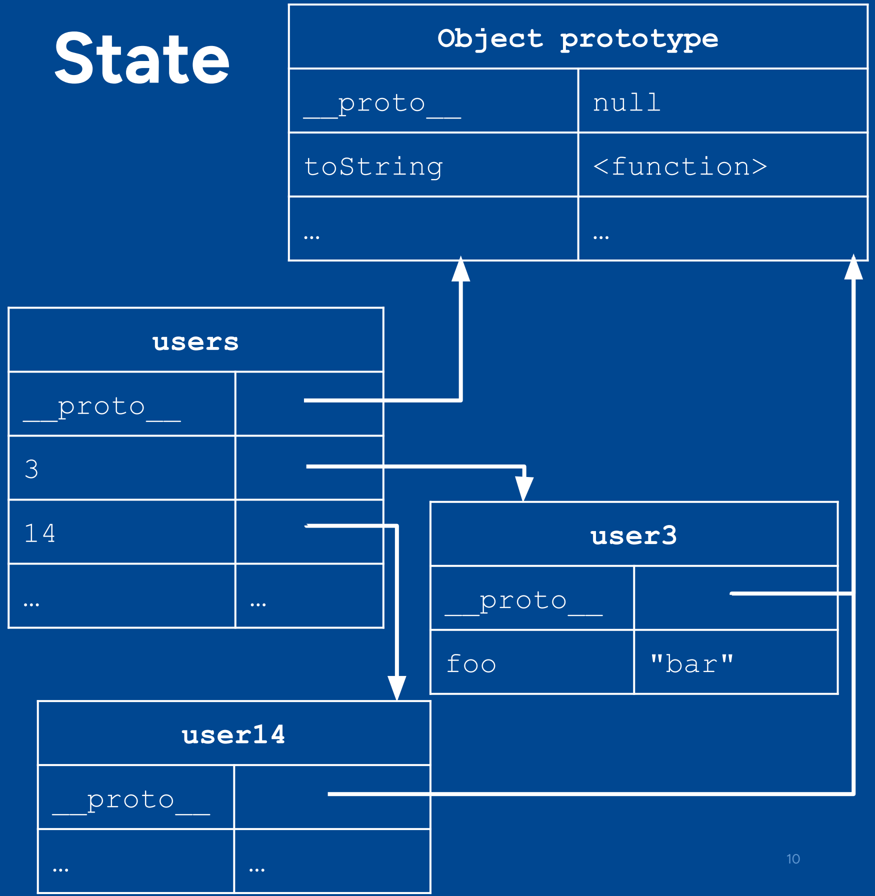


Program

```
example.com/__proto__
{
  "key": "shell",
  "value": "calc"
}
```

```
router.post("/:uid", (req, res) => {
  users[req.uid][req.key]=req.value;
  exec("echo 'A value was stored'");
  res.status(200).send();
});
```

State

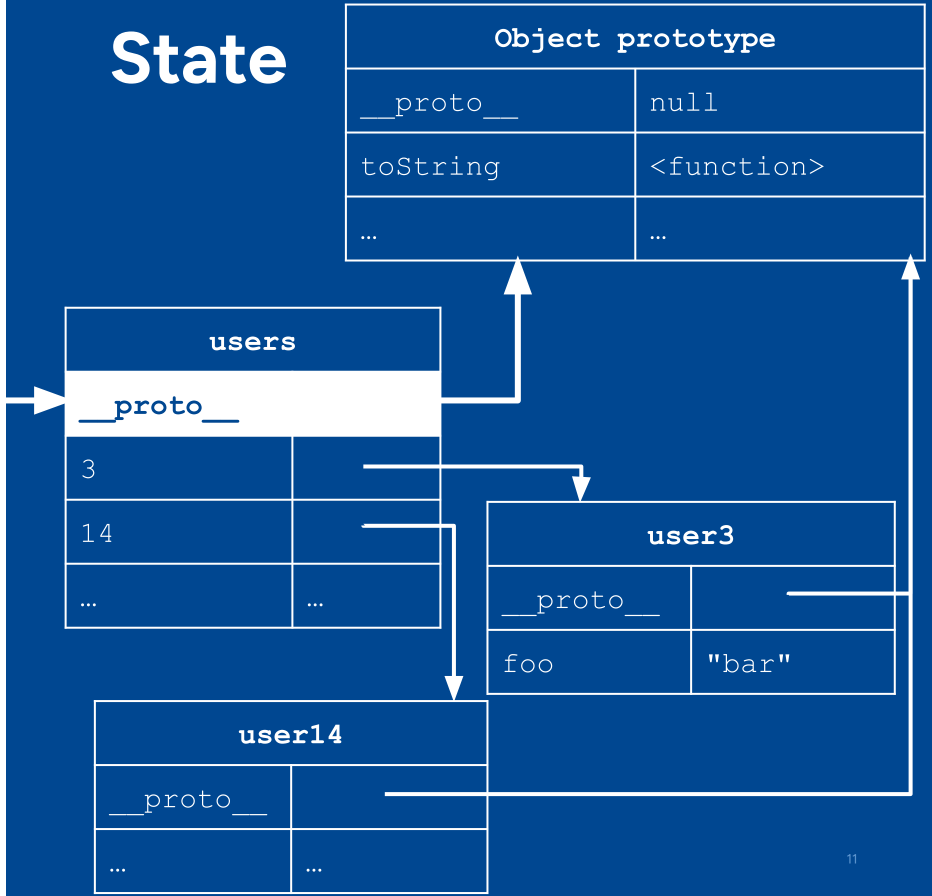


```
example.com/__proto__
{
  "key": "shell",
  "value": "calc"
}
```

Program

```
router.post("/:uid", (req, res) => {
  users[__proto__][req.key]=req.value;
  exec("echo 'A value was stored'");
  res.status(200).send();
});
```

State

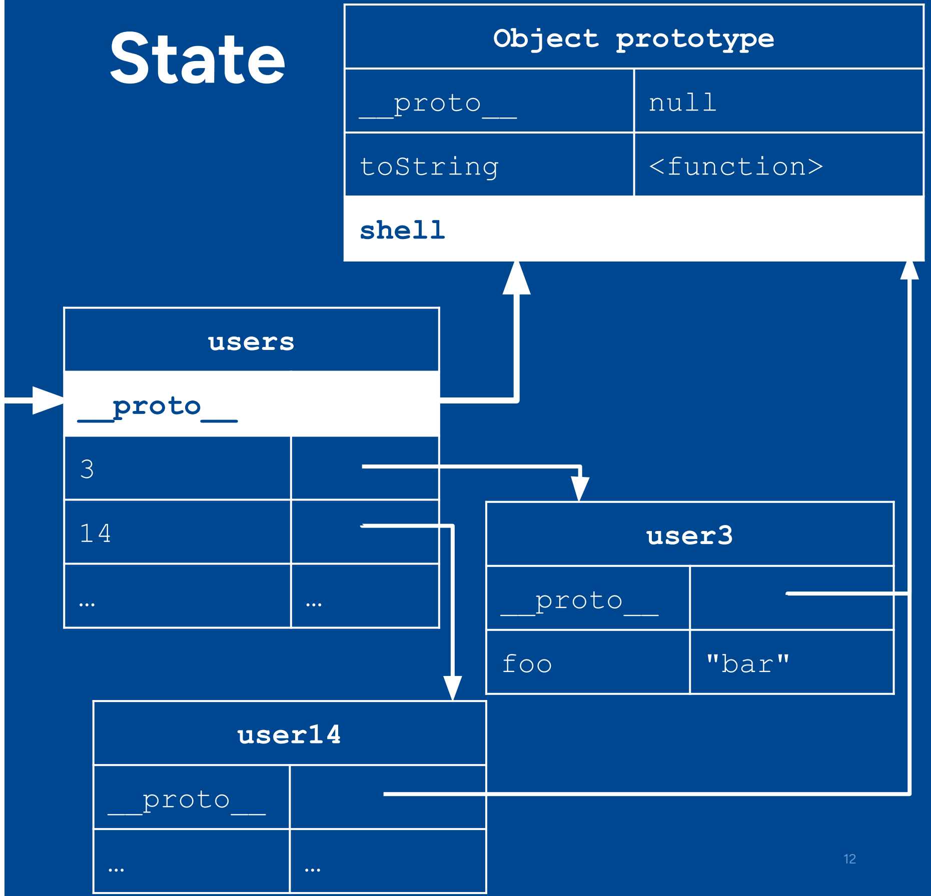


Program

```
example.com/__proto__
{
  "key": "shell",
  "value": "calc"
}
```

```
router.post("/:uid", (req, res) => {
  users[__proto__]["shell"]=req.value;
  exec("echo 'A value was stored'");
  res.status(200).send();
});
```

State

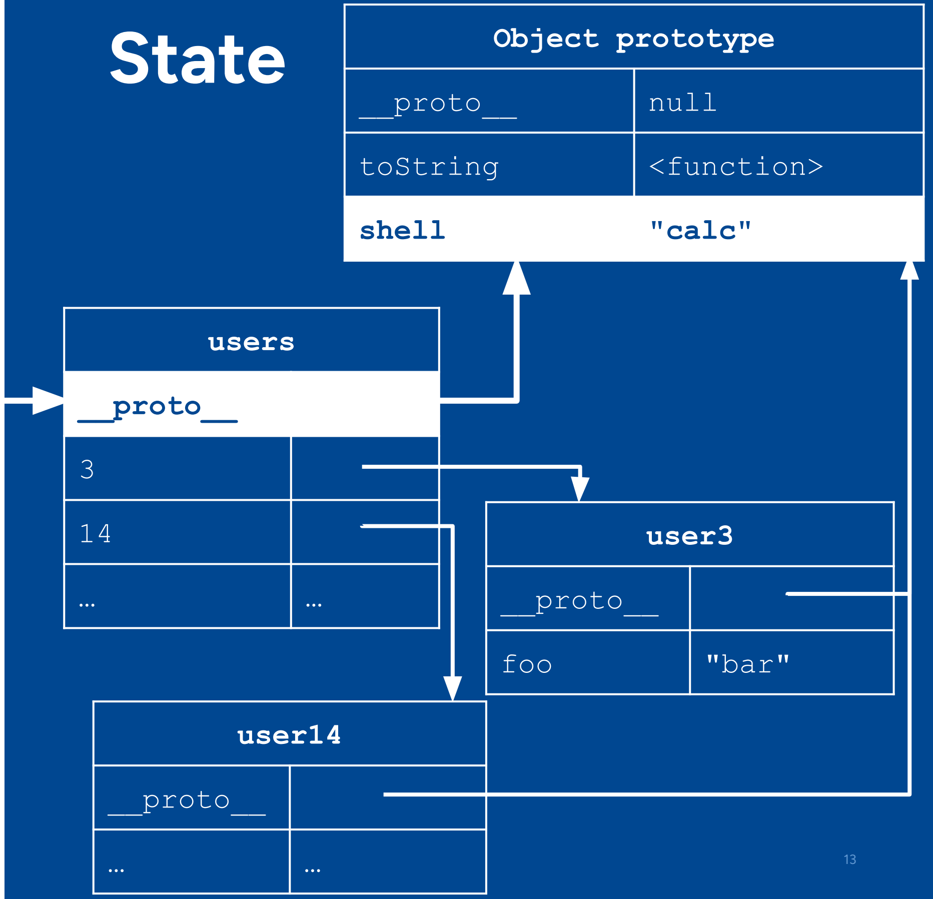


```
example.com/__proto__
{
  "key": "shell",
  "value": "calc"
}
```

Program

```
router.post("/:uid", (req, res) => {
  users[ __proto__ ] [ "shell" ] = "calc" ;
  exec("echo 'A value was stored'");
  res.status(200).send();
});
```

State



Inheritance in JavaScript

- Prototype based: reuse existing objects for inheritance
- No stratification: exposed as regular programming construct
- Leads to pollution

Definition: *Gadget*

An otherwise benign piece of code which inadvertently read from polluted properties to execute security-sensitive operations.

Program

```
router.post("/:uid", (req, res) => {
  users[req.uid][req.key]=req.value;
  exec("echo 'A value was stored'");
  res.status(200).send();
});
```

State

Object prototype	
__proto__	null
toString	<function>
shell	"calc"

Program

```
router.post("/:uid", (req, res) => {
  users[req.uid][req.key]=req.value;
  exec("echo 'A value was stored'");
  res.status(200).send();
});
```

State

Object prototype	
__proto__	null
toString	<function>
shell	"calc"

Program

```
// exec("echo 'A value was stored'");
function exec(cmd, opts) {

  opts = opts || {};

  const sh = opts.shell || "bash";

  op_spawn(`${sh} -c '${clean(cmd)}'`);

}
```

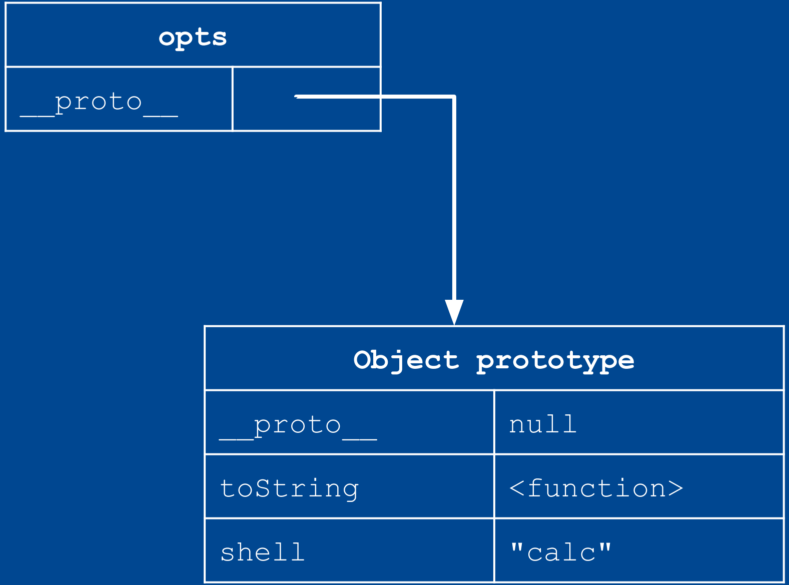
State

Object prototype	
__proto__	null
toString	<function>
shell	"calc"

Program

```
// exec("echo 'A value was stored'");
function exec(cmd, opts) {

  opts = opts || {};

  const sh = opts.shell || "bash";

  op_spawn(`${sh} -c '${clean(cmd)}'`);

}
```

State



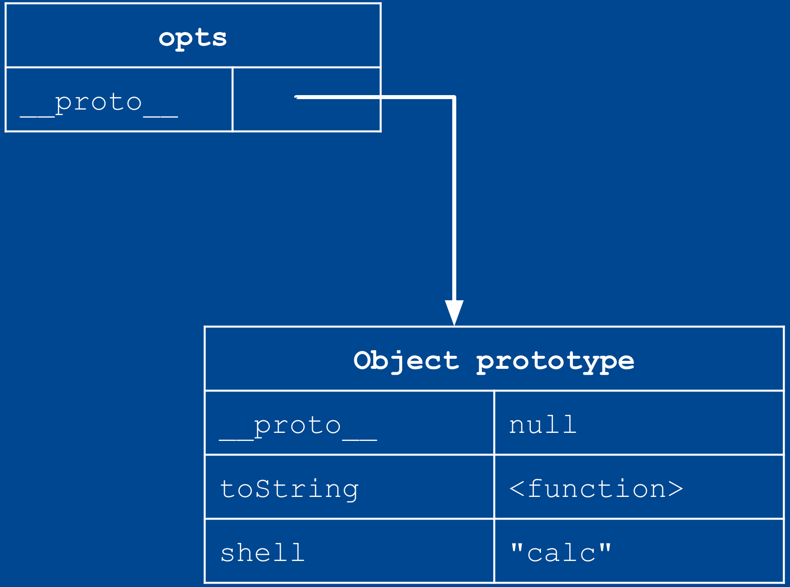
opts	
__proto__	

Object prototype	
__proto__	null
toString	<function>
shell	"calc"

Program

```
// exec("echo 'A value was stored'");
function exec(cmd, opts) {

  opts = opts || {};

  const sh = opts.shell || "bash";

  op_spawn(`${sh} -c '${clean(cmd)}'`);

}
```

State



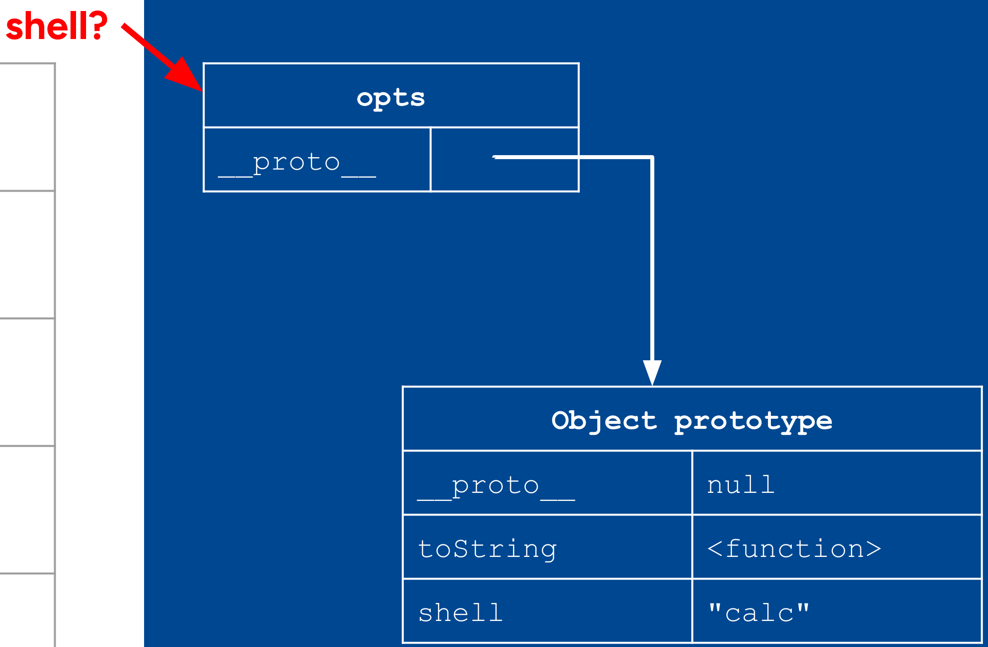
opts	
__proto__	

Object prototype	
__proto__	null
toString	<function>
shell	"calc"

Program

```
// exec("echo 'A value was stored'");
function exec(cmd, opts) {

  opts = opts || {};

  const sh = opts.shell || "bash";

  op_spawn(`${sh} -c '${clean(cmd)}'`);

}
```

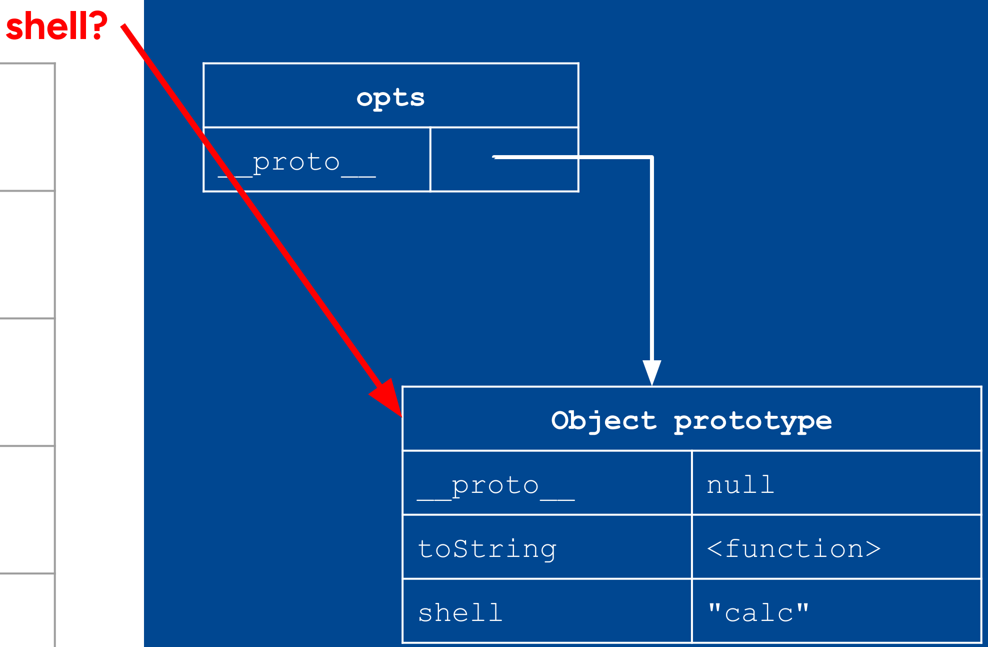
State



Program

```
// exec("echo 'A value was stored'");
function exec(cmd, opts) {

  opts = opts || {};

  const sh = opts.shell || "bash";

  op_spawn(`${sh} -c '${clean(cmd)}'`);

}
```

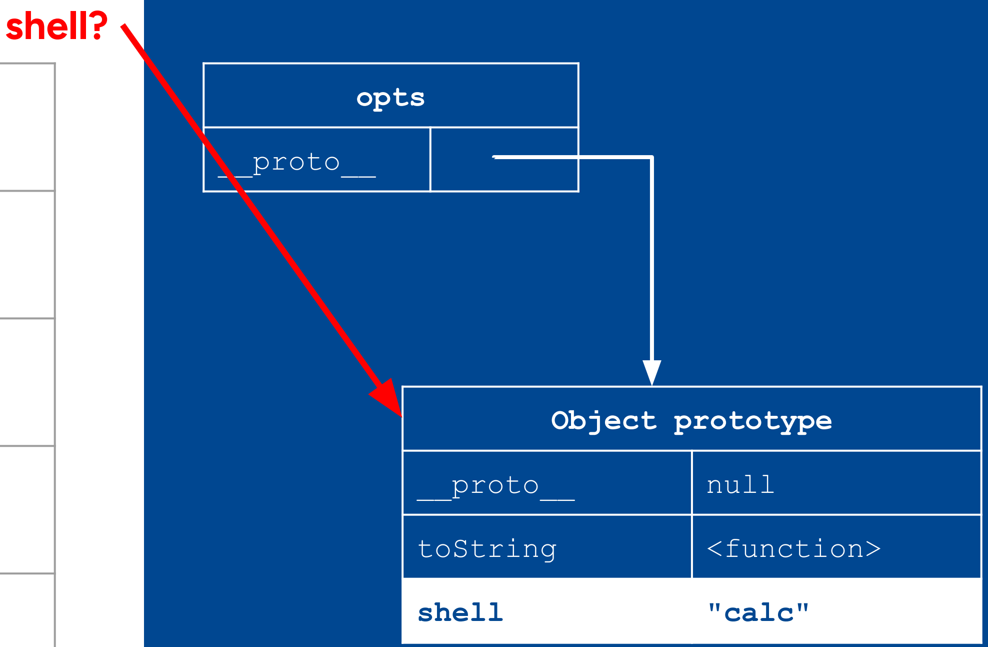
State



Program

```
// exec("echo 'A value was stored'");
function exec(cmd, opts) {

  opts = opts || {};

  const sh = opts.shell || "bash";

  op_spawn(`${sh} -c '${clean(cmd)}'`);

}
```

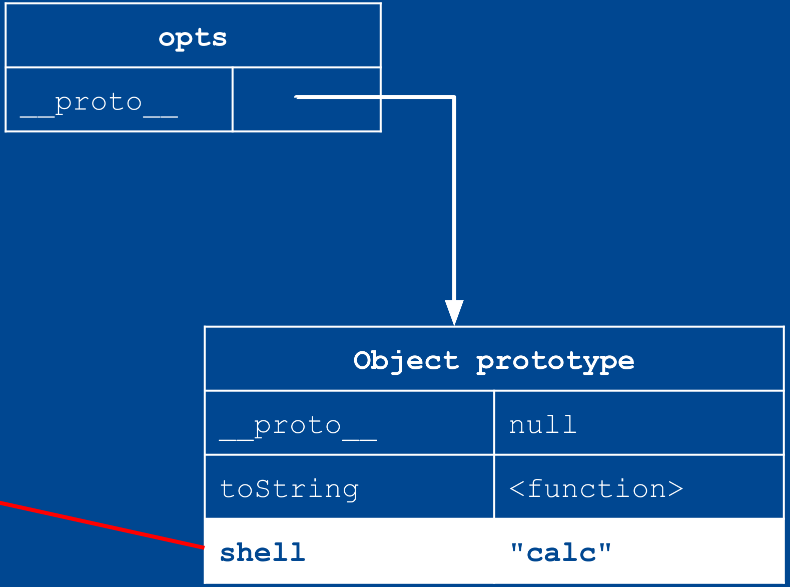
State



Program

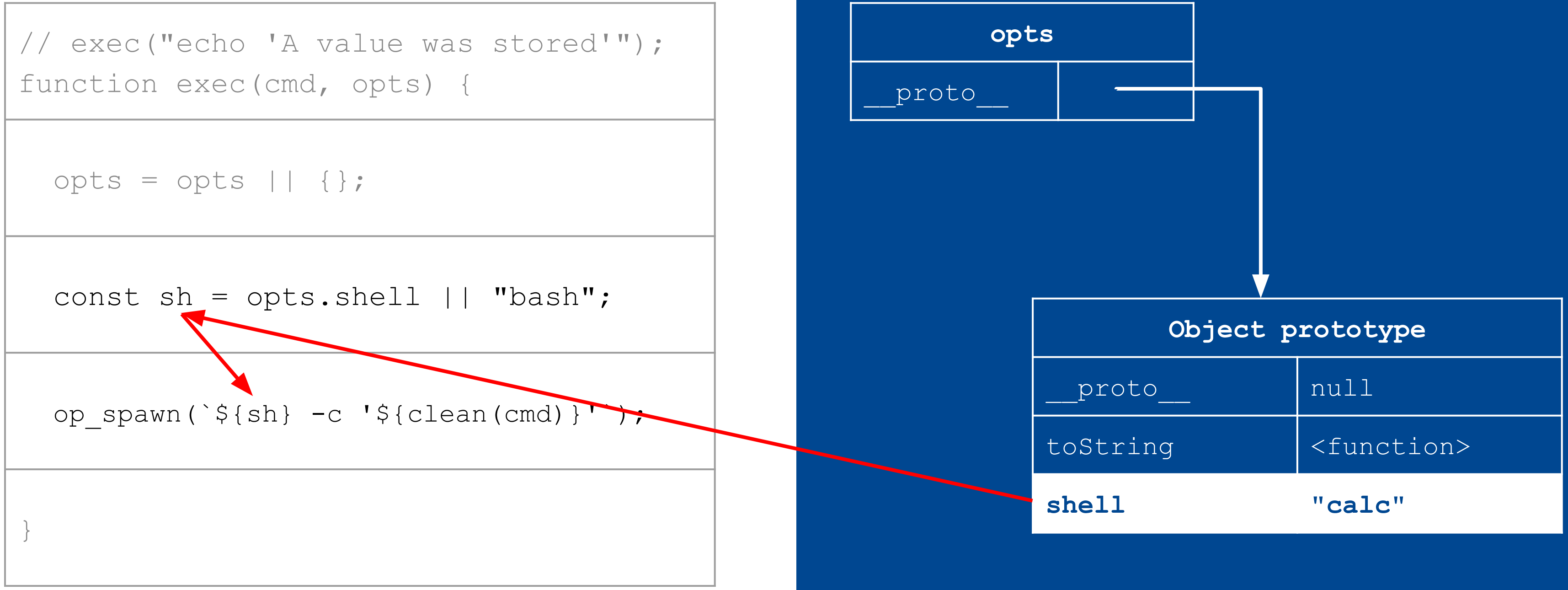
```
// exec("echo 'A value was stored'");
function exec(cmd, opts) {

  opts = opts || {};

  const sh = opts.shell || "bash";

  op_spawn(`${sh} -c '${clean(cmd)}'`);

}
```

State



Program

State

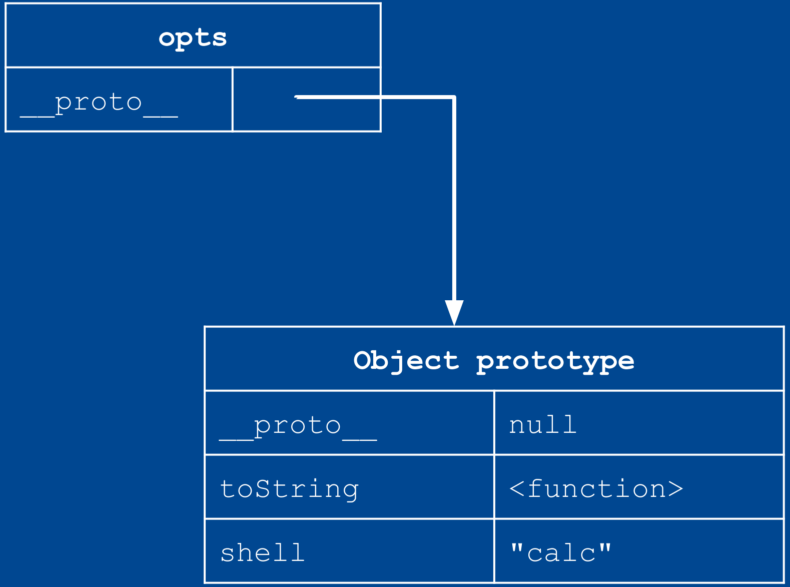


```
// exec("echo 'A value was stored'");
function exec(cmd, opts) {

  opts = opts || {};

  const sh = opts.shell || "bash";

  op_spawn(`${sh} -c '${clean(cmd)}'`);

}
```

Program

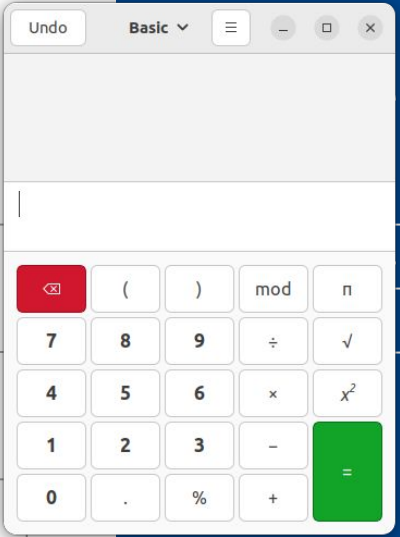
```
// exec("echo 'A value was stored'");
function exec(cmd, opts) {

  opts = opts || {};

  const sh = opts.shell || "bash";

  op_spawn(          calc -c '...'          );

}
```

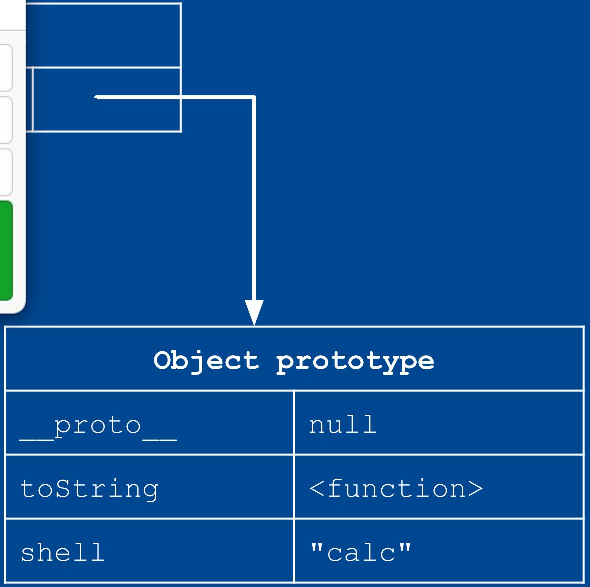
State



Program

```
// exec("echo 'A value was stored'");
function exec(cmd, opts) {

  opts = opts || {};

  const sh = opts.shell || "bash";

  op_spawn(            calc -c '...'            );

}
```





Object prototype	
__proto__	null
toString	<function>
shell	"calc"

Definition: *Universal Gadget*

A *gadget* affecting all programs because it is present in the JavaScript runtime (Node.js or Deno).

RQ: Can we find these automatically and effectively?

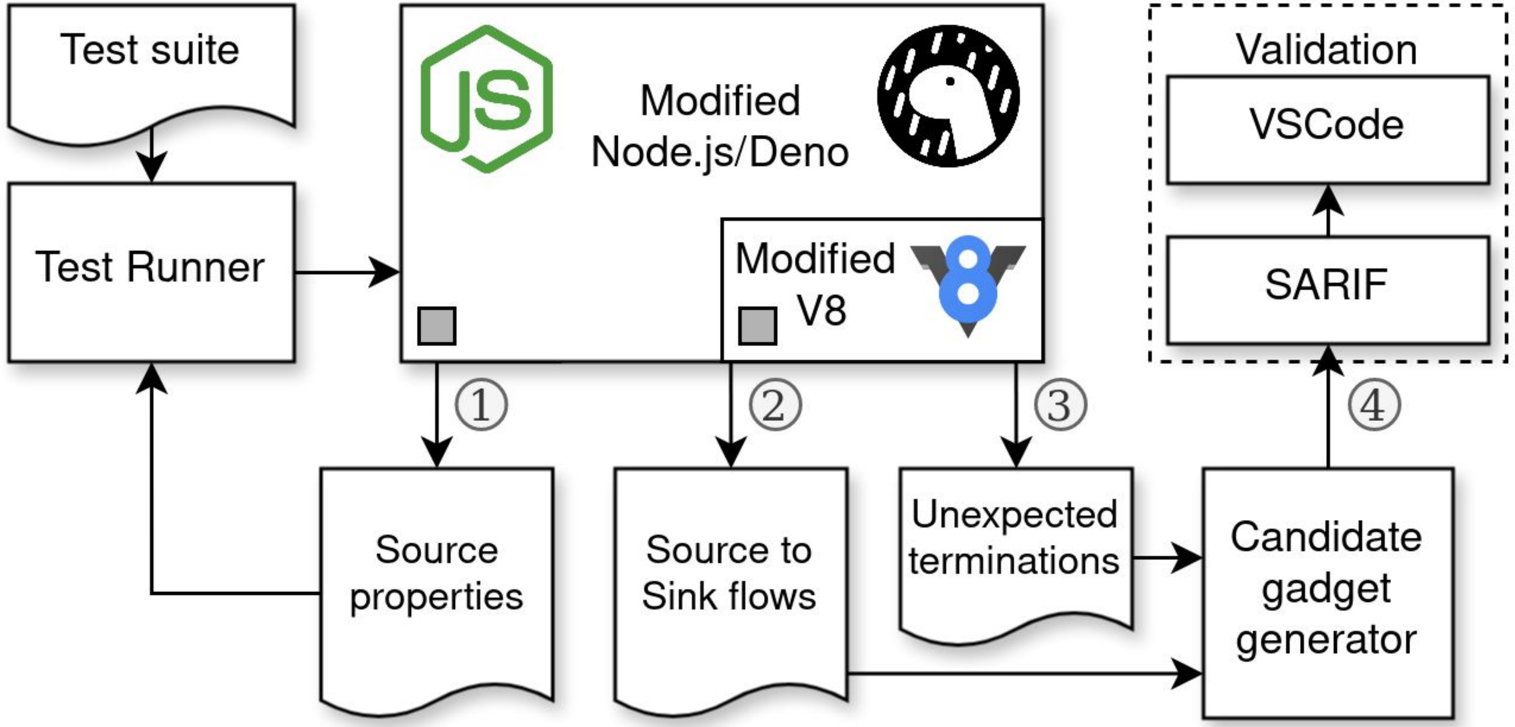
Thread Model

- Server-side JavaScript/TypeScript
- Node.js or Deno
- Assume pollution
- Find gadgets (ACE, SSRF, Privilege Escalation, Path Traversal, etc.)

Preview

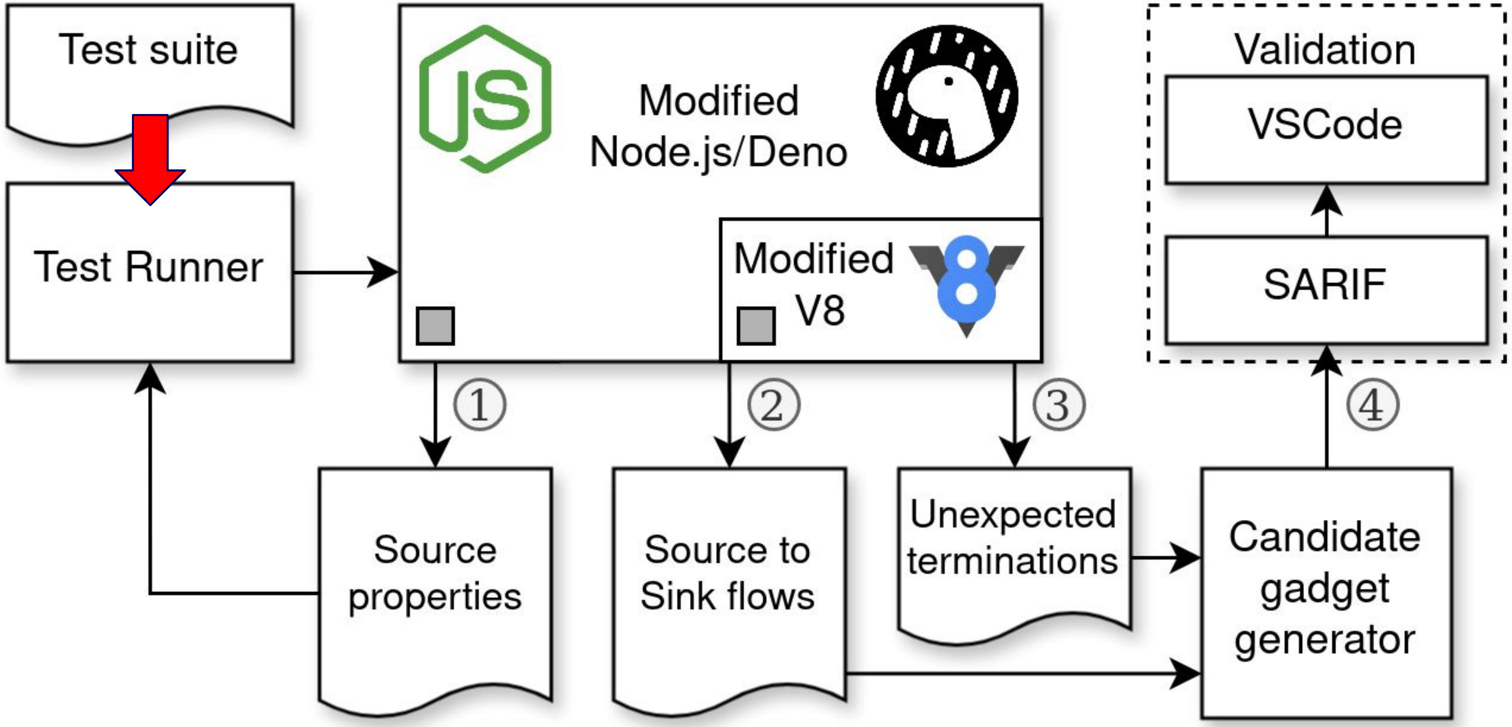
```
function exec(cmd, opts) {
  opts = opts || {};
  const sh = opts.shell || "bash";
  op_spawn(`${sh} -c '${sanitize(cmd)}'`);
}
```

GHunter



Test suite
Test Runner
Modified Node.js/Deno
Modified V8
Validation
VSCode
SARIF
1
2
3
4
Source properties
Source to Sink flows
Unexpected terminations
Candidate gadget generator

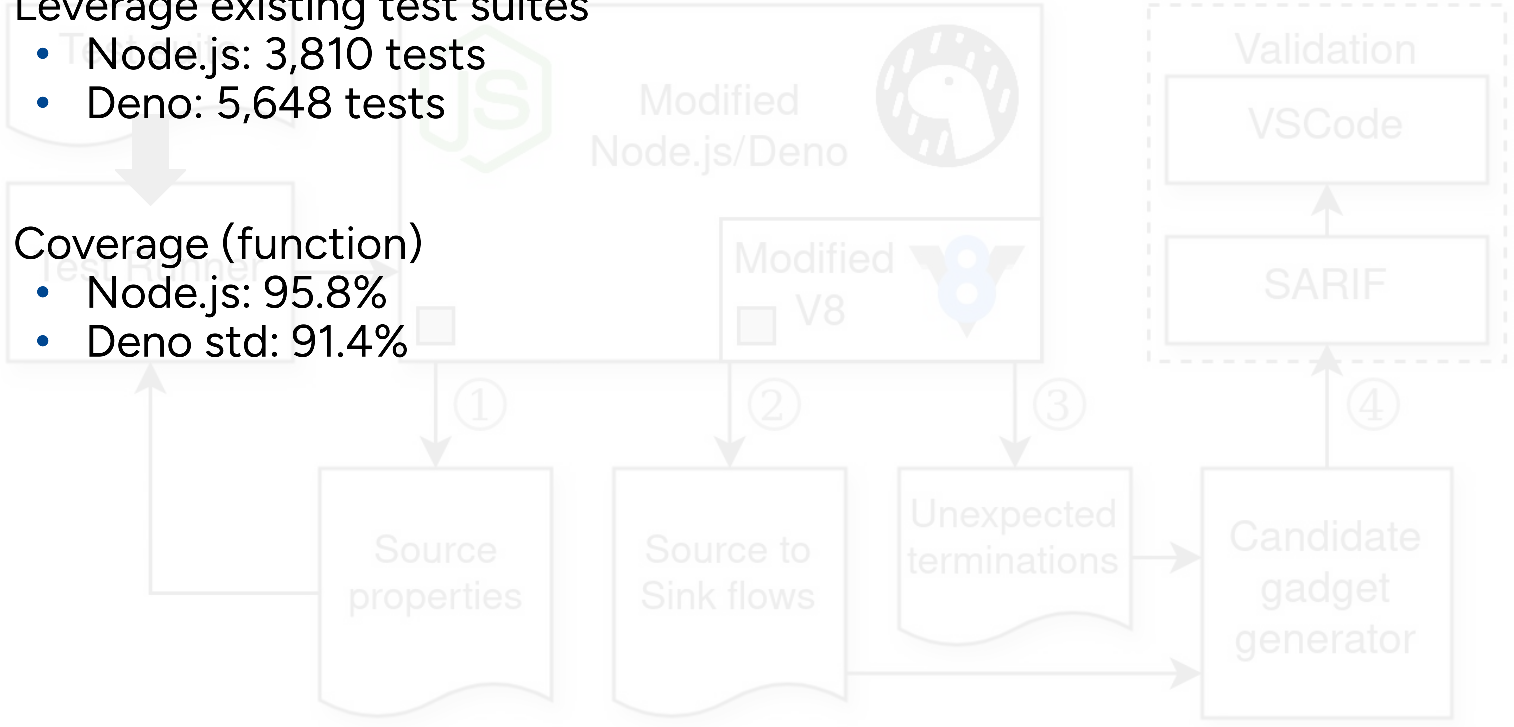
GHunter


Test suite
Test Runner
Modified Node.js/Deno
Modified V8
Validation
VSCode
SARIF
①
②
③
④
Source properties
Source to Sink flows
Unexpected terminations
Candidate gadget generator

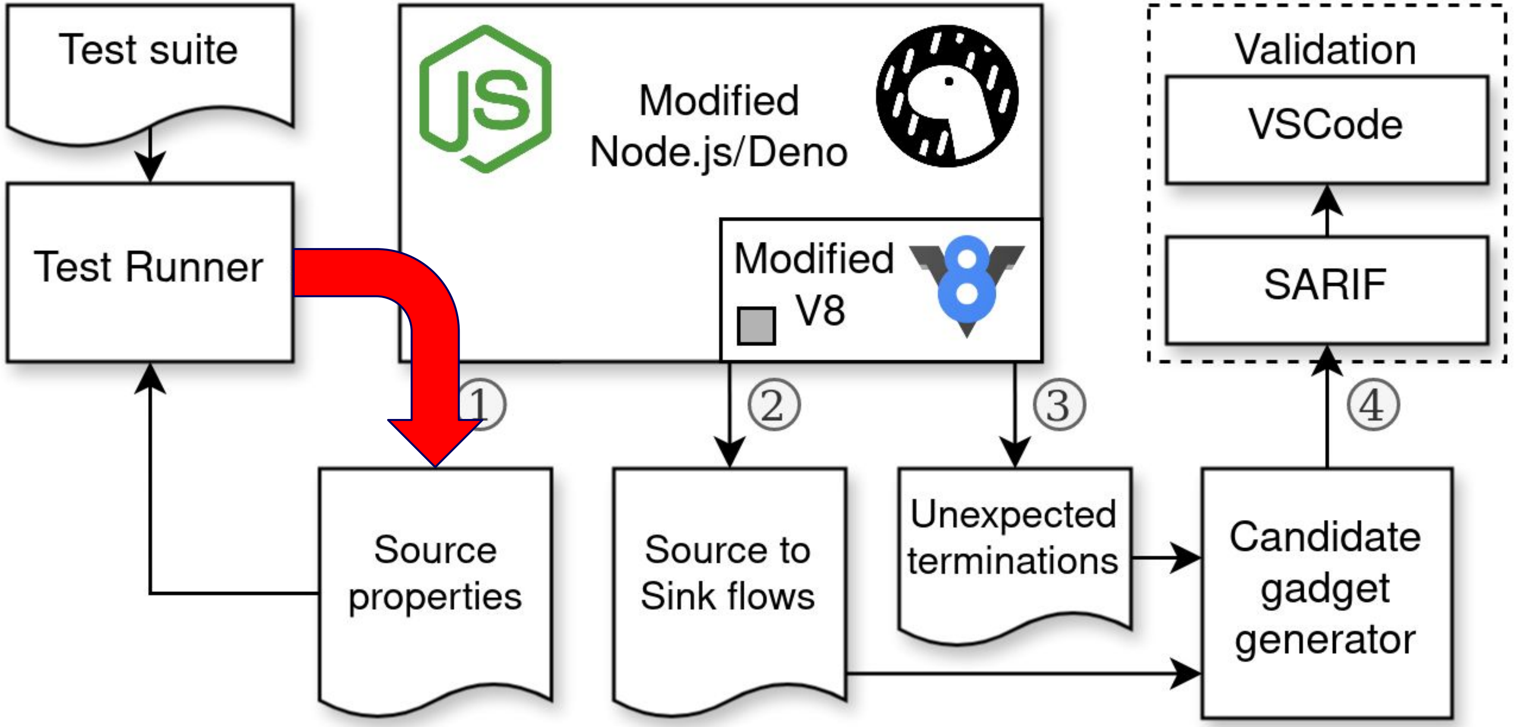
GHunter

- Leverage existing test suites
  - Node.js: 3,810 tests
  - Deno: 5,648 tests

- Coverage (function)
  - Node.js: 95.8%
  - Deno std: 91.4%



GHunter



Test suite
Test Runner
Modified Node.js/Deno
Modified V8
Validation
VSCode
SARIF
1
2
3
4
Source properties
Source to Sink flows
Unexpected terminations
Candidate gadget generator

GHunter

- Modify V8
  - 8 files
  - 233 lines
- Shared by Node.js and Deno

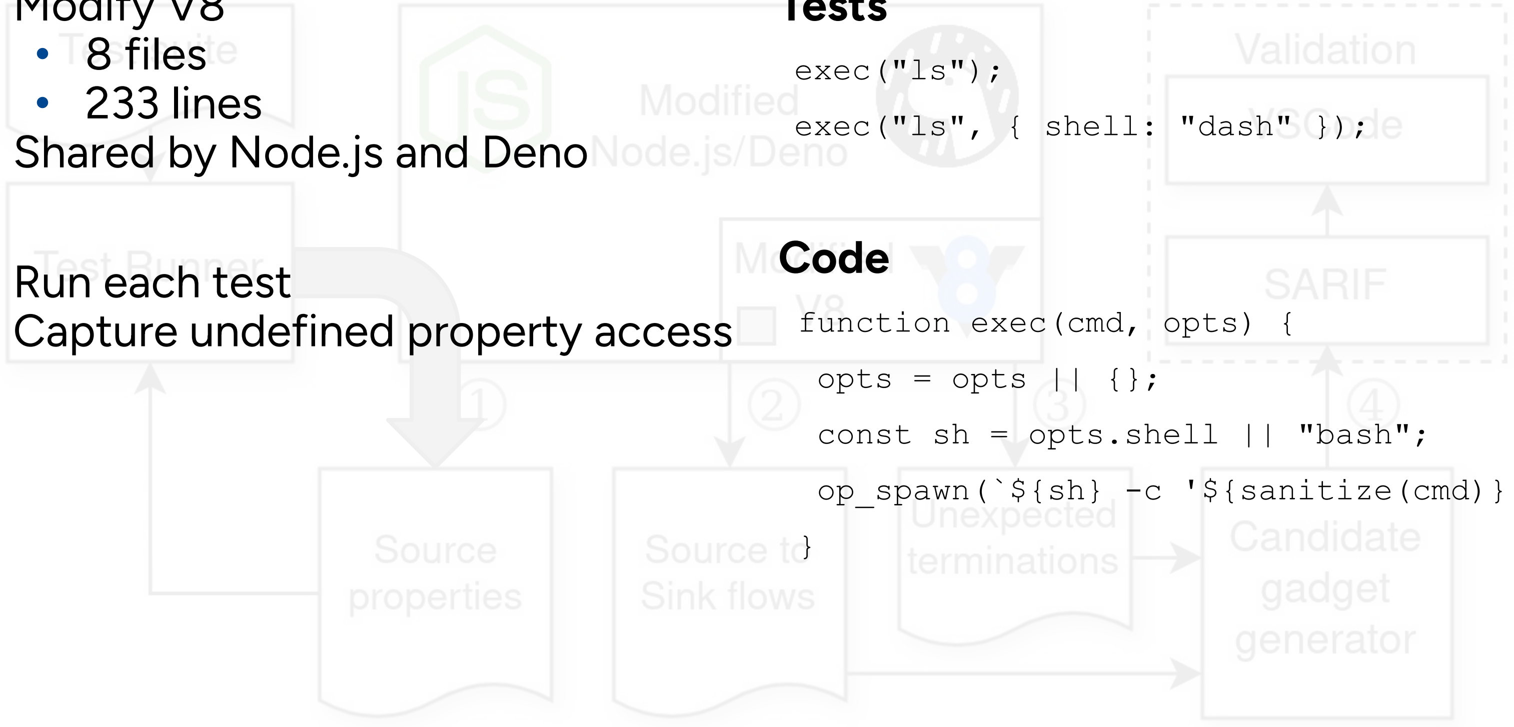
- Run each test
- Capture undefined property access

Tests

```
exec("ls");
exec("ls", { shell: "dash" });
```

Code

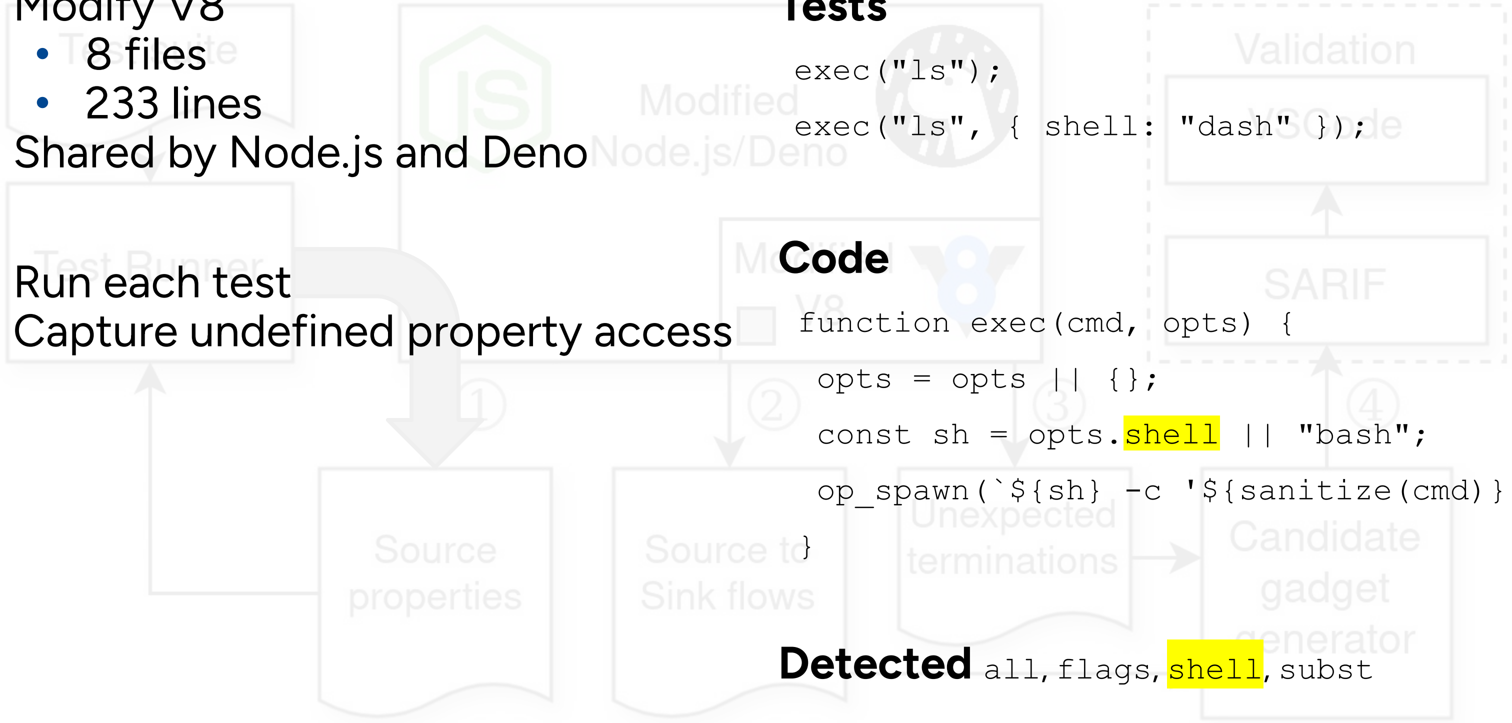
```
function exec(cmd, opts) {
 opts = opts || {};
 const sh = opts.shell || "bash";
 op_spawn(`${sh} -c '${sanitize(cmd)}'`);
}
```



GHunter

- Modify V8
  - 8 files
  - 233 lines
- Shared by Node.js and Deno

- Run each test
- Capture undefined property access



Tests

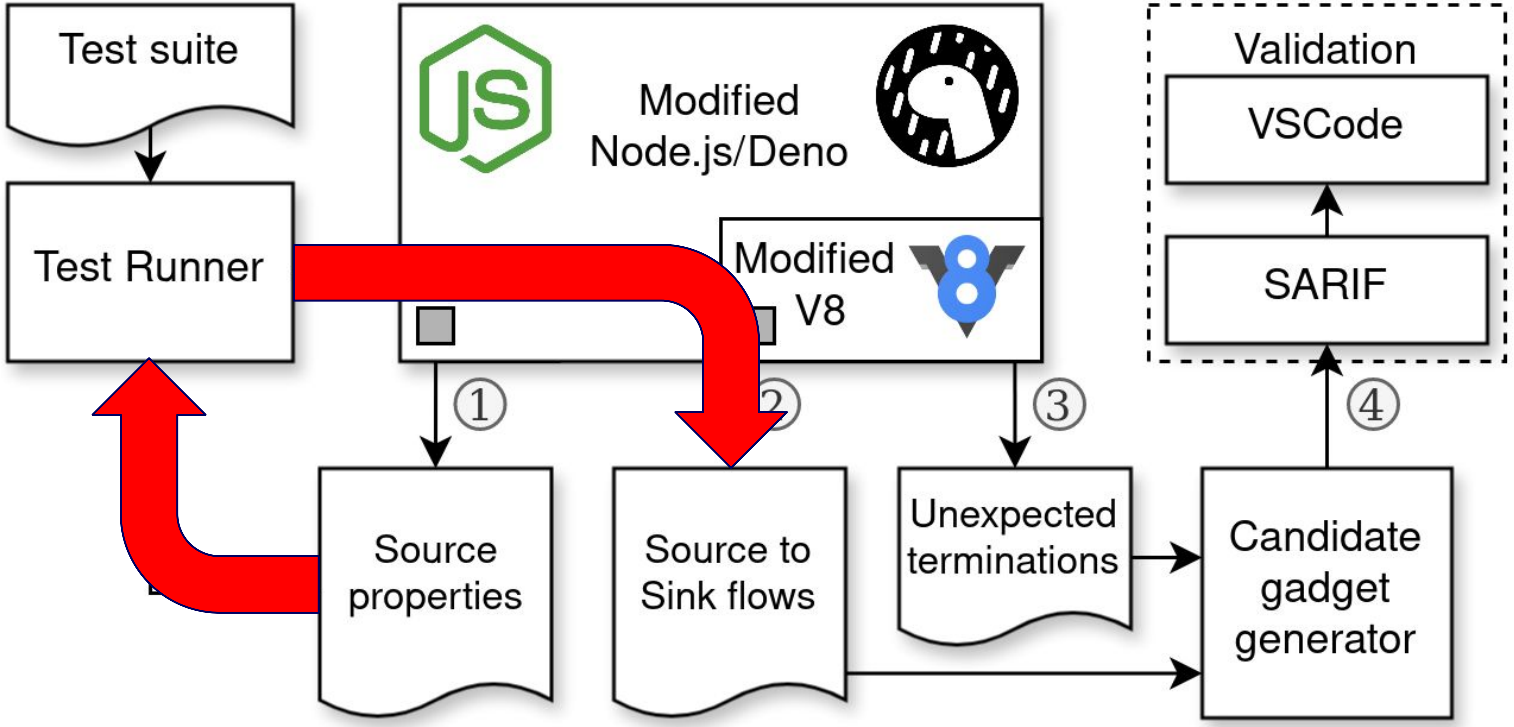
```
exec("ls");
exec("ls", { shell: "dash" });
```

Code

```
function exec(cmd, opts) {
  opts = opts || {};
  const sh = opts.shell || "bash";
  op_spawn(`${sh} -c '${sanitize(cmd)}'`);
}
```

Detected `all, flags, shell, subst`

GHunter



Test suite
Test Runner
Modified Node.js/Deno
Modified V8
Validation
VSCode
SARIF
①
②
③
④
Source properties
Source to Sink flows
Unexpected terminations
Candidate gadget generator

GHunter

- Sinks: calls into native code
- Modify binding code
- Monitor dynamic code evaluation (`eval()`)

- Run each test N times
- Pollute 1 property per run

- Lightweight taint tracking
  - Pollute with taint value
  - Detect taint value in sinks

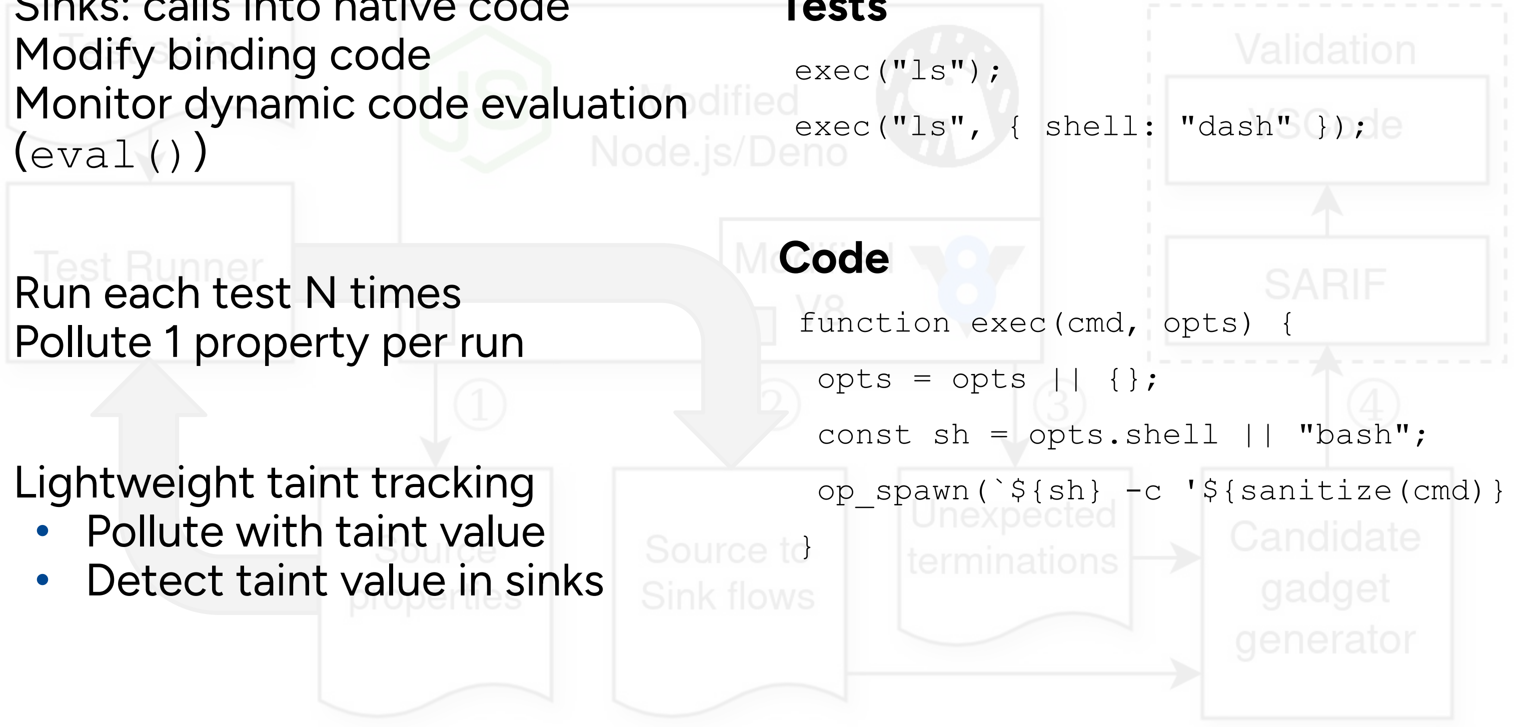
Detected `all`, `flags`, `shell`, `subst`

Tests

```
exec("ls");
exec("ls", { shell: "dash" });
```

Code

```
function exec(cmd, opts) {
  opts = opts || {};
  const sh = opts.shell || "bash";
  op_spawn(`${sh} -c '${sanitize(cmd)}'`);
}
```



GHunter

- Sinks: calls into native code
- Modify binding code
- Monitor dynamic code evaluation (`eval()`)

- Run each test N times
- Pollute 1 property per run

- Lightweight taint tracking
  - Pollute with taint value
  - Detect taint value in sinks

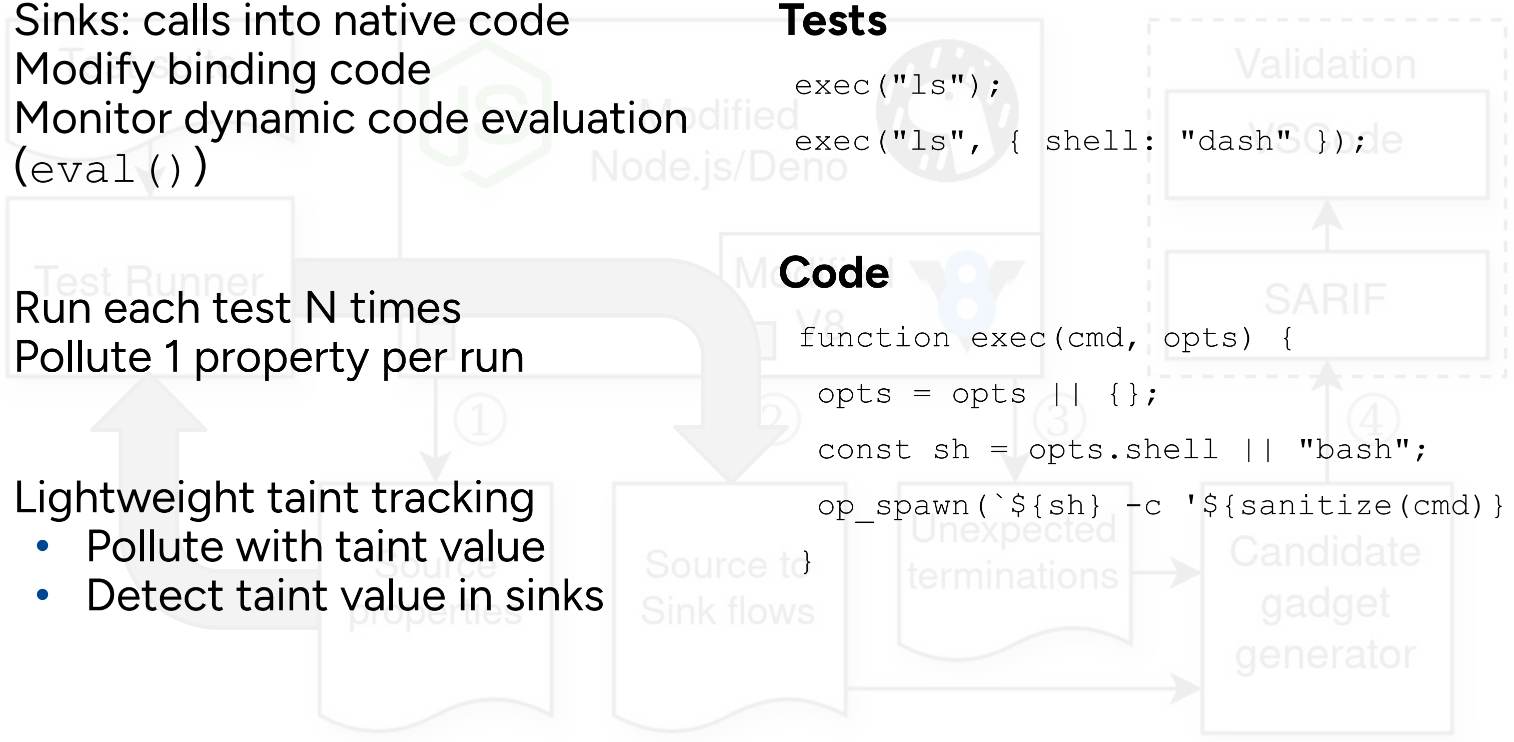
Detected `all`, `flags`, `shell`, `subst`

Tests

```
exec("ls");
exec("ls", { shell: "dash" });
```

Code

```
function exec(cmd, opts) {
  opts = opts || {};
  const sh = opts.shell || "bash";
  op_spawn(`${sh} -c '${sanitize(cmd)}'`);
}
```



GHunter

- Sinks: calls into native code
- Modify binding code
- Monitor dynamic code evaluation (`eval()`)

- Run each test N times
- Pollute 1 property per run

- Lightweight taint tracking
  - Pollute with taint value
  - Detect taint value in sinks

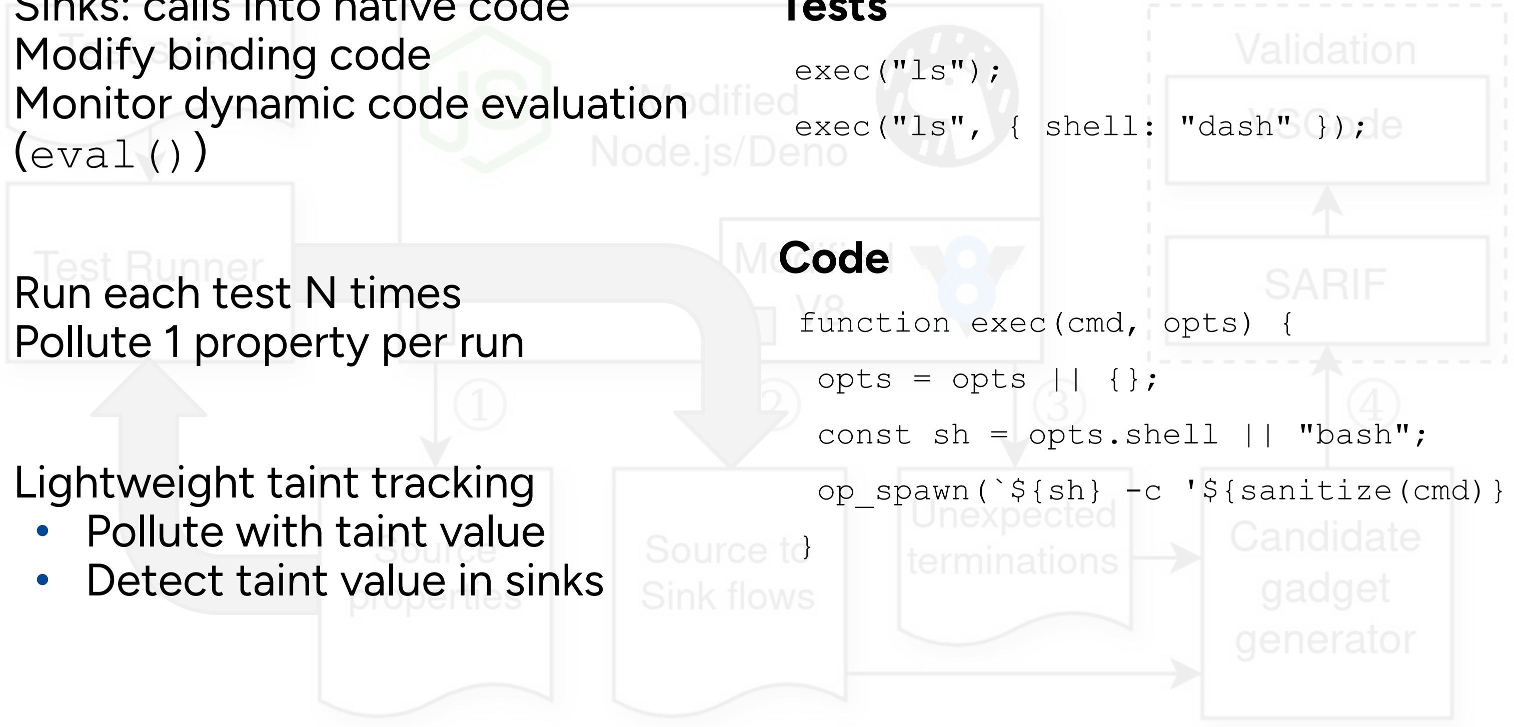
Detected `all`, `flags`, `shell`, `subst`

Tests

```
exec("ls");
exec("ls", { shell: "dash" });
```

Code

```
function exec(cmd, opts) {
  opts = opts || {};
  const sh = opts.shell || "bash";
  op_spawn(`${sh} -c '${sanitize(cmd)}'`);
}
```



GHunter

- Sinks: calls into native code
- Modify binding code
- Monitor dynamic code evaluation (`eval()`)

- Run each test N times
- Pollute 1 property per run

- Lightweight taint tracking
  - Pollute with taint value
  - Detect taint value in sinks

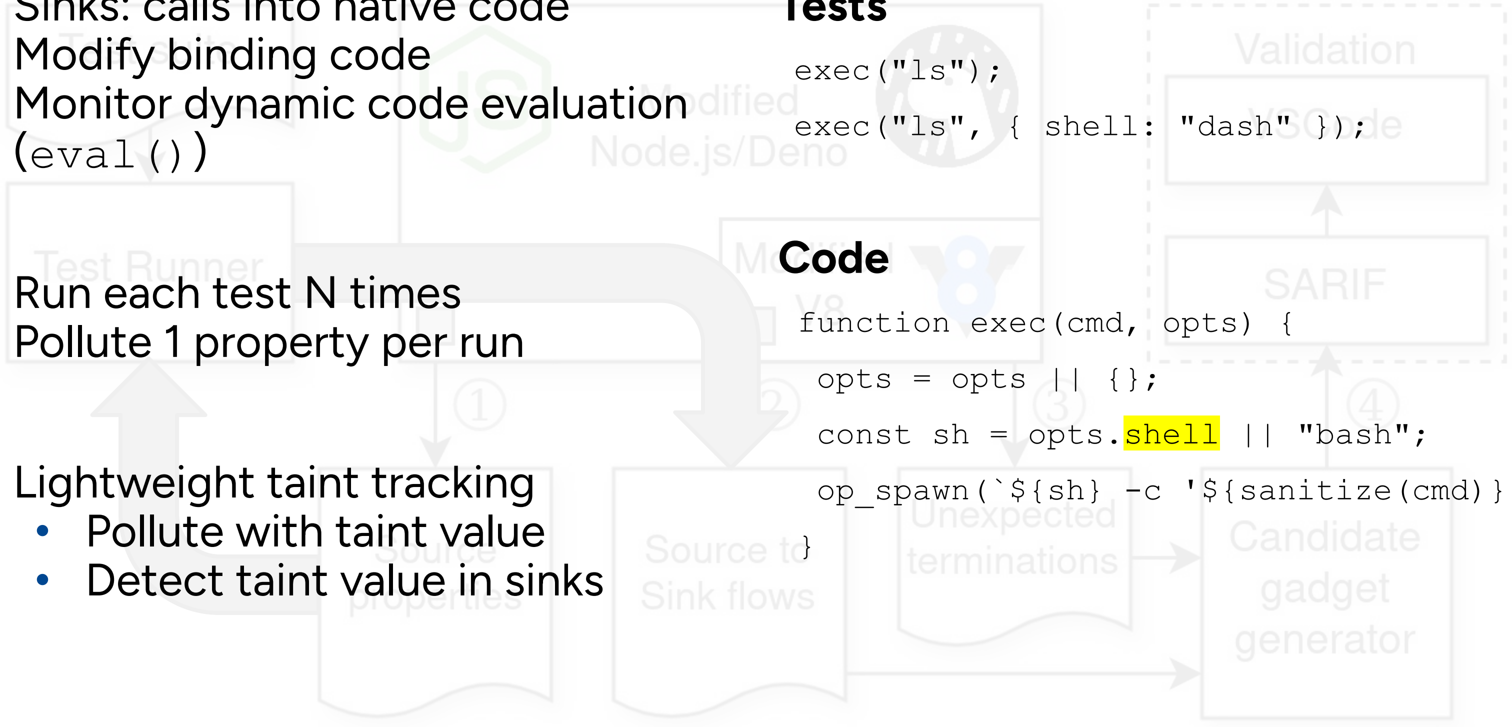
Detected `all`, `flags`, `shell`, `subst`

Tests

```
exec("ls");
exec("ls", { shell: "dash" });
```

Code

```
function exec(cmd, opts) {
  opts = opts || {};
  const sh = opts.shell || "bash";
  op_spawn(`${sh} -c '${sanitize(cmd)}'`);
}
```



GHunter

- Sinks: calls into native code
- Modify binding code
- Monitor dynamic code evaluation (`eval()`)

- Run each test N times
- Pollute 1 property per run

- Lightweight taint tracking
  - Pollute with taint value
  - Detect taint value in sinks

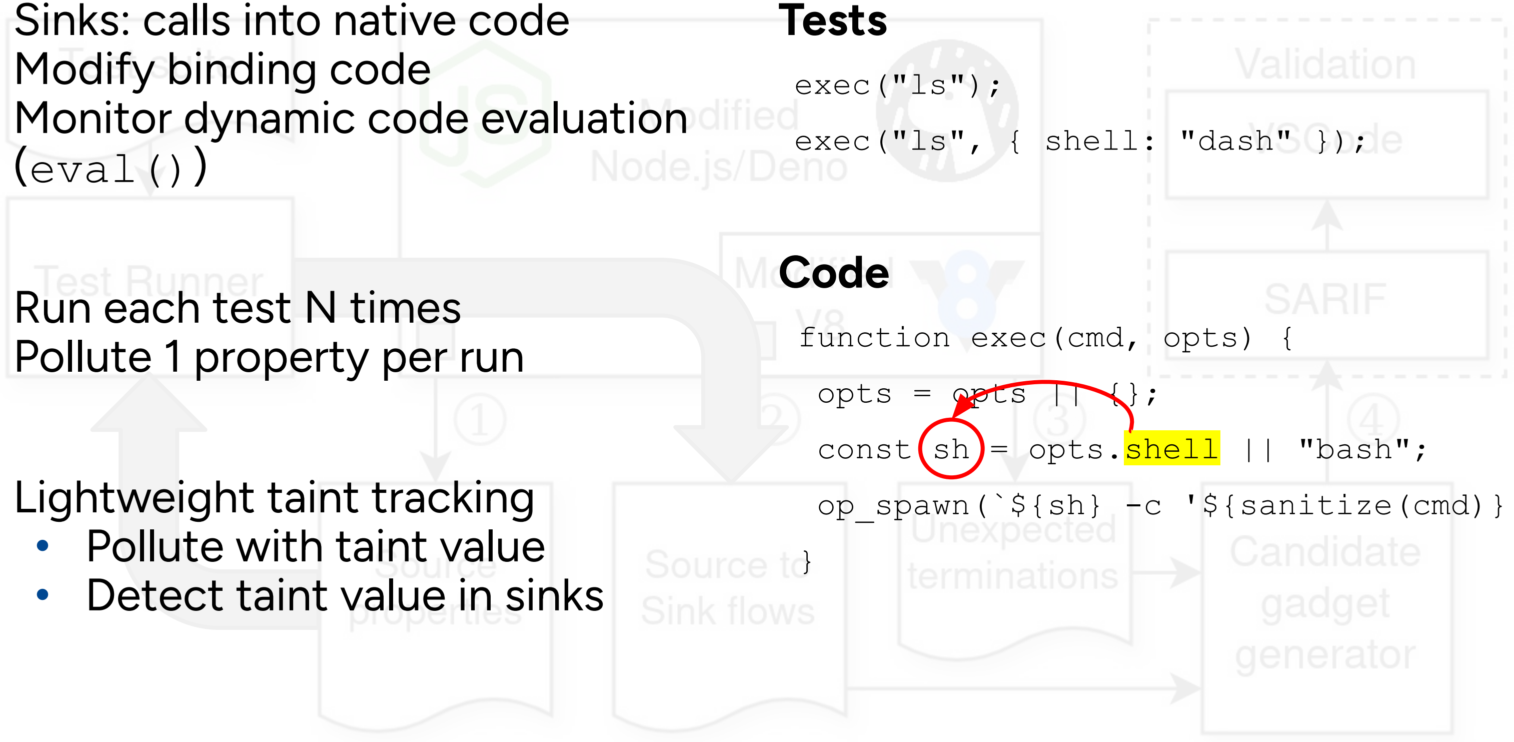
Detected `all`, `flags`, `shell`, `subst`

Tests

```
exec("ls");
exec("ls", { shell: "dash" });
```

Code

```
function exec(cmd, opts) {
  opts = opts || {};
  const sh = opts.shell || "bash";
  op_spawn(`${sh} -c '${sanitize(cmd)}'`);
}
```



GHunter

- Sinks: calls into native code
- Modify binding code
- Monitor dynamic code evaluation (`eval()`)

- Run each test N times
- Pollute 1 property per run

- Lightweight taint tracking
  - Pollute with taint value
  - Detect taint value in sinks

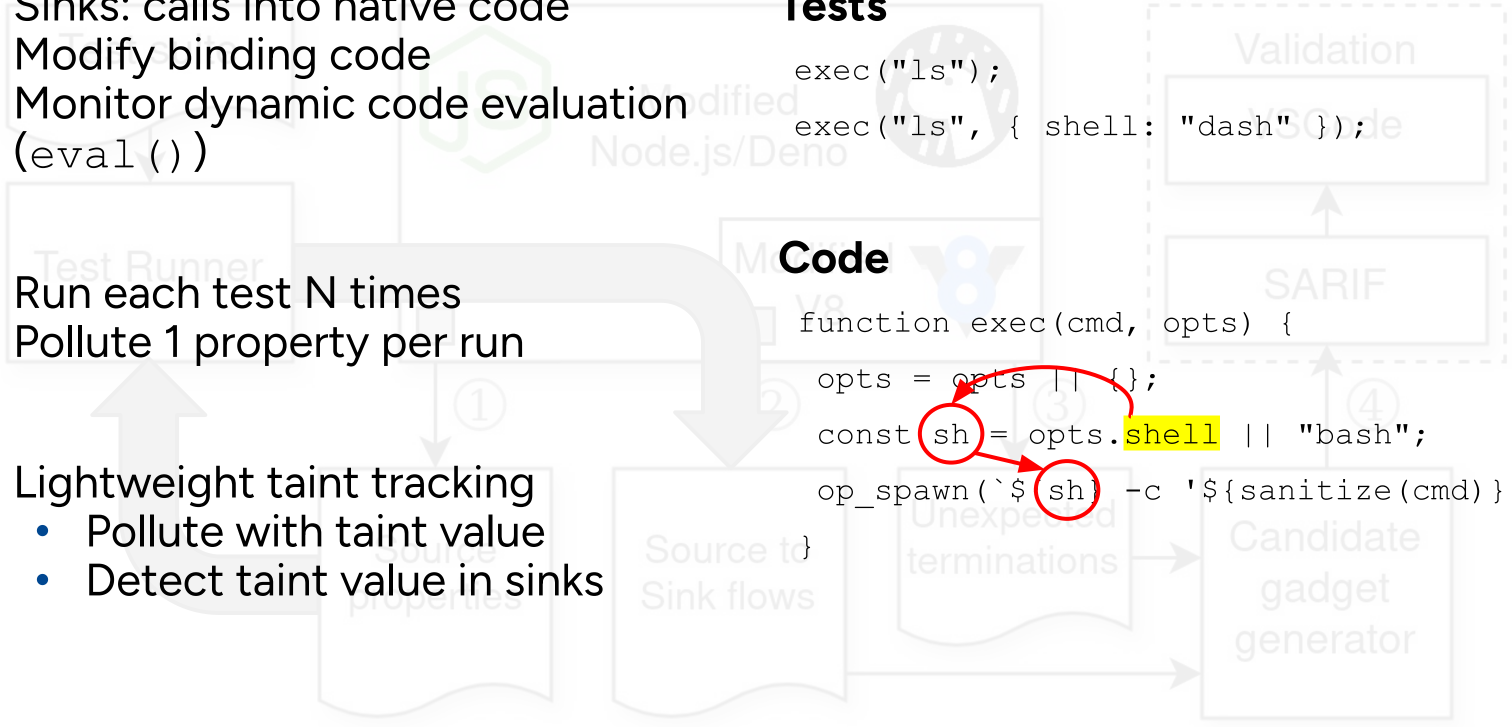
Detected `all`, `flags`, `shell`, `subst`

Tests

```
exec("ls");
exec("ls", { shell: "dash" });
```

Code

```
function exec(cmd, opts) {
  opts = opts || {};
  const sh = opts.shell || "bash";
  op_spawn(`${sh} -c '${sanitize(cmd)}'`);
}
```



GHunter

- Sinks: calls into native code
- Modify binding code
- Monitor dynamic code evaluation (`eval()`)

- Run each test N times
- Pollute 1 property per run

- Lightweight taint tracking
  - Pollute with taint value
  - Detect taint value in sinks

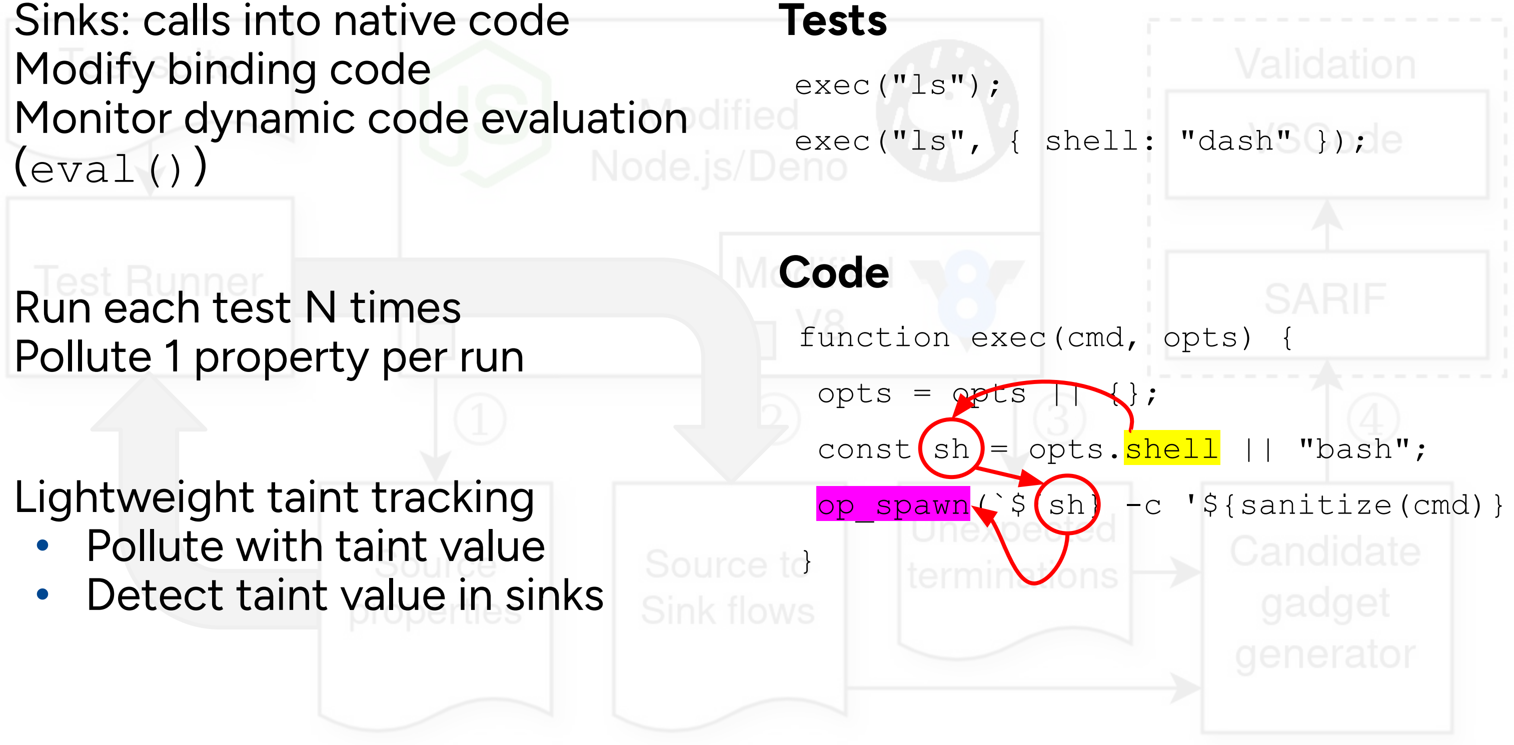
Detected `all`, `flags`, `shell`, `subst`

Tests

```
exec("ls");
exec("ls", { shell: "dash" });
```

Code

```
function exec(cmd, opts) {
 opts = opts || {};
 const sh = opts.shell || "bash";
 op_spawn(`${sh} -c '${sanitize(cmd)}'`);
}
```



GHunter

- Sinks: calls into native code
- Modify binding code
- Monitor dynamic code evaluation (`eval()`)

- Run each test N times
- Pollute 1 property per run

- Lightweight taint tracking
    - Pollute with taint value
    - Detect taint value in sinks

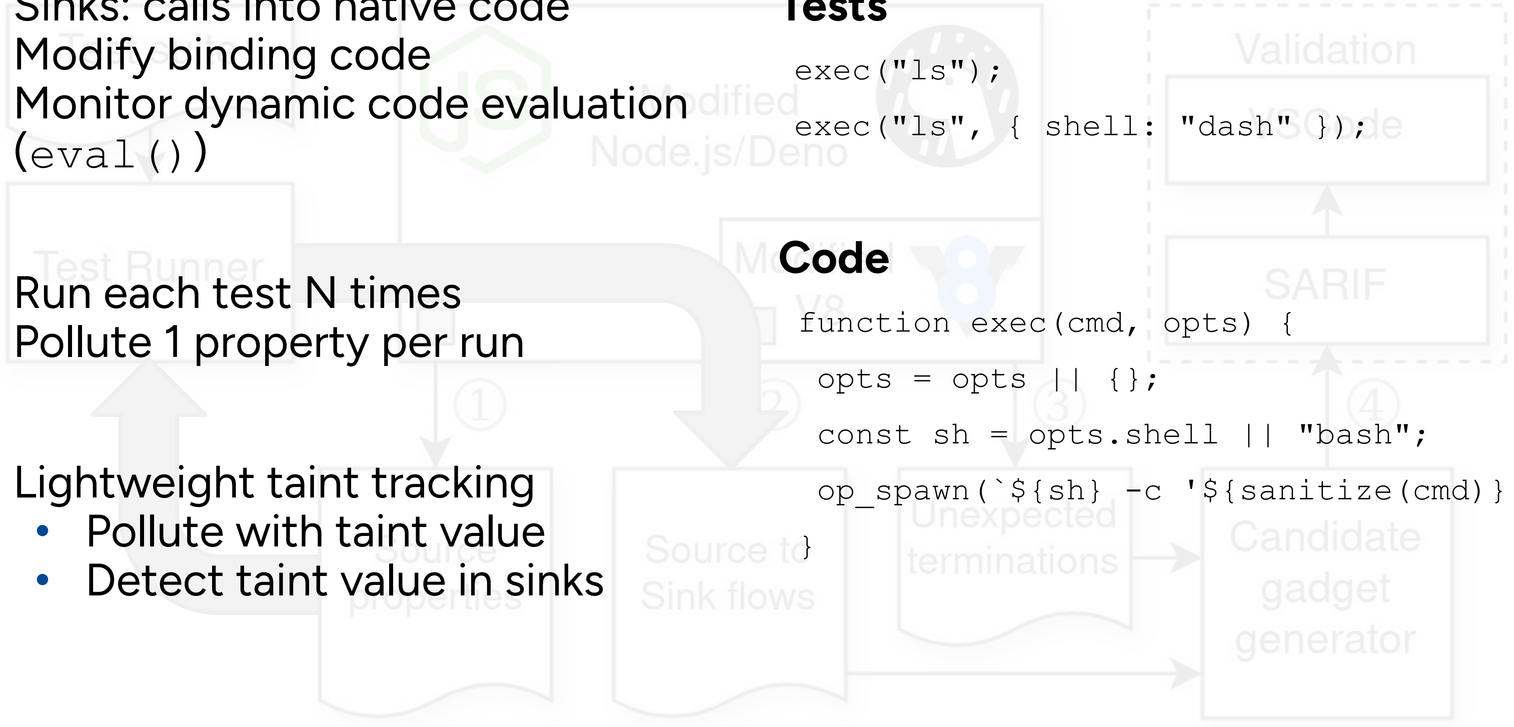
Detected `all`, `flags`, `shell`, `subst`

Tests

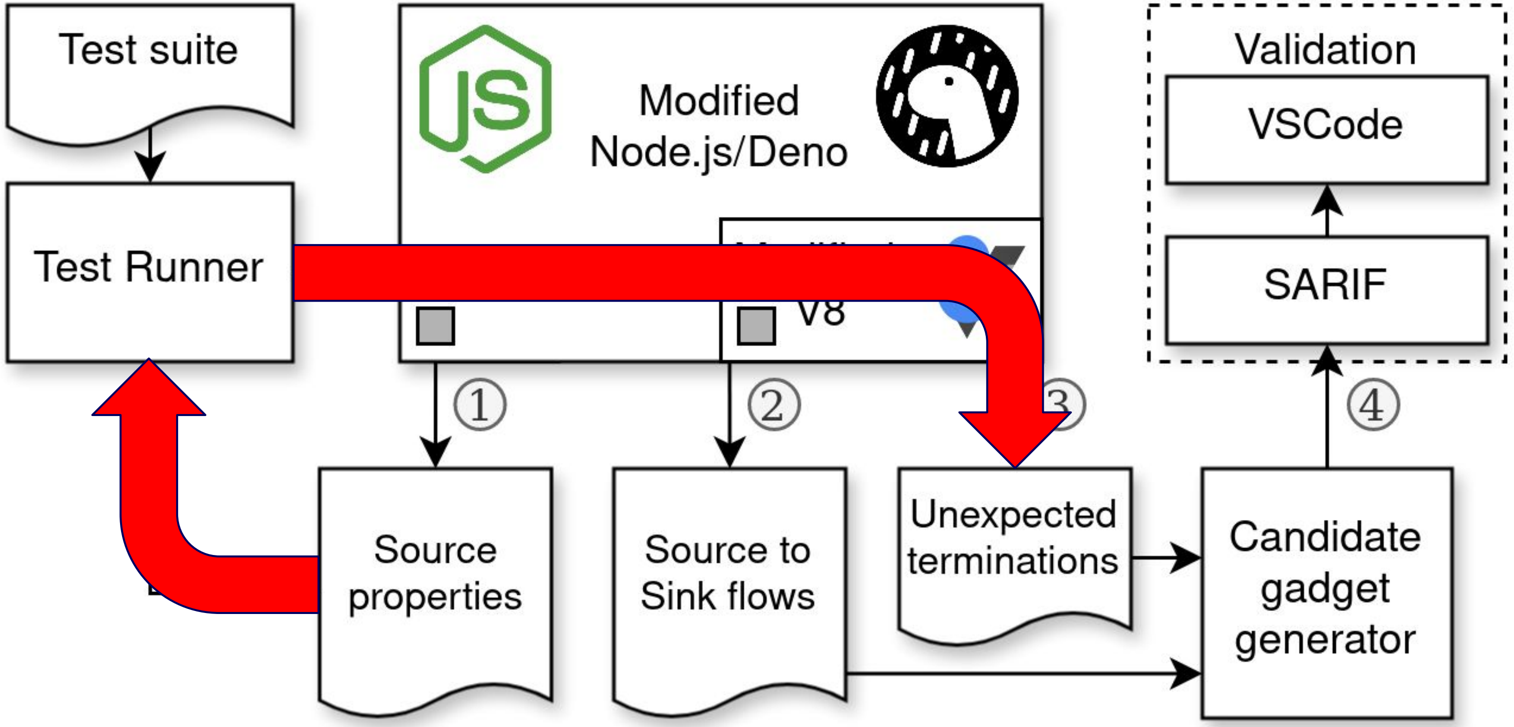
```
exec("ls");
exec("ls", { shell: "dash" });
```

Code

```
function exec(cmd, opts) {
  opts = opts || {};
  const sh = opts.shell || "bash";
  op_spawn(`${sh} -c '${sanitize(cmd)}'`);
}
```



GHunter



Test suite
Test Runner
Modified Node.js/Deno
V8
1
2
3
4
Source properties
Source to Sink flows
Unexpected terminations
Candidate gadget generator
Validation
VSCode
SARIF

GHunter

- Unmodified engine
- Run each test N times
- Pollute 1 property per run
- Detect unexpected crashes and timeouts

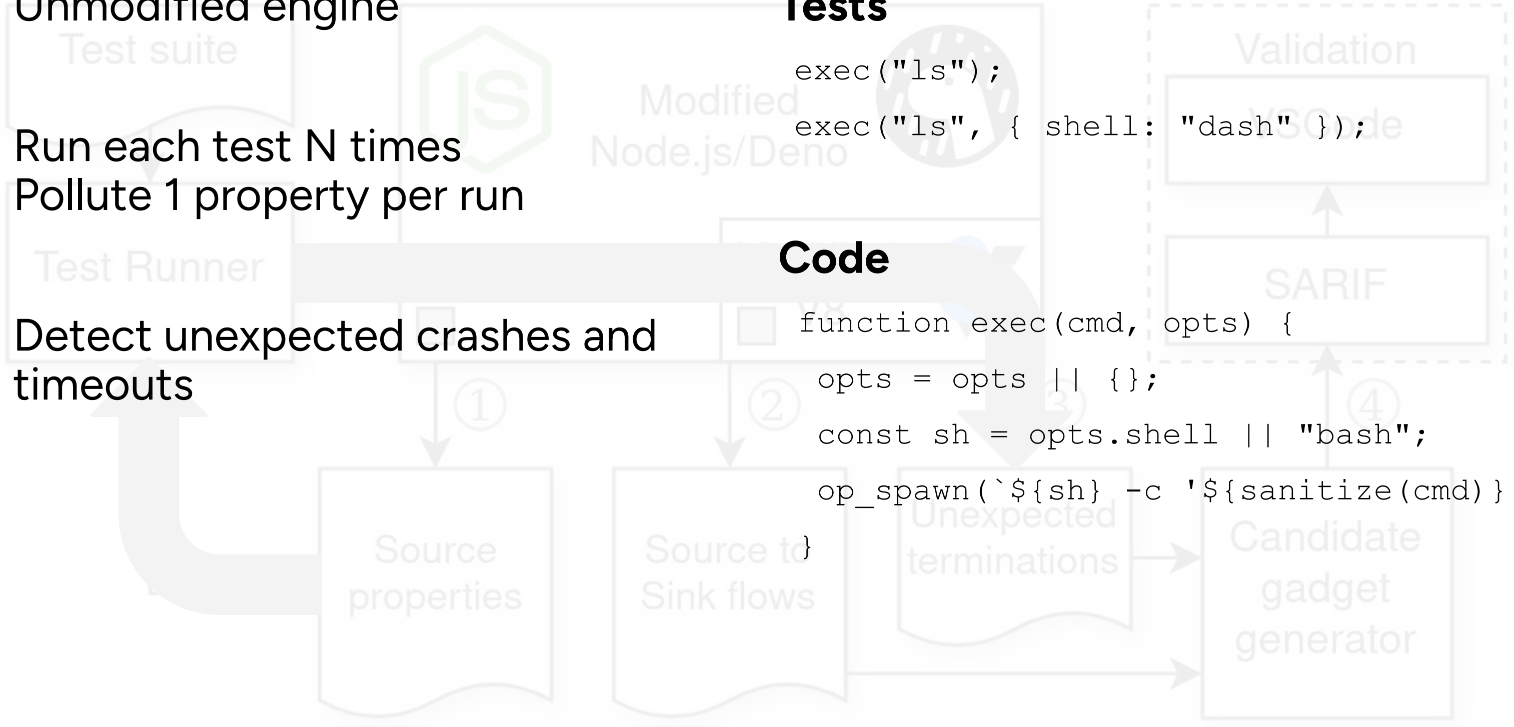
Detected `all`, `flags`, `shell`, `subst`

Tests

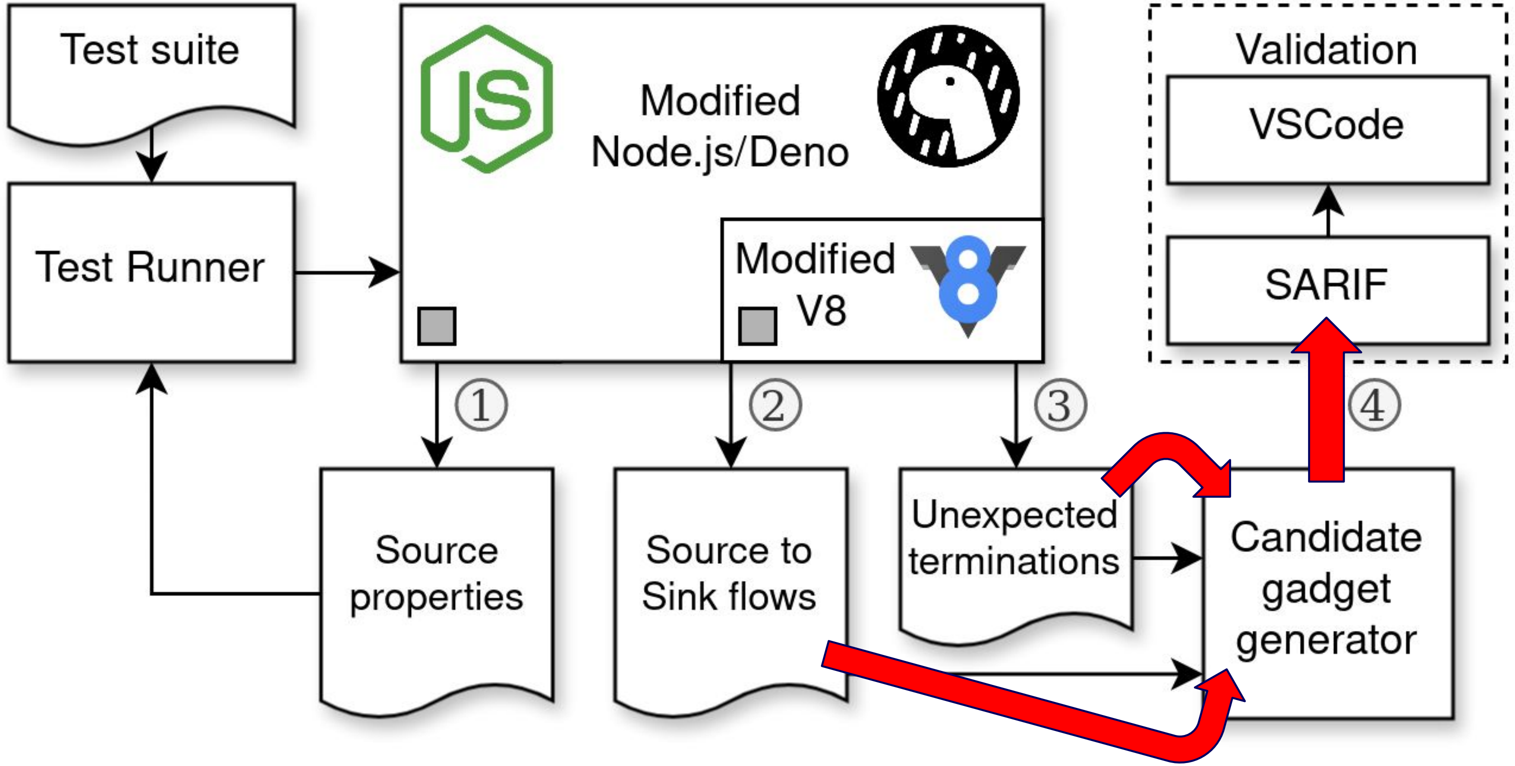
```
exec("ls");
exec("ls", { shell: "dash" });
```

Code

```
function exec(cmd, opts) {
  opts = opts || {};
  const sh = opts.shell || "bash";
  op_spawn(`${sh} -c '${sanitize(cmd)}'`);
}
```



GHunter



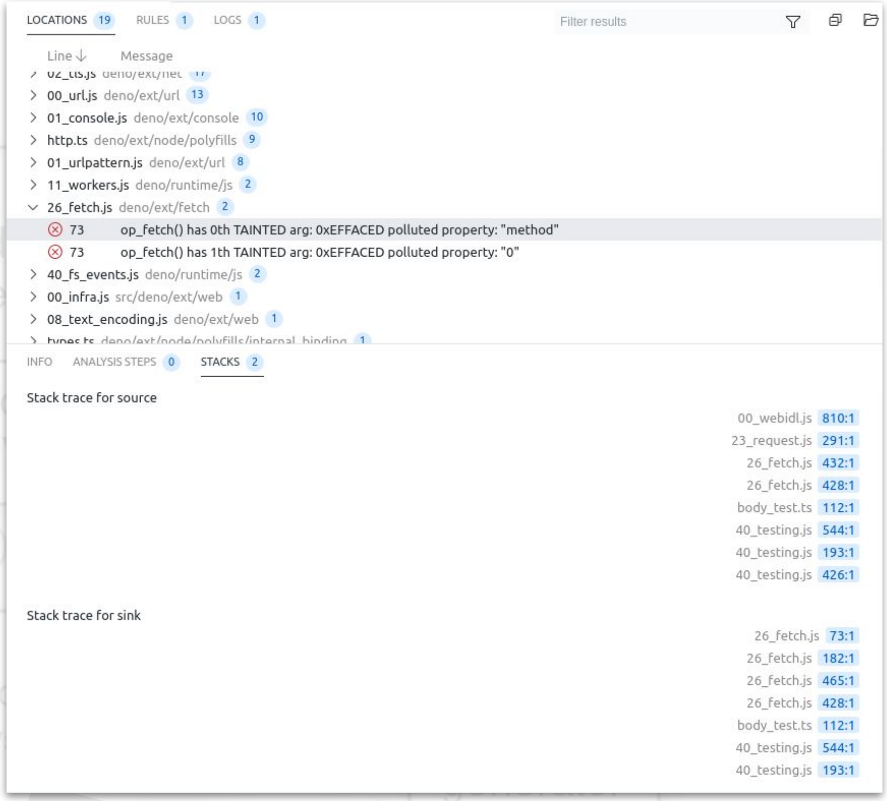
Test suite
Test Runner
Modified Node.js/Deno
Modified V8
Validation
VSCode
SARIF
①
②
③
④
Source properties
Source to Sink flows
Unexpected terminations
Candidate gadget generator

GHunter

- Preprocess (duplicates, uninteresting sinks)
- Generate SARIF file

- Manually review SARIF file
- Manually construct gadget proof of concepts

- 31 hours for Node.js
- 15 hours for Deno



```
02_tls.js deno/ext/net 17
00_url.js deno/ext/url 13
01_console.js deno/ext/console 10
http.ts deno/ext/node/polyfills 9
01_urlpattern.js deno/ext/url 8
11_workers.js deno/runtime/js 2
26_fetch.js deno/ext/fetch 2
  73   op_fetch() has 0th TAINTED arg: 0xEFFACED polluted property: "method"
  73   op_fetch() has 1th TAINTED arg: 0xEFFACED polluted property: "0"
40_fs_events.js deno/runtime/js 2
00_infra.js src/deno/ext/web 1
08_text_encoding.js deno/ext/web 1
types.ts deno/ext/node/polyfills/internal_binding 1
```

Stack trace for source

```
00_webidl.js 810:1
23_request.js 291:1
26_fetch.js 432:1
26_fetch.js 428:1
body_test.ts 112:1
40_testing.js 544:1
40_testing.js 193:1
40_testing.js 426:1
```

Stack trace for sink

```
26_fetch.js 73:1
26_fetch.js 182:1
26_fetch.js 465:1
26_fetch.js 428:1
body_test.ts 112:1
40_testing.js 544:1
40_testing.js 193:1
```

Node.js

Total	56
Arbitrary Code/Command Execution	14
Server Side Request Forgery	6
Privilege Escalation	7
Cryptographic Downgrade	2
…	…

Deno

Total	67
Arbitrary Code/Command Execution	5
Server Side Request Forgery	3
Privilege Escalation	24
Cryptographic Downgrade	0
…	…

Examples

Node.js - ACE

```
// Gadget
import('./any_file.mjs')
```

Deno - SSRF

Examples

Node.js - ACE

```
// Pollution
Object.prototype.source =
            'console.log("foobar")'



// Gadget
import('./any_file.mjs')
```

Deno - SSRF

Examples

Node.js - ACE

```
// Pollution
Object.prototype.source =
          'console.log("foobar")'
```

```
// Gadget
```
~~import('./any_file.mjs')~~
```
console.log("foobar")
```

Deno - SSRF

Examples

Node.js - ACE

```
// Pollution
Object.prototype.source =
          'console.log("foobar")'
```

```
// Gadget
~~import('./any_file.mjs')~~
console.log("foobar")
```

Deno - SSRF

```
// Gadget
fetch('http://example.com')
```

Examples

Node.js - ACE

```
// Pollution
Object.prototype.source =
          'console.log("foobar")'
```

```
// Gadget
```
~~import('./any_file.mjs')~~
```
console.log("foobar")
```

Deno - SSRF

```
// Pollution
Object.prototype[0] = 'http://fake.com'
Object.prototype.method = 'POST'
Object.prototype.body = '{"foo":"bar"}'
Object.prototype.headers =
   {'content-type': 'application/json'}

// Gadget
fetch('http://example.com')
```

Examples

Node.js - ACE

```
// Pollution
Object.prototype.source =
          'console.log("foobar")'
```

```
// Gadget
~~import('./any_file.mjs')~~
console.log("foobar")
```

Deno - SSRF

```
// Pollution
Object.prototype[0] = 'http://fake.com'
Object.prototype.method = 'POST'
Object.prototype.body = '{"foo":"bar"}'
Object.prototype.headers =
   {'content-type': 'application/json'}
```

```
// Gadget
~~fetch('http://example.com')~~
fetch('http://fake.com', {
 method: 'POST', body: '{"foo":"bar"}',
 header: {...}
})
```

Conclusion

- We have presented a semi-automated pipeline able to find universal gadgets

  github.com/KTH-LangSec/ghunter

- We have used GHunter in a study of universal gadgets, finding a total 123 exploitable gadgets

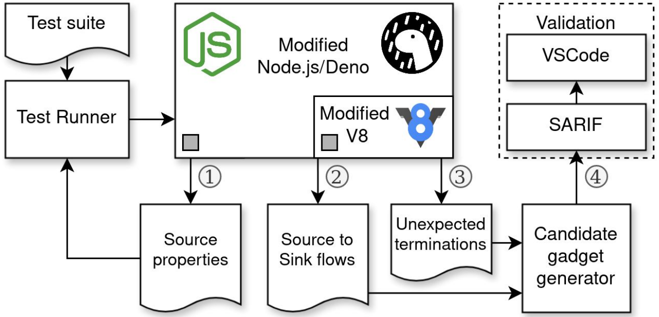
  github.com/KTH-LangSec/server-side-prototype-pollution

- More in the paper:
  - Comparison to Silent Spring [1]
  - Systematize existing mitigation
  - New high-severity exploit in Kibana

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED



GHunter

Gadgets

[1]: Shcherbakov, Mikhail, Musard Balliu, and Cristian-Alexandru Staicu. "Silent spring: Prototype pollution leads to remote code execution in Node. js." 32nd USENIX Security Symposium (USENIX Security 23). 2023.

KTH
VETENSKAP
OCH KONST

Vulnerability Scope

- Arbitrary Code/Command Execution
- Server Side Request Forgery
- Privilege Escalation
- Cryptographic Downgrade
- Path Traversal
- Unauthorized Modifications
- Log Pollution
- Denial of Service
    - *Excluding runtime exceptions*

On Node.js v21.0.0 using GHunter's findings as ground truth

GHunter

Candidates	**29**
True Positive False Positive	**18** **11**
False Negative	**2**
Precision	**0.62**
Recall	**0.90**

Silent Spring [1]

Candidates	**55**
True Positive False Positive	**10** **45**
False Negative	**10**
Precision	**0.18**
Recall	**0.50**

[1]: Shcherbakov, Mikhail, Musard Balliu, and Cristian-Alexandru Staicu. "Silent spring: Prototype pollution leads to remote code execution in Node. js." 32nd USENIX Security Symposium (USENIX Security 23). 2023.

On Node.js v16.13.1 using Silent Spring's findings as ground truth

GHunter

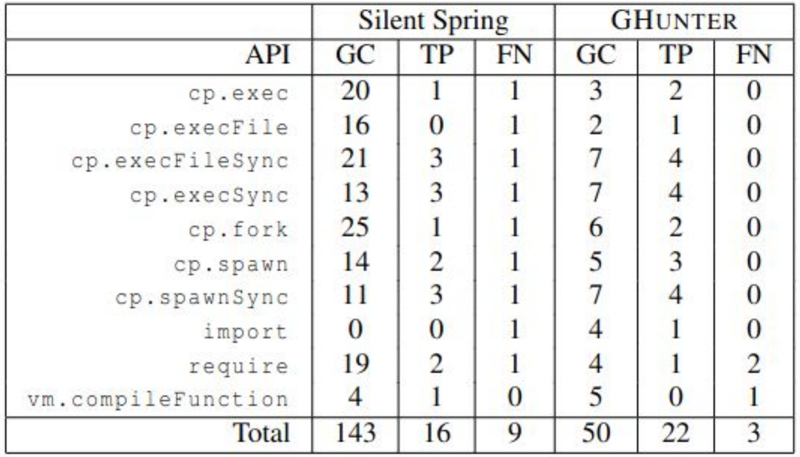
Candidates	**51**
True Positive False Positive	**22** **29**
False Negative	**3**
Precision	**0.43**
Recall	**0.88**

Silent Spring [1]

Candidates	**143**
True Positive False Positive	**16** **127**
False Negative	**9**
Precision	**0.11**
Recall	**0.64**

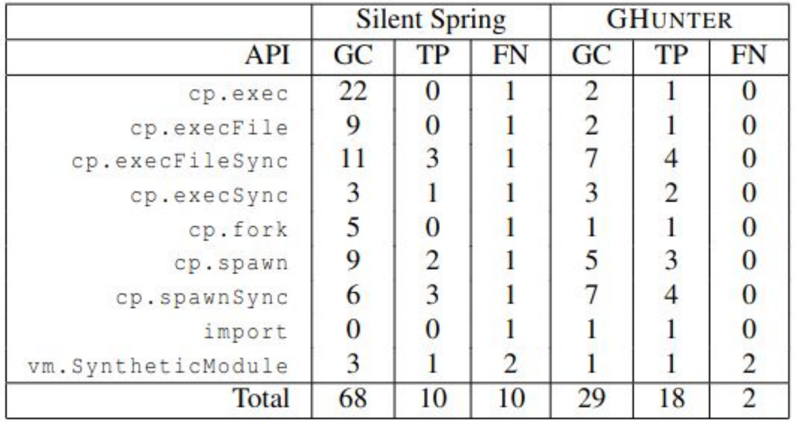
[1]: Shcherbakov, Mikhail, Musard Balliu, and Cristian-Alexandru Staicu. "Silent spring: Prototype pollution leads to remote code execution in Node. js." 32nd USENIX Security Symposium (USENIX Security 23). 2023.

Full Comparison to Silent Spring



	Silent Spring			GHunter		
API	GC	TP	FN	GC	TP	FN
`cp.exec`	20	1	1	3	2	0
`cp.execFile`	16	0	1	2	1	0
`cp.execFileSync`	21	3	1	7	4	0
`cp.execSync`	13	3	1	7	4	0
`cp.fork`	25	1	1	6	2	0
`cp.spawn`	14	2	1	5	3	0
`cp.spawnSync`	11	3	1	7	4	0
`import`	0	0	1	4	1	0
`require`	19	2	1	4	1	2
`vm.compileFunction`	4	1	0	5	0	1
Total	143	16	9	50	22	3

Table 2: Comparison of results from Silent Spring to GHunter on Node.js v16.13.1 using Silent Spring gadgets as ground truth.



	Silent Spring			GHunter		
API	GC	TP	FN	GC	TP	FN
`cp.exec`	22	0	1	2	1	0
`cp.execFile`	9	0	1	2	1	0
`cp.execFileSync`	11	3	1	7	4	0
`cp.execSync`	3	1	1	3	2	0
`cp.fork`	5	0	1	1	1	0
`cp.spawn`	9	2	1	5	3	0
`cp.spawnSync`	6	3	1	7	4	0
`import`	0	0	1	1	1	0
`vm.SyntheticModule`	3	1	2	1	1	2
Total	68	10	10	29	18	2

Table 3: Comparison of results from Silent Spring to GHunter on Node.js v21.0.0 using GHunter ACE gadgets as ground truth.

Mitigations

- **G1: Explicit access to own properties**

  If the code accesses a property in only a few instances, developers should verify each access explicitly.

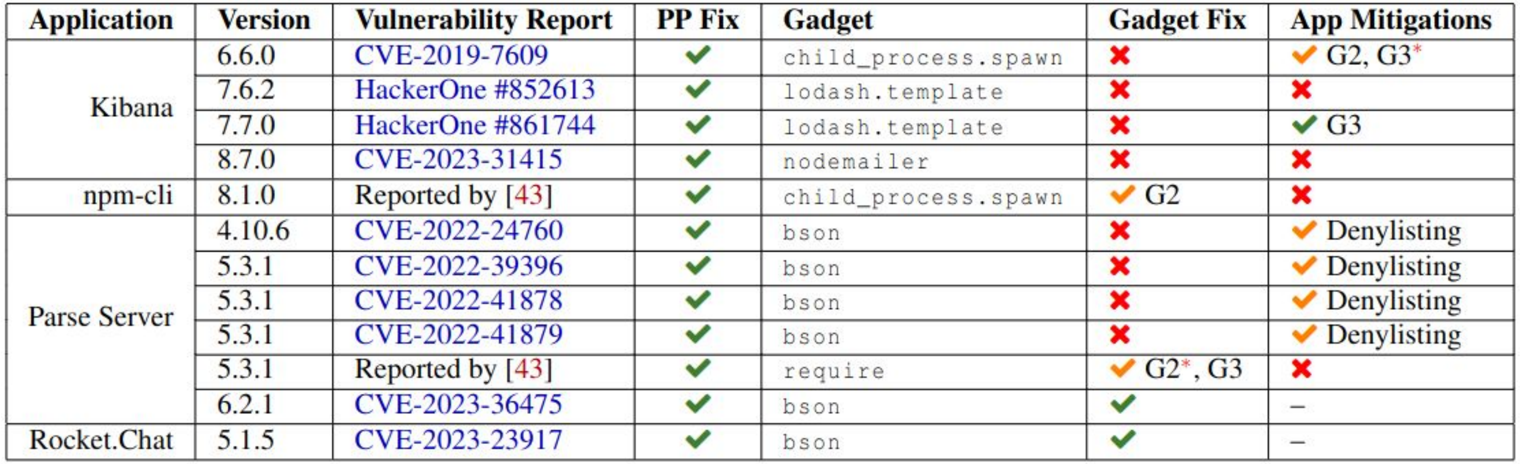
- **G2: Safe object creation**

  When creating an object, developers should use either `null` prototypes or built-in objects `Map` and `Set`.

- **G3: Safe copy of input data**

  Whenever an object is received as input data, developers should copy the object's properties to a safe object.

Mitigations Review



Application	Version	Vulnerability Report	PP Fix	Gadget	Gadget Fix	App Mitigations
Kibana	6.6.0	CVE-2019-7609	✓	`child_process.spawn`	✗	✔ G2, G3*
	7.6.2	HackerOne #852613	✓	`lodash.template`	✗	✗
	7.7.0	HackerOne #861744	✓	`lodash.template`	✗	✓ G3
	8.7.0	CVE-2023-31415	✓	`nodemailer`	✗	✗
npm-cli	8.1.0	Reported by [43]	✓	`child_process.spawn`	✔ G2	✗
Parse Server	4.10.6	CVE-2022-24760	✓	`bson`	✗	✔ Denylisting
	5.3.1	CVE-2022-39396	✓	`bson`	✗	✔ Denylisting
	5.3.1	CVE-2022-41878	✓	`bson`	✗	✔ Denylisting
	5.3.1	CVE-2022-41879	✓	`bson`	✗	✔ Denylisting
	5.3.1	Reported by [43]	✓	`require`	✔ G2*, G3	✗
	6.2.1	CVE-2023-36475	✓	`bson`	✓	–
Rocket.Chat	5.1.5	CVE-2023-23917	✓	`bson`	✓	–

Table 4: A summary of the RCEs exploited via prototype pollution. For each application, we list the vulnerable version, a reference to the report, and the exploited gadget. *PP Fix* shows whether the prototype pollution was fixed; *Gadget Fix* shows whether the gadget was fixed, including any applied guidelines; *App Mitigations* details if mitigations against the attack were implemented in the application. ✗ indicates that no fix has been applied; ✔ indicates that a fix was applied but later bypassed; ✓ indicates that a fix was applied and effectively protects against similar attacks. (*) denotes a guideline that might be bypassed.