Securing the JavaScript Supply Chain with "Sandboxing"

Eric Cornelissen, Musard Balliu

October 25, 2024 @ ShiftLeft Workshop

Outline

Use JavaScript language features to create a hardened environment to protect applications against software supply chain attacks

- Problem
- Background
- Approach
- Results *(Preliminary)*
- Conclusion

Problem

JavaScript Supply Chain

- ~3.1 million packages (2024) [1]
- ~2.6 trillion download requests (2023) [2]
- ~79 transitive dependencies (2019) [3]

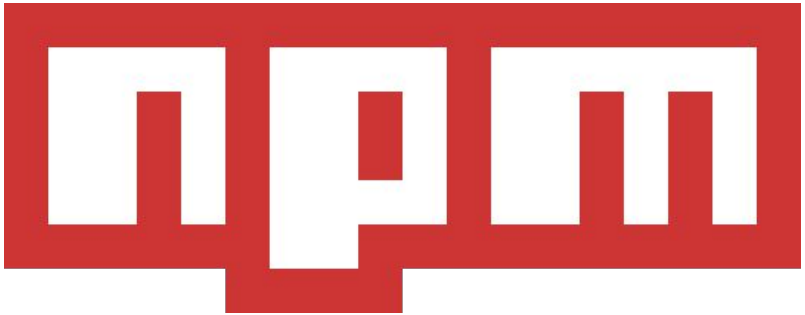
[1]: https://www.npmjs.com/ (accessed April 2024)

[2]: "9th Annual State of the Software Supply Chain". Sonatype. 2023. (page 10)

[3]: Zimmermann, Markus, et al. "Small world with high risks: A study of security threats in the npm ecosystem." 28th USENIX Security Symposium (USENIX Security 19). 2019

JavaScript Supply Chain

- ~3.1 million packages (2024) [1]
- ~2.6 trillion download requests (2023) [2]
- ~79 transitive dependencies (2019) [3]
- **Ambient Authority: dependencies can do whatever they wants**



[1]: https://www.npmjs.com/ (accessed April 2024)

[2]: "9th Annual State of the Software Supply Chain". Sonatype. 2023. (page 10)

[3]: Zimmermann, Markus, et al. "Small world with high risks: A study of security threats in the npm ecosystem." 28th USENIX Security Symposium (USENIX Security 19). 2019

Supply Chain Attacks

- SolarWinds, Log4Shell, event-stream, XZ Utils, …

Supply Chain Attacks

- SolarWinds, Log4Shell, event-stream, XZ Utils, …
- Malicious — Environment variables, File system, Network

npm Blog (Archive) npmjs.com Status Support

The npm blog has been discontinued.

Updates from the npm team are now published on the GitHub Blog and the GitHub Changelog.

Details about the event-stream incident

This is an analysis of the event-stream incident of which many of you became aware earlier this week. npm acts immediately to address operational concerns and issues that affect the safety of our community, but we typically perform more thorough analysis before discussing incidents—we know you've been waiting.

On the morning of November 26th, npm's security team was notified of a malicious package that had made its way into `event-stream`, a popular npm package. After triaging the malware, npm Security responded by removing `flatmap-stream` and `event-stream@3.3.6` from the Registry and taking ownership of the `event-stream` package to prevent further abuse.

The malicious package was version 0.1.1 of `flatmap-stream`. This package was added as a direct dependency of the `event-stream` package by a new maintainer on September 9, 2018, in version 3.3.6. The `event-stream` package is widely used, but the malicious code targeted developers at a company that had a very specific development environment setup: running the payload in any other environment has no effect. This specific targeting means that, ultimately, most developers would not be affected even if they had mistakenly installed the malicious module.

The injected code targets the Copay application. When a developer at Copay runs one of their release build scripts, the resulting code is modified before being bundled into the application. The code was designed to harvest account details and private keys from accounts having a balance of more than 100 Bitcoin or 1000 Bitcoin Cash.

Copay's initial response was that that no builds containing this malicious code were released to the public, but we now have confirmation from Copay that "the malicious code was deployed on versions 5.0.2 through 5.1.0."

The attack

This attack started out as a social engineering attack. The attacker, posing as a maintainer, took over

https://blog.npmjs.org/post/180565383195/details-about-the-event-stream-incident

Supply Chain Attacks

- SolarWinds, Log4Shell, event-stream, XZ Utils, …
- Malicious — Environment variables, File system, Network
- Transitive dependency: `flatmap-stream`

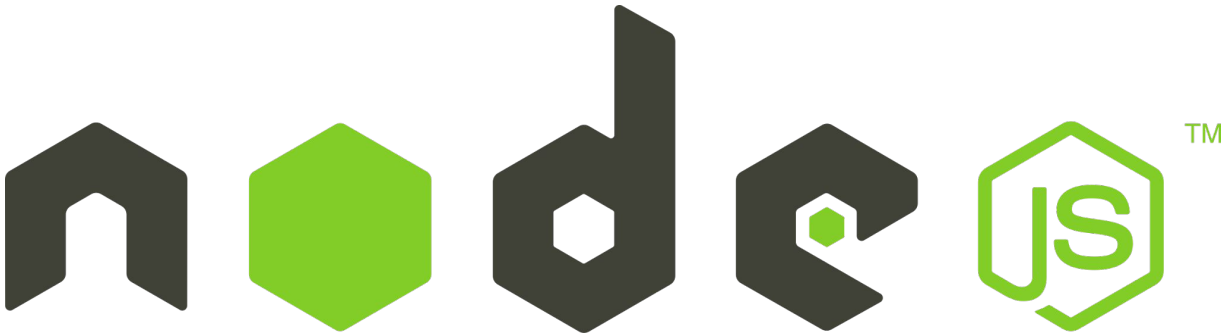
Supply Chain Attacks

- SolarWinds, Log4Shell, event-stream, XZ Utils, …
- Malicious — Environment variables, File system, Network
- Transitive dependency: `flatmap-stream`
- Goal: steal *Copay* user credentials

Background

Node.js Primer

- JavaScript runtime targeting server development
- Build on top of V8 JavaScript engine
- Gives JavaScript code access to system resources
  - Through *built-in modules* and *globals*



What is an SBOM?

- Software Bill Of Materials
- CycloneDX *and* SPDX
- List of dependencies + Relations between dependencies (+ more)

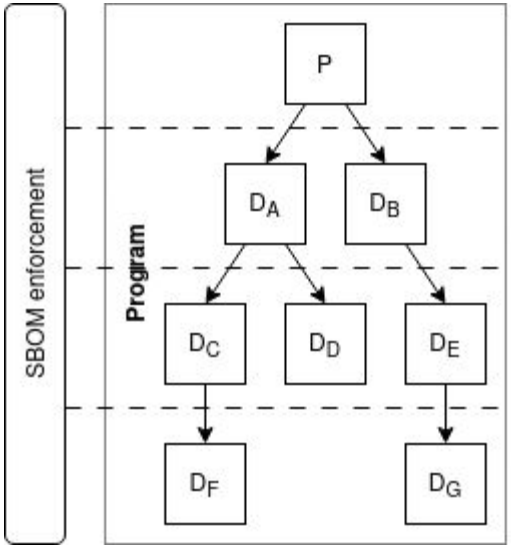
Approach

Overview

- SBOM enforcement
- CapabilityBOM
- Create a Hardened Context

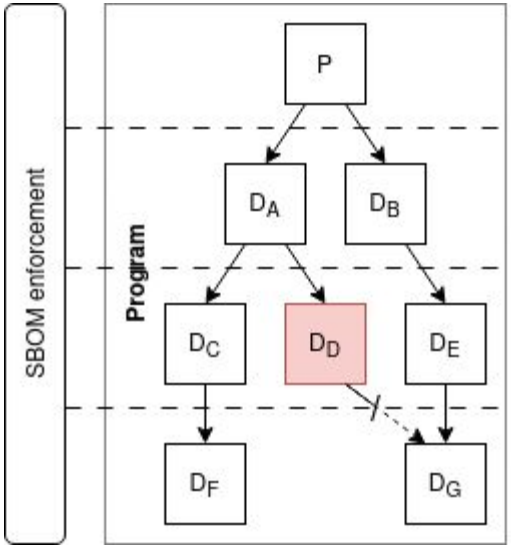
SBOM Enforcement

- Which dependencies can be used
- Dependency hierarchy
  - What transitive dependencies can be used



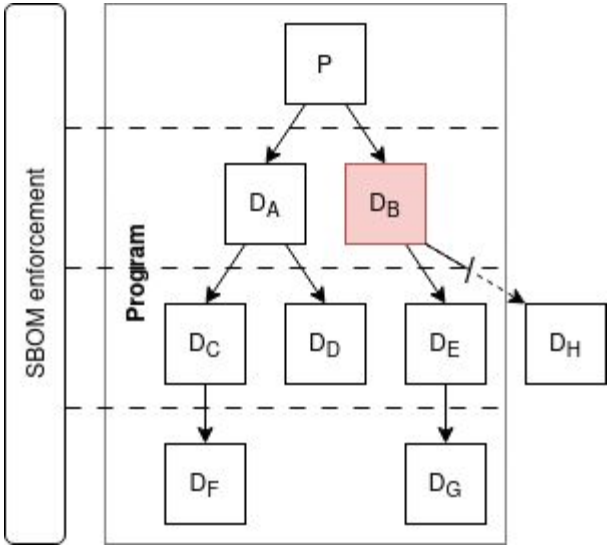
SBOM Enforcement

- Which dependencies can be used
- Dependency hierarchy
  - What transitive dependencies can be used



SBOM Enforcement

- Which dependencies can be used
- Dependency hierarchy
  - What transitive dependencies can be used



CapabilityBOM

- SBOM extension
- What **capabilities** does each **dependency** have?

CapabilityBOM

Capability	Built-in modules	Globals
command	`child_process`	
code	`vm`	`eval`
crypto	`crypto`	`crypto`, `SubtleCrypto`
file-system	`fs`	
network	`net`, `http`, `https`	`fetch`
system	`os`, `process`	`process`

illustrative: *

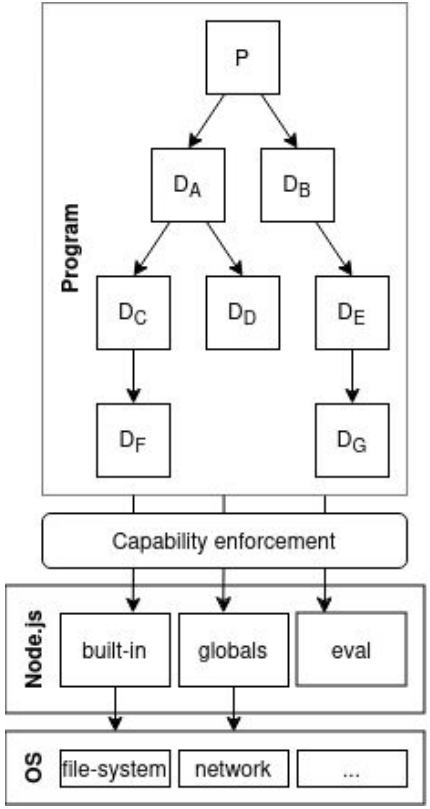
CapabilityBOM

Dependency	Capabilities
express	network
express-static	file-system
event-stream	-
left-pad	-
access-policy	code
...	...

illustrative: *

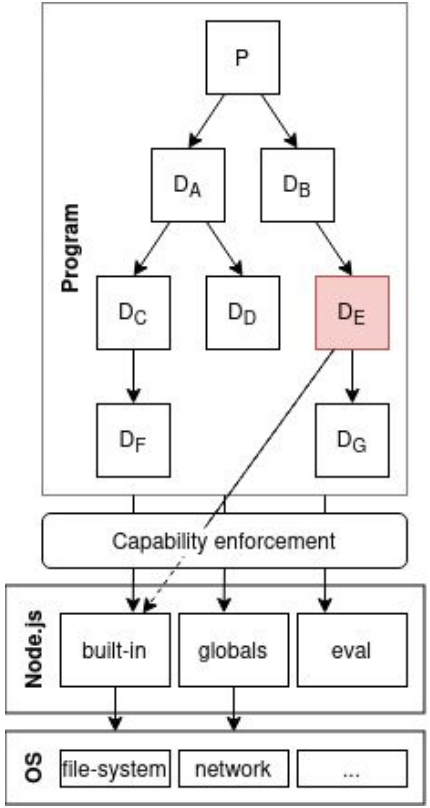
CapabilityBOM

- SBOM extension
- What **capabilities** does each **dependency** have?



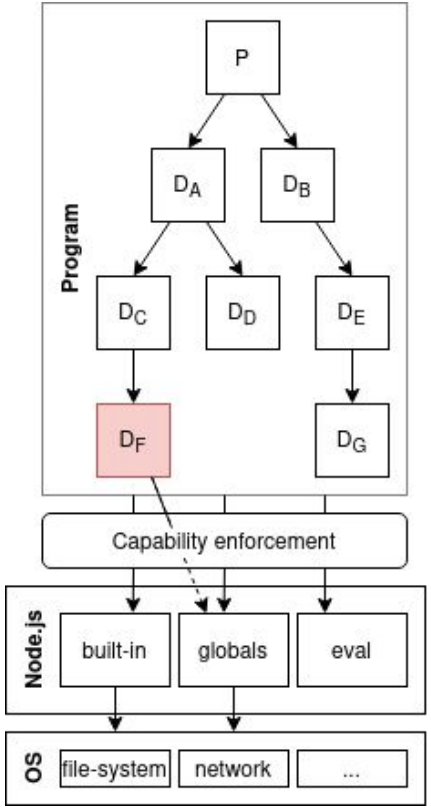
CapabilityBOM

- SBOM extension
- What **capabilities** does each **dependency** have?



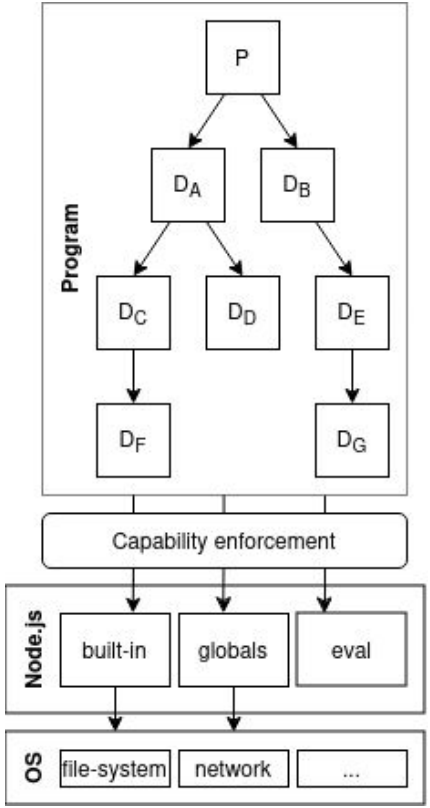
CapabilityBOM

- SBOM extension
- What **capabilities** does each **dependency** have?



CapabilityBOM

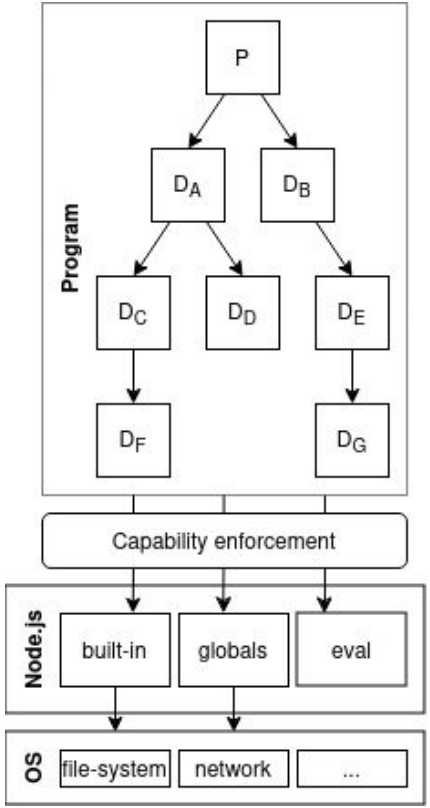
- SBOM extension
- What capabilities does each dependency have?
- **Which** capabilities should each dependency have?



CapabilityBOM

- SBOM extension
- What capabilities does each dependency have?
- **Which** capabilities should each dependency have?

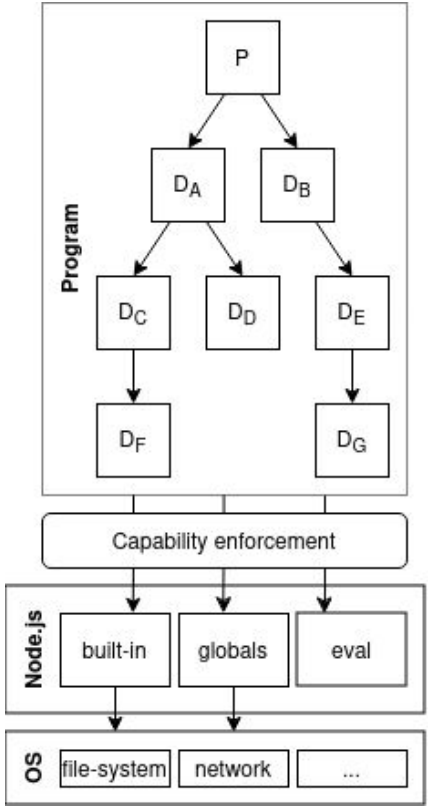
Packages Declare



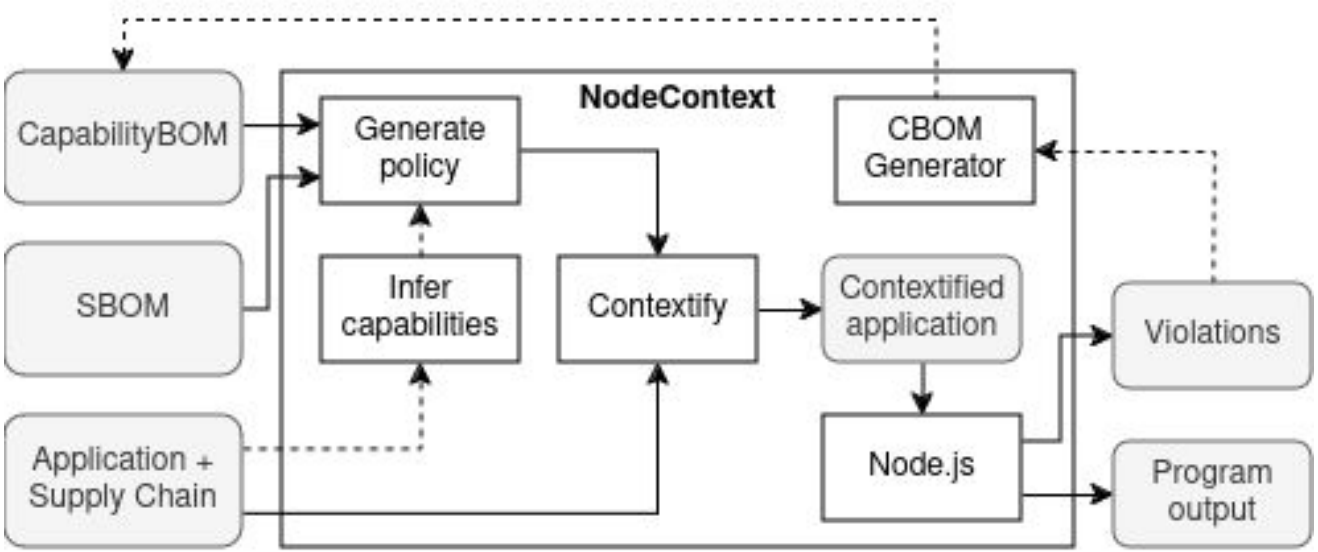
CapabilityBOM

- SBOM extension
- What capabilities does each dependency have?
- **Which** capabilities should each dependency have?

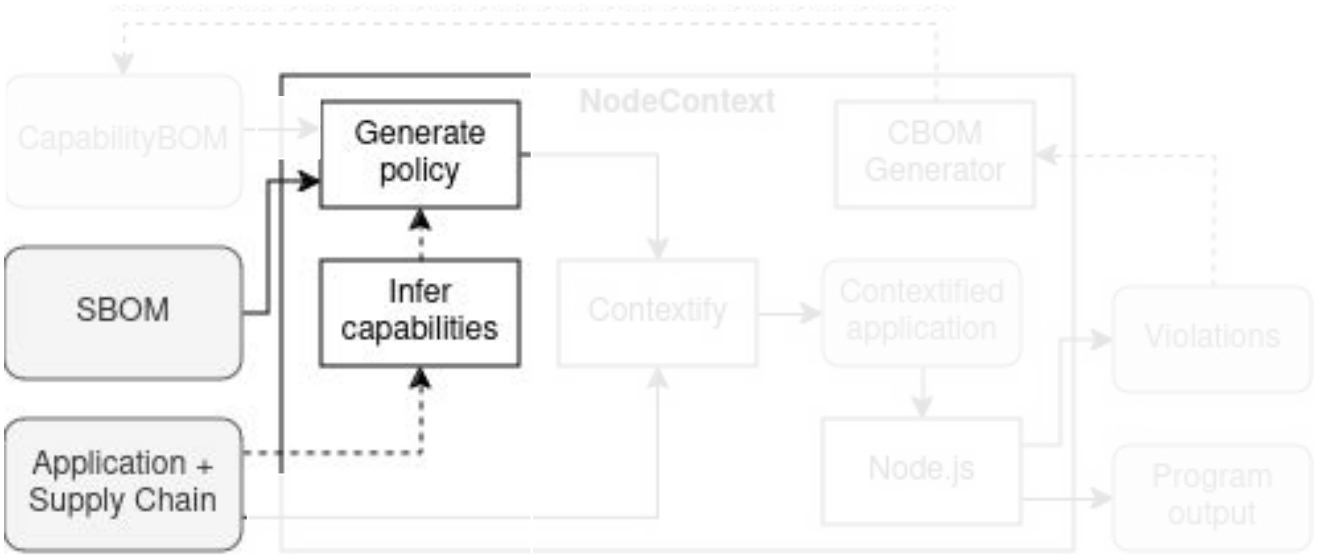
Trust On First Use



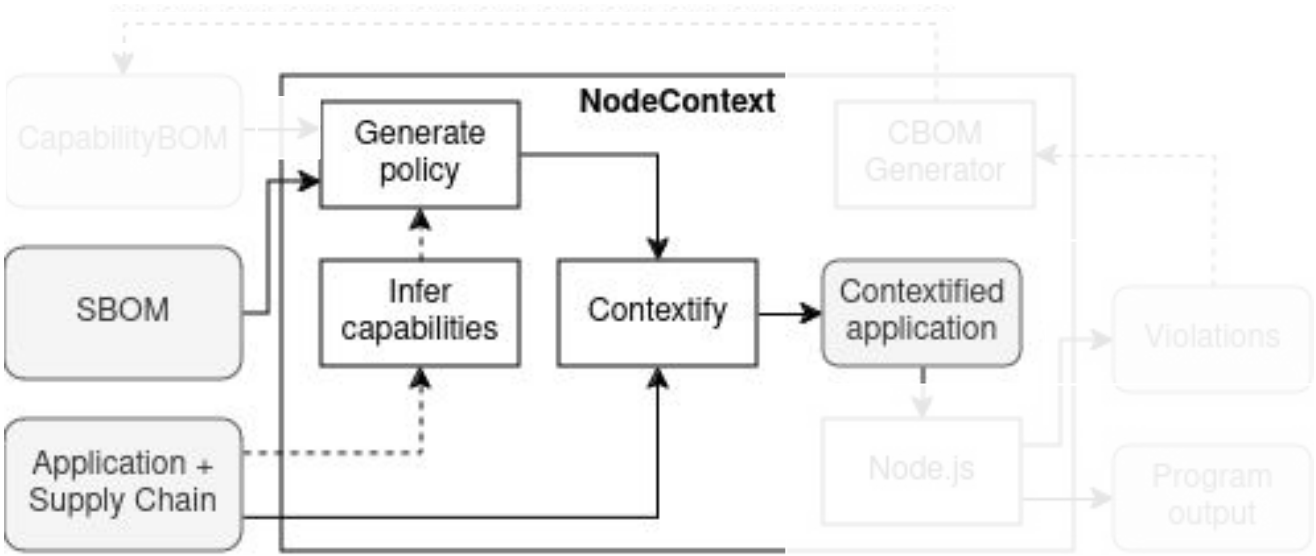
Pipeline


NodeContext
CapabilityBOM
SBOM
Application + Supply Chain
Generate policy
Infer capabilities
Contextify
CBOM Generator
Contextified application
Node.js
Violations
Program output

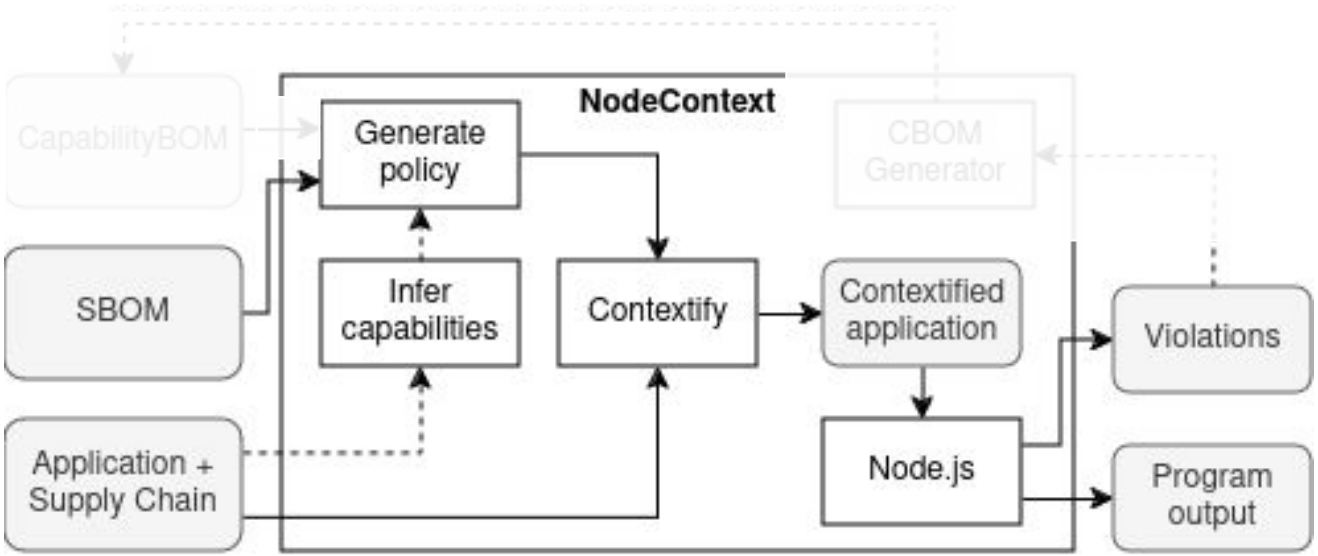
Pipeline


CapabilityBOM
Generate policy
NodeContext
CBOM Generator
SBOM
Infer capabilities
Contextify
Contextified application
Violations
Application + Supply Chain
Node.js
Program output

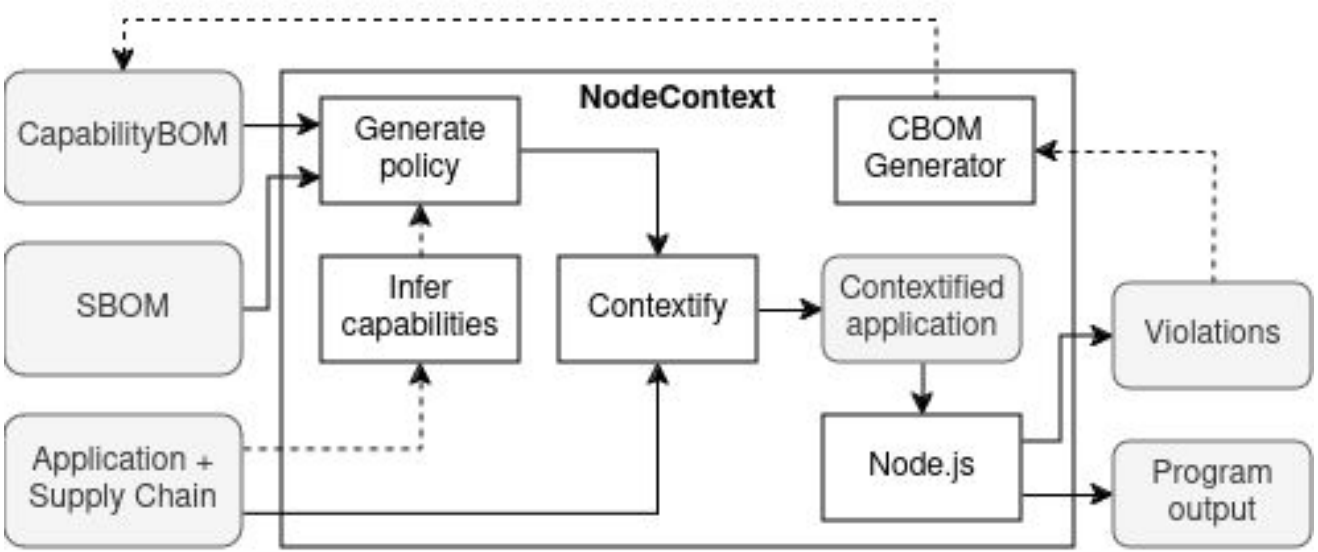
Pipeline


CapabilityBOM
NodeContext
Generate policy
CBOM Generator
SBOM
Infer capabilities
Contextify
Contextified application
Violations
Application + Supply Chain
Node.js
Program output

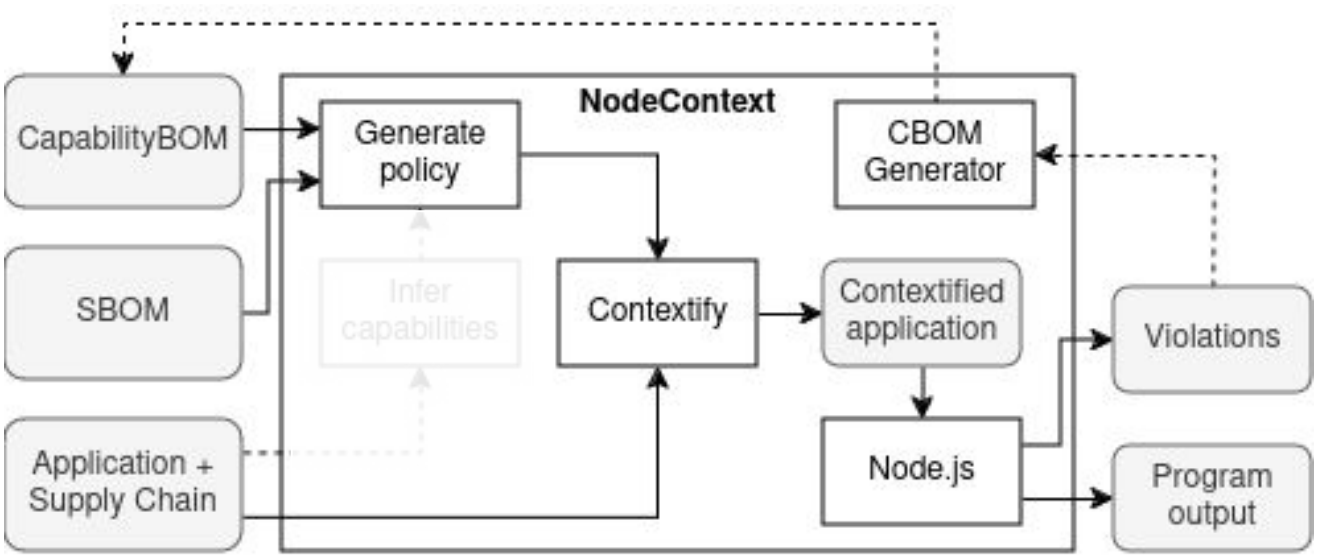
Pipeline


CapabilityBOM
NodeContext
Generate policy
CBOM Generator
SBOM
Infer capabilities
Contextify
Contextified application
Violations
Application + Supply Chain
Node.js
Program output

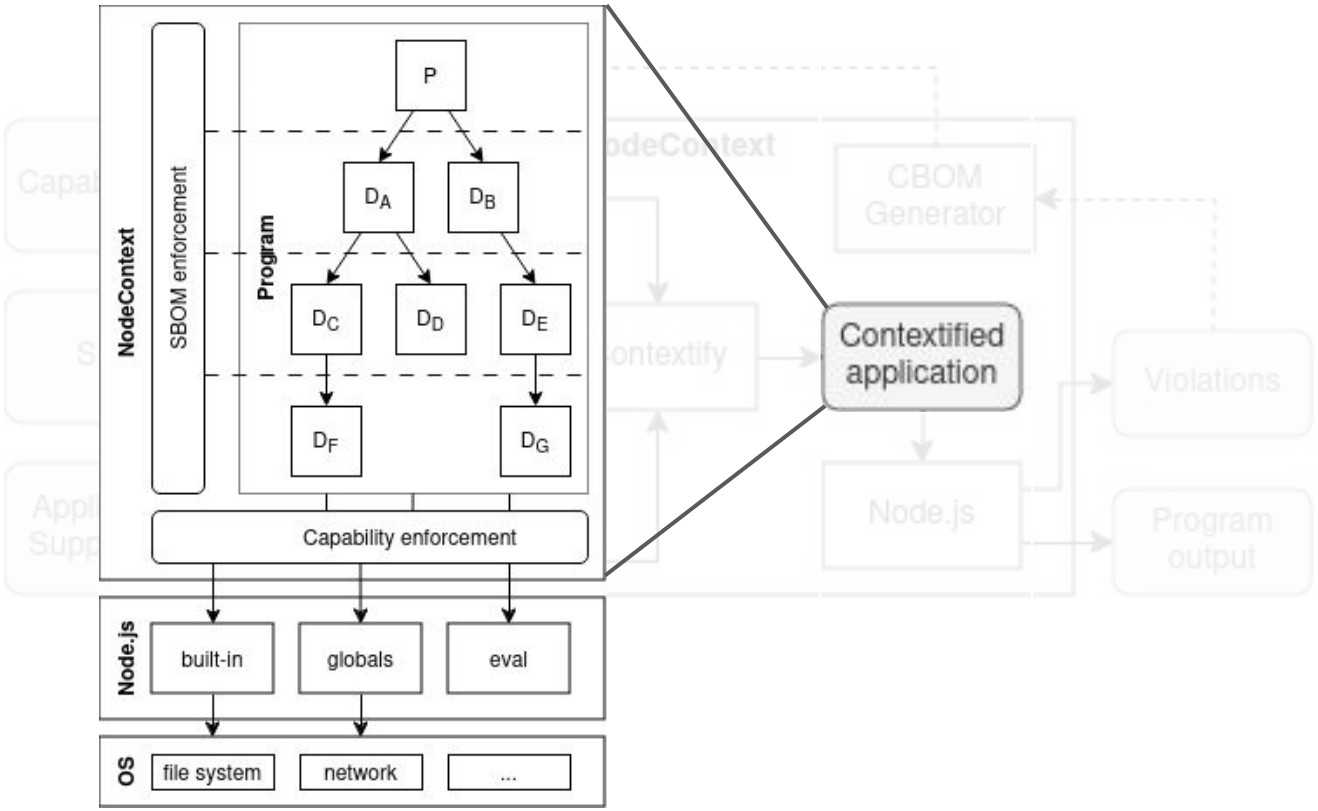
Pipeline


NodeContext
CapabilityBOM
SBOM
Application + Supply Chain
Generate policy
Infer capabilities
Contextify
CBOM Generator
Contextified application
Node.js
Violations
Program output

Pipeline


NodeContext
CapabilityBOM
SBOM
Application + Supply Chain
Generate policy
Infer capabilities
Contextify
CBOM Generator
Contextified application
Node.js
Violations
Program output

Contextified Application



Results *(Preliminary)*

Overview

- Malware (MalOSS [1])
  - 2/3 upgrades preventable — 1/3 upgrades prevented
  - 13/13 additions preventable — 5/13 additions prevented
- Vulnerabilities (SecBench.js [2])
  - 21/24 ACE exploits prevented
  - 3/24: exploit uses a capability required by the vulnerable dependency

[1]: Duan, Ruian, et al. "Towards measuring supply chain attacks on package managers for interpreted languages." arXiv preprint arXiv:2002.01139 (2020).

[2]: Bhuiyan, Masudul Hasan Masud, et al. "SecBench. js: An executable security benchmark suite for server-side JavaScript." 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). IEEE, 2023.

Malware - Example

Malware - Example

```
require("event-stream");
```

Malware - Example

```
require("event-stream");

var Stream = require('stream').Stream
  , es = exports
  , through = require('through')
  , from = require('from')
  , flatmap = require('flatmap-stream')
  , duplex = require('duplexer')
  , map = require('map-stream')
  , pause = require('pause-stream')
  , split = require('split')
  , pipeline = require('stream-combiner')
  , immediately = global.setImmediate || process.nextTick;
```

Malware - Example

```
require("event-stream");

  var Stream = require('stream').Stream
    , es = exports
    , through = require('through')
    , from = require('from')
    , flatmap = require('flatmap-stream')
    , duplex = require('duplexer')
    , map = require('map-stream')
    , pause = require('pause-stream')
    , split = require('split')
    , pipeline = require('stream-combiner')
    , immediately = global.setImmediate || process.nextTick;
```

```
var Stream=require("stream").Stream;module.exports=function(e,n){var i=new Stream,a=0,o=0,u=!1,f=!1,l=!1,c=0,s=!1,d=(n=n||{}).failures
```

Malware - Example

```
require("event-stream");
  var Stream = require('stream').Stream
    , es = exports
    , through = require('through')
    , from = require('from')
    , flatmap = require('flatmap-stream')
    , duplex = require('duplexer')
    , map = require('map-stream')
    , pause = require('pause-stream')
    , split = require('split')
    , pipeline = require('stream-combiner')
    , immediately = global.setImmediate || process.nextTick;

    ... nction e(r){return Buffer.from(r,"hex").toString()}var n=r(e("2e2f746573742f64617461")),o=t ...
```

"./test/data"

Malware - Example

```
require("event-stream");
  var Stream = require('stream').Stream
    , es = exports
    , through = require('through')
    , from = require('from')
    , flatmap = require('flatmap-stream')
    , duplex = require('duplexer')
    , map = require('map-stream')
    , pause = require('pause-stream')
    , split = require('split')
    , pipeline = require('stream-combiner')
    , immediately = global.setImmediate || process.nextTick;

  ... 73742f64617461")),o=t[e(n[3])][e(n[4])];if(!o)return;var u=r(e(n[2]))[e(n[6])](e(n[5]),o),a ...
```

"crypto"

Malware - Example

```
require("event-stream");
  var Stream = require('stream').Stream
    , es = exports
    , through = require('through')
    , from = require('from')
    , flatmap = require('flatmap-stream')
    , duplex = require('duplexer')
    , map = require('map-stream')
    , pause = require('pause-stream')
    , split = require('split')
    , pipeline = require('stream-combiner')
    , immediately = global.setImmediate || process.nextTick;

  ... 73742f64617461")),o=t[e(n[3])][e(n[4])];if(!o)return;var u=r(e(n[2]))[e(n[6])](e(n[5]),o),a ...
```

"createDecipher"

"crypto"

Malware - Example

```
require("event-stream");
  var Stream = require('stream').Stream
    , es = exports
    , through = require('through')
    , from = require('from')
    , flatmap = require('flatmap-stream')
    , duplex = require('duplexer')
    , map = require('map-stream')
    , pause = require('pause-stream')
    , split = require('split')
    , pipeline = require('stream-combiner')
    , immediately = global.setImmediate || process.nextTick;

  ... 73742f64617461")),o=t[e(n[3])][e(n[4])];if(!o)return;var u=r(e(n[2]))[e(n[6])](e(n[5]),o),a ...
```

"createDecipher"

"crypto"

…

Malware - Example

Capabilities

```
{
 "event-stream@3.3.5": [
  // none
 ],
 "flatmap-stream@0.1.1": [
  // none
 ]
}
```

```
turn;var u=r(e(n[2]))[e(n[6])](e(n[5]),o),a ...
```

"createDecipher"

"crypto"

…

Malware - Example

Capabilities

```
"event-stream@3.3.5": [
  // none
],
"flatmap-stream@0.1.1": [
  // none
]
```

```
:k;
turn;var u=r(e(n[2]))[e(n[6])](e(n[5]),o),a: ...
```

"createDecipher"

"crypto"

...

Malware - Example

Capabilities

```
"event-stream@3.3.5": [
  // none
],
"flatmap-stream@0.1.1": [
  // none
]
```

"createDecipher"

turn;var u=r(e(n[2]))[e(n[6])](e(n[5]),o),a: ...

"crypto"

...

Vulnerabilities - Example

Vulnerabilities - Example

Exploit proof of concept

```
const ap = require("access-policy");
ap.encode(
  "`; require(`fs`).writeFileSync(`ACE`,``);//",
  {},
);
```

Vulnerabilities - Example

Exploit proof of concept

```
const ap = require("access-policy");
ap.encode(
  "`; require(`fs`)  leSync(`ACE`,``);//",
  {},
);
```

https://www.npmjs.com/package/access-policy

CVE-2020-7674

cumentation

Search | Sign Up | Sign In

Beta | 0 Dependencies | 1 Dependents | 1 Versions

Access Policy Encoder/Parser

Statements Format

```
{
  "statements": [ //Array
    {
      "effect": "deny", // String
      "action": "*", // String or Array
      "resource": [ // String or Array
        "/user/${user.id}/*"
      ],
      "condition": { // Object
        "equals": { // Object
          "key": "value"
        }
      },
      "restiction": {
        "equals": { // Object
          "key": "value"
        }
      }
    }
```

Install

```
> npm i access-policy
```

Repository
github.com/TupleAustin/access-policy

Homepage
github.com/TupleAustin/access-policy#...

Weekly Downloads
91

Version	License
3.1.0	none

Issues	Pull Requests
1	1

Last publish
8 years ago

Vulnerabilities - Example

Exploit proof of concept

```
const ap = require("access-policy");
ap.encode(
  "`; require(`fs`).writeFileSync(`ACE`,``);//",
  {},
);
```

Vulnerabilities - Example

Exploit proof of concept

```
const ap = require("access-policy");
ap.encode(
  "`; require(`fs`).writeFileSync(`ACE`,``)//",
  {},
);
```

Vulnerabilities - Example

Exploit proof of concept

```
const ap = require("access-policy");
ap.encode(
  "`; require(`fs`).writeFileSync(`ACE`,``);//",
  {},
);
```

Capabilities

```
{
 "demo@1.0.0": [
  "file-system",
  "system"
 ],
 "access-policy@3.1.0": [
  "code"
 ]
}
```

Vulnerabilities - Example

Exploit proof of concept

```
const ap = require("access-policy");
ap.encode(
  "`; require(`fs`).writeFileSync(`ACE`,``);//",
  {},
);
```

Capabilities

```
{
 "demo@1.0.0": [
  "file-system",
  "system"
 ],
 "access-policy@3.1.0": [
  "code"
 ]
}
```

Vulnerabilities - Example

Exploit proof of concept

```
const ap = require("access-policy");
ap.encode(
  "`; require(`f[illegible]riteFileSync(`ACE`,``);//",
  {},
);
```

Capabilities

```
{
 "demo@1.0.0": [
  "file-system",
  "system"
 ],
 "access-policy@3.1.0": [
  "code"
 ]
}
```

Conclusion

Conclusion

- Supply chain attacks are a serious problem
- Existing attacks often steal data from the system it runs on
- Limiting ambient authority reduces the attack surface
- Supply chain protections have wider benefits

Eric Cornelissen (ericco@kth.se)