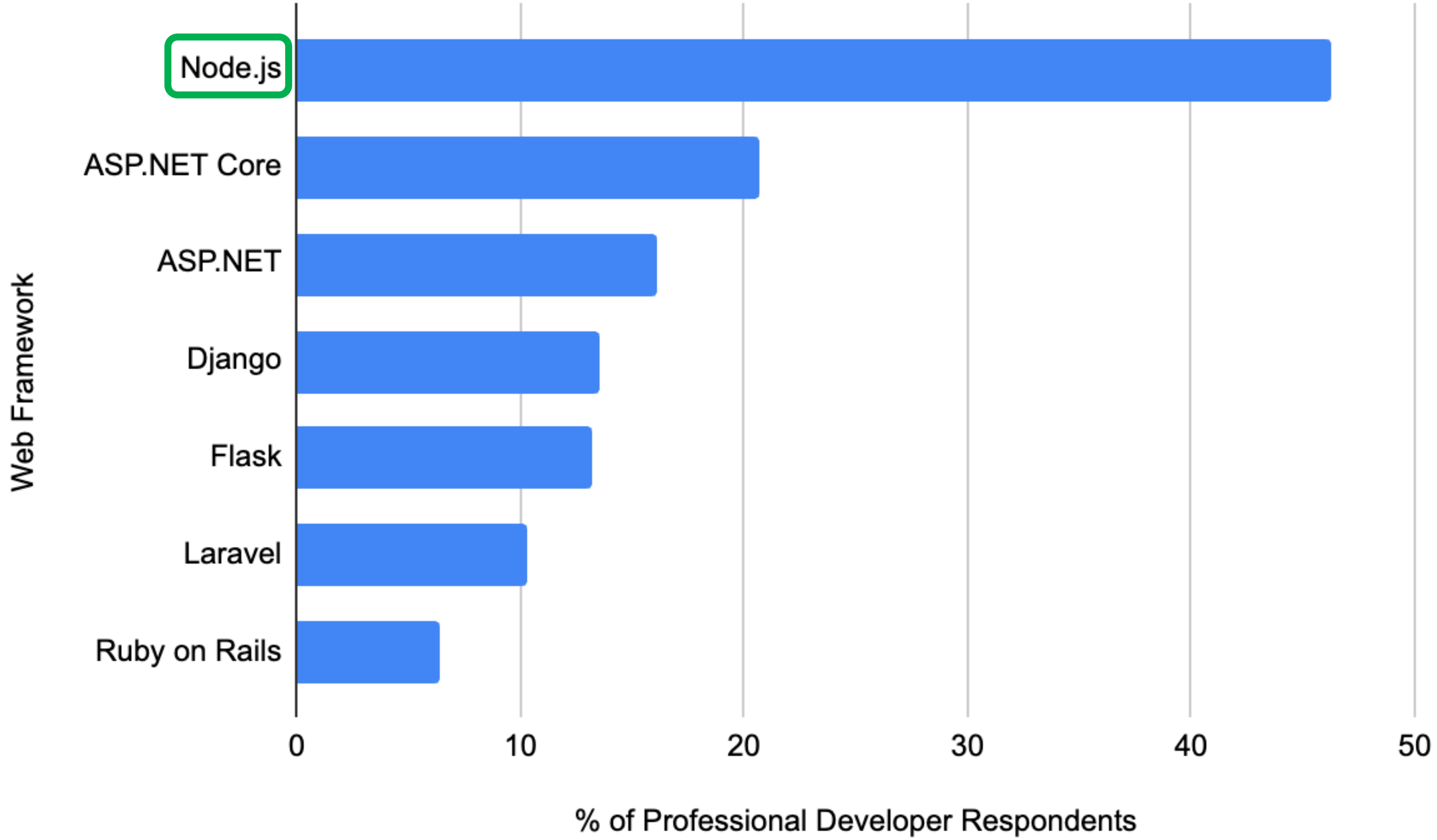
NodeMedic: End-to-End Analysis of Node.js Vulnerabilities with Provenance Graphs

Darion Cassel, Wai Tuck Wong, Limin Jia

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

Introduction: Node.js JavaScript Runtime

Node.js is widely used for server-side, desktop, and IoT development



2022 Developer Survey

npm: Ecosystem of 1 million+ packages developers can use

Node.js is Popular for Attackers Too

Node.js package vulnerabilities in the news

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

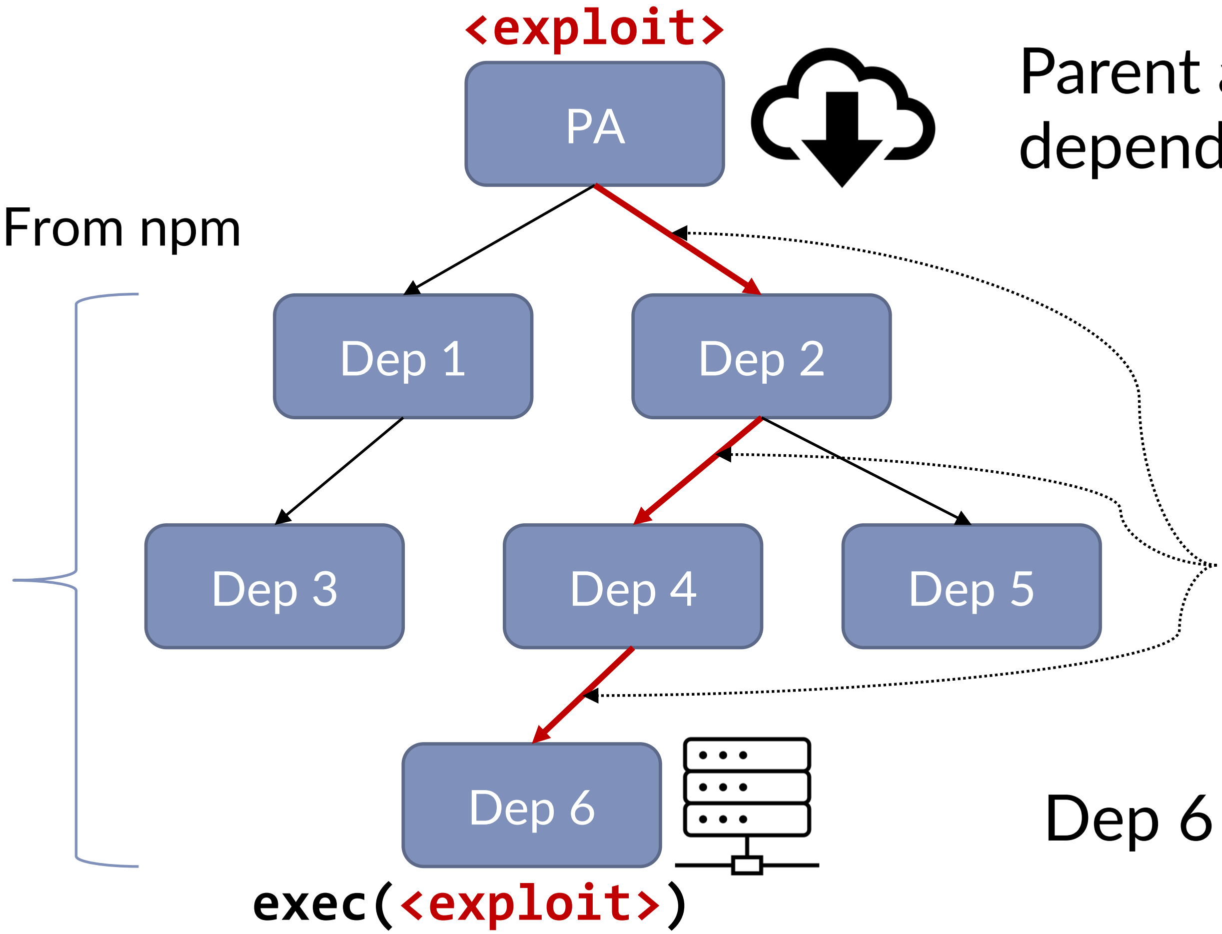
Have you updated your Electron app? We hope so. There was a bad code-injection bug in it

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

Background: Node.js Package Attacker Model



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

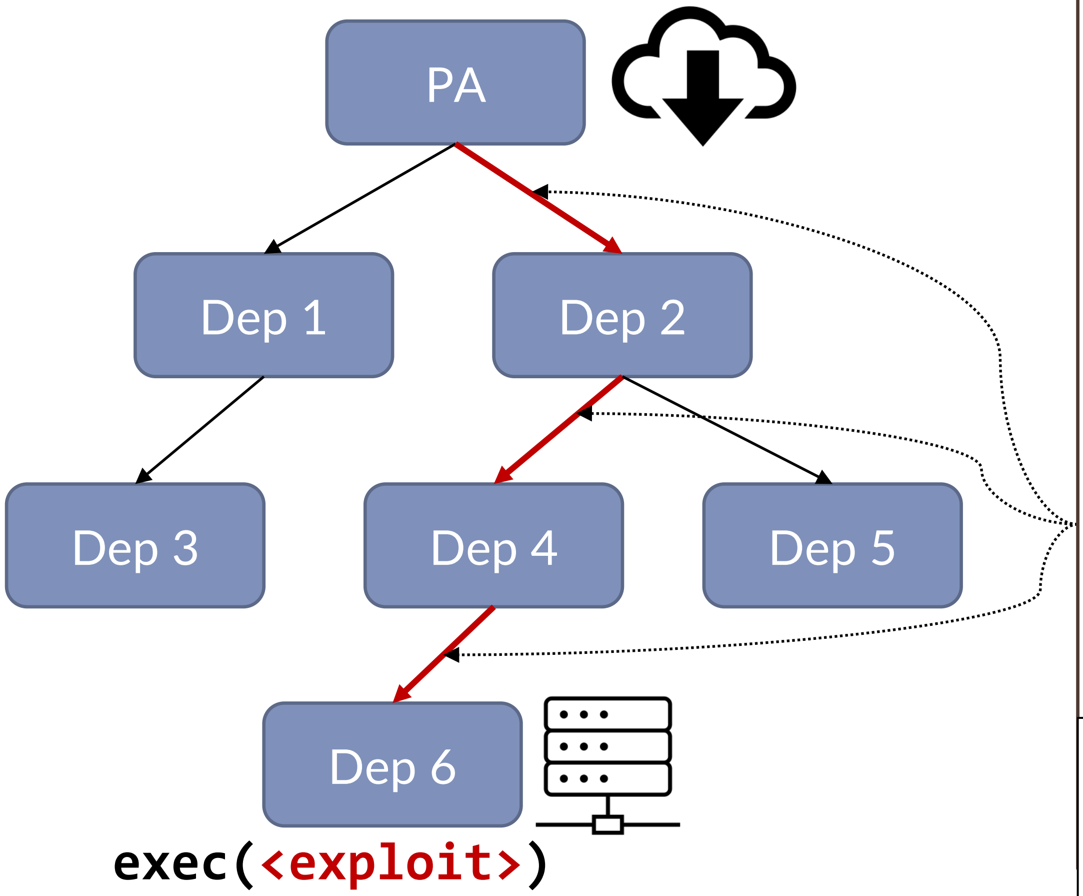
Arbitrary Code Execution (ACE; CWE-94)
Arbitrary Command Injection (ACI; CWE-88)

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

Background: Node.js Package Attacker Model



Prior work detects these flows with **dynamic taint analysis**

[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. *IEEE Transactions on Software Engineering (TSE)*, 2018.

Challenge: Average package has **79 dependencies** to be checked [Zimmerman 2019]

Challenges for Node.js Package Dynamic Taint Analysis

1. Driving package APIs
2. Precise analysis of built-in datatypes
3. **Scaling to large dependency trees**

End-to-End Analysis Infrastructure

4. Triage of tainted flows
5. **Confirmation of tainted flows**

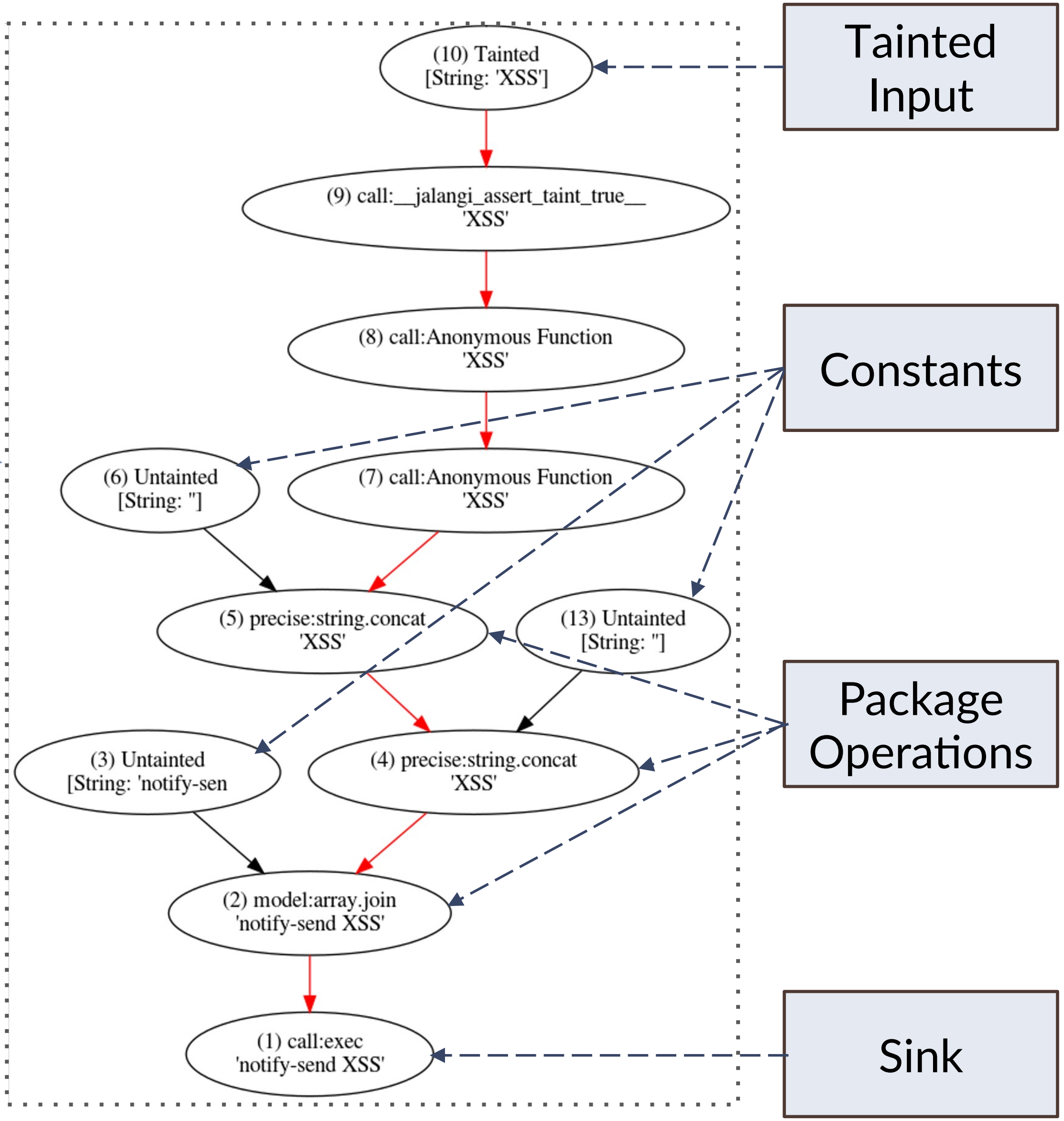
Provenance Graphs

Augmenting Taint Analysis with Provenance Graphs

Provenance Tracking

- Prior: Policy-based taint propagation
- **+ Graph** of operations performed

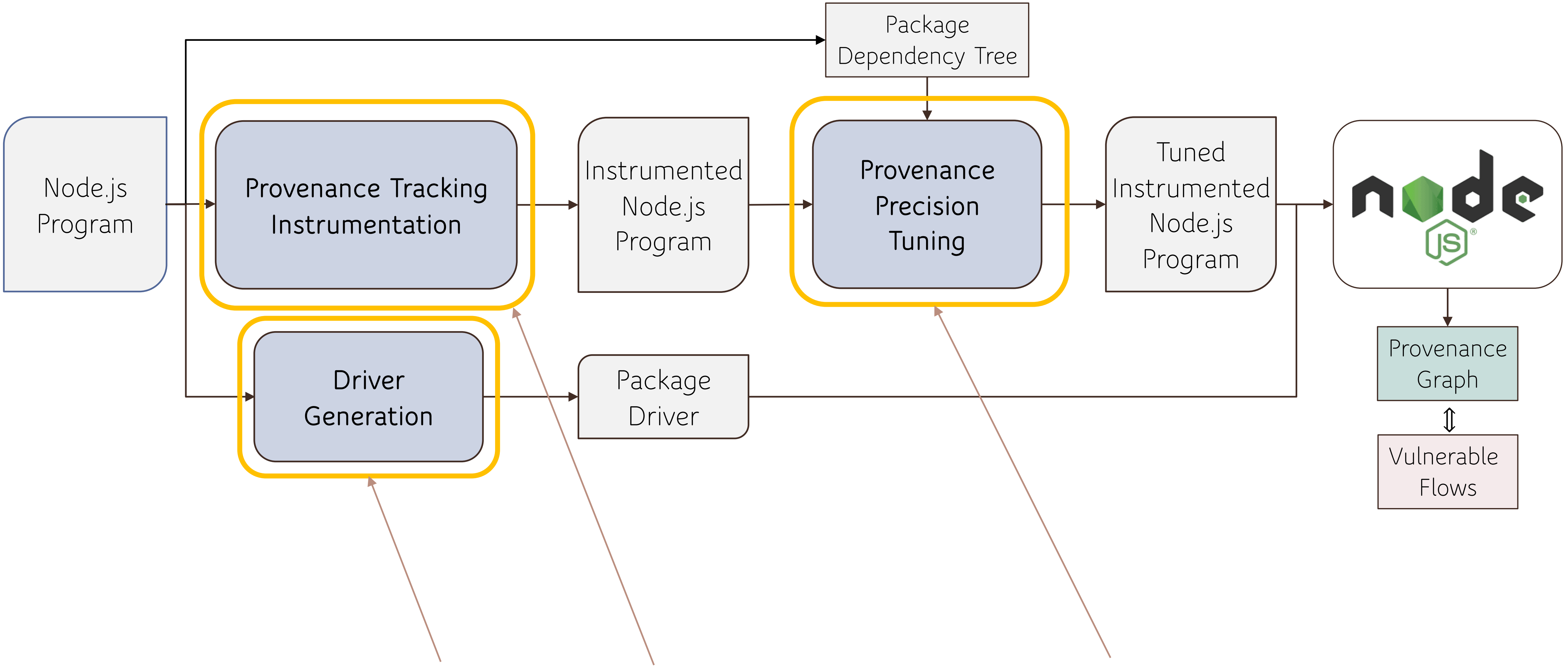
Provenance graph
output of NodeMedic



Foundation for **further analysis**:

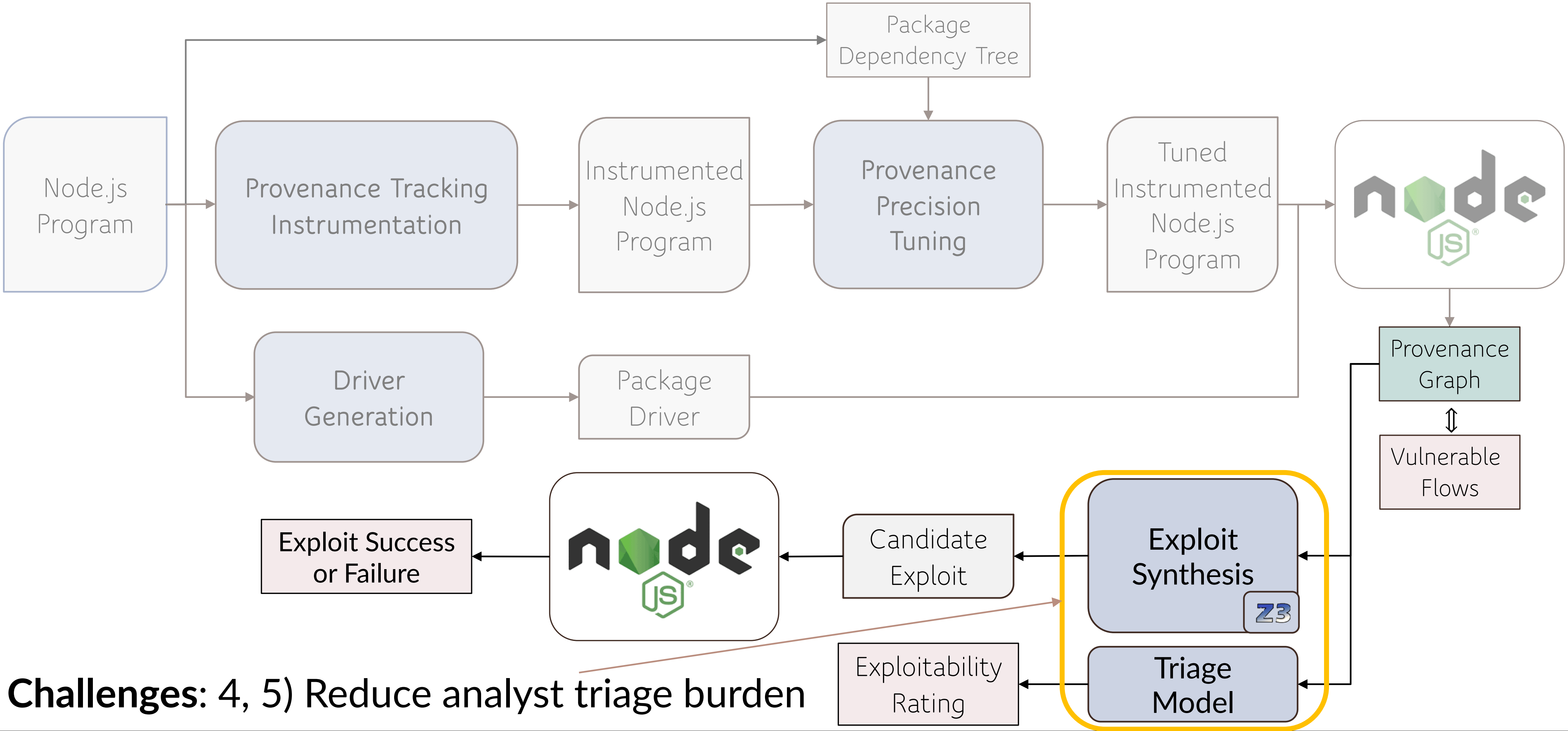
- Exploit synthesis (*covered later*)
- Triage (*see paper*)

NodeMedic End-to-End Analysis Infrastructure (1/2)



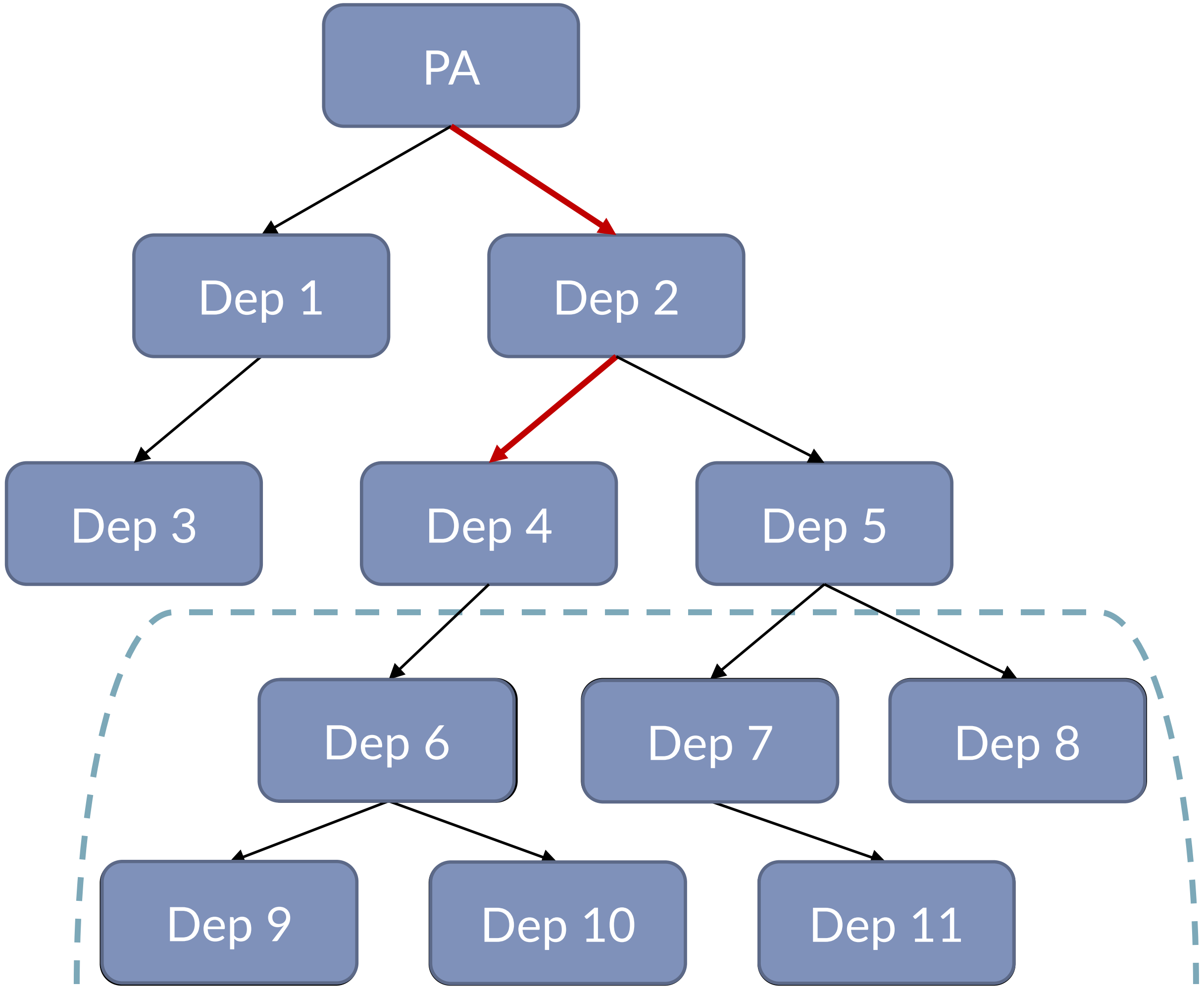
Challenges: 1) Driving package APIs 2) Precise analysis 3) Scalable analysis

NodeMedic End-to-End Analysis Infrastructure (2/2)



Challenges: 4, 5) Reduce analyst triage burden

Solution: Scalable Analysis of Large Dependency Trees



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

Solution: Reducing Analyst Triage Burden (1/2)

Motivation: Analyst must manually *confirm* reported tainted flows

- *Confirm*: Construct a proof-of-concept (PoC) exploit
- Reduces analysis scalability

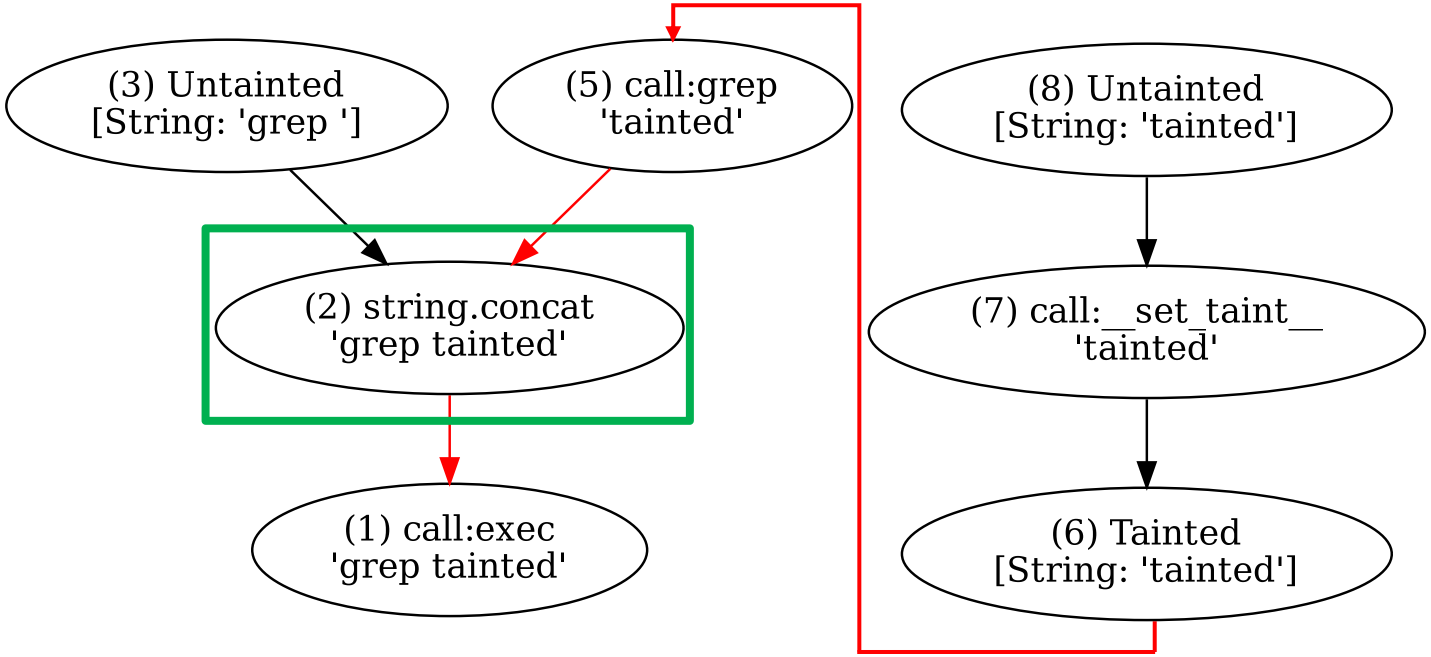
Insight: Provenance graph contains operations performed on tainted value

Package API

```
function grep(inpt) {
    exec('grep ' + inpt);
}
```

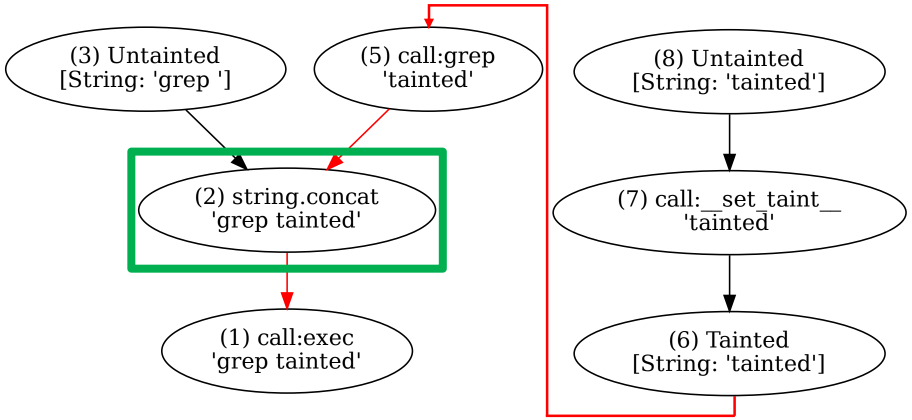
Driver Code

```
__set_taint__('tainted');
grep('tainted');
```



Solution: Reducing Analyst Triage Burden (2/2)

① Provenance graph → SMT formula encoding operations and PoC




```
(declare-const i0 String)
(assert (str.contains
      (str.++
          “grep ” i0
      )
     “ $(touch success);#”
))
```

② Solve with Z3 and derive model if SAT

```
1 (i0 “ $(touch success);#”)
```

③ Rerun package with candidate PoC

```
grep(“ $(touch success);#”);
```

④ Check for PoC success

```
success
```

Results: Large-Scale Evaluation on Real Node.js Packages

Result: Scalable analysis of 10,000 packages from npm

Prior work: ~20 packages [1, 2]

Package Results	Count
Inherent package issues	394
Package analysis timeout	258
No tainted flows	9175
Tainted flows	173

Result: Able to automatically confirm 108 potential flows

Type	Count	Confirmed	Percent
Arbitrary command injection (ACI)	133	102	76%
Arbitrary code execution (ACE)	22	6	27%
Total	155	108	70%

More in the Paper and our Repository

→ In the paper:

- Precise provenance analysis
- Custom propagation policies
- Triage rating methodology

→ github.com/NodeMedicAnalysis

- End-to-end infrastructure
- 589 taint precision tests
- Case studies