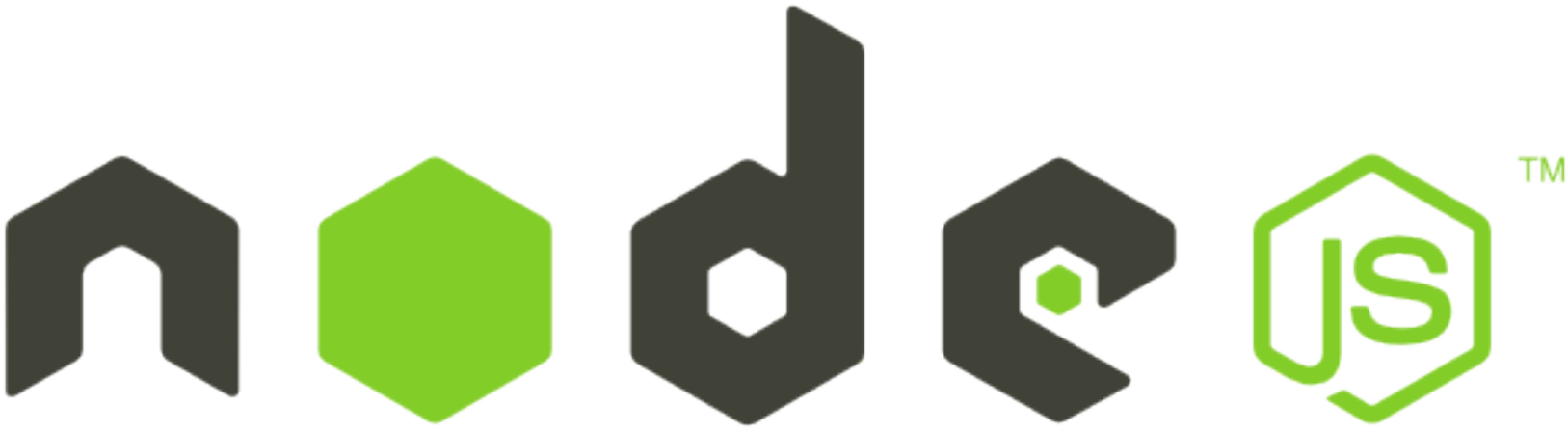
NodeSentry:
Security Architecture for Server-Side JavaScript

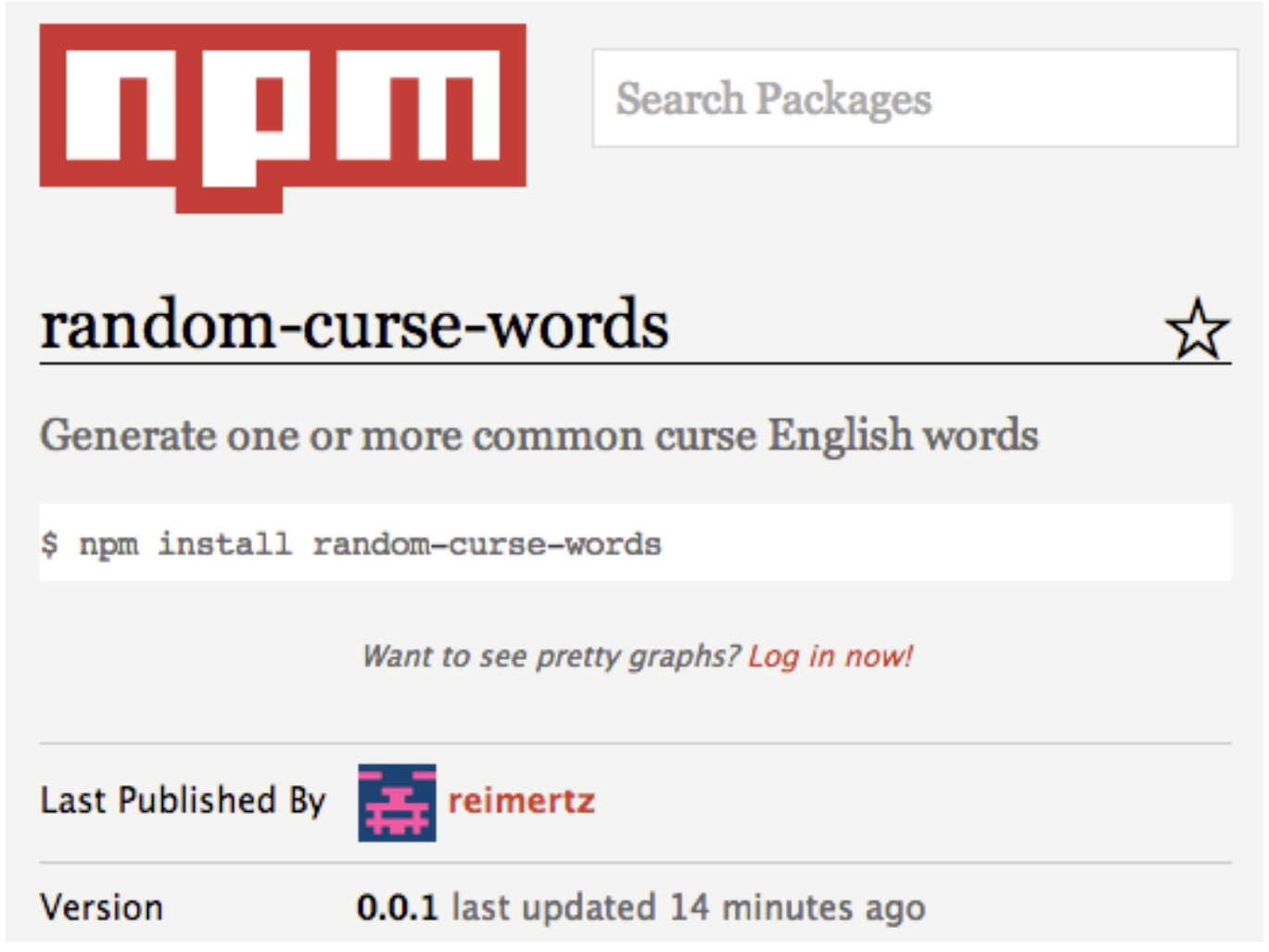
Willem De Groef Fabio Massacci Frank Piessens

iMinds-DistriNet, KU Leuven (BE)

University of Trento (IT)

ACSAC'14 - December 12th, 2014


node
JS
TM


Search Packages
random-curse-words
Generate one or more common curse English words
$ npm install random-curse-words
Want to see pretty graphs? Log in now!
Last Published By
reimertz
Version
0.0.1 last updated 14 minutes ago

npm is the package manager for javascript.

111,650
total packages

31,062,125
downloads in the last day

159,417,778
downloads in the last week

660,279,179
downloads in the last month

Example of vulnerable library

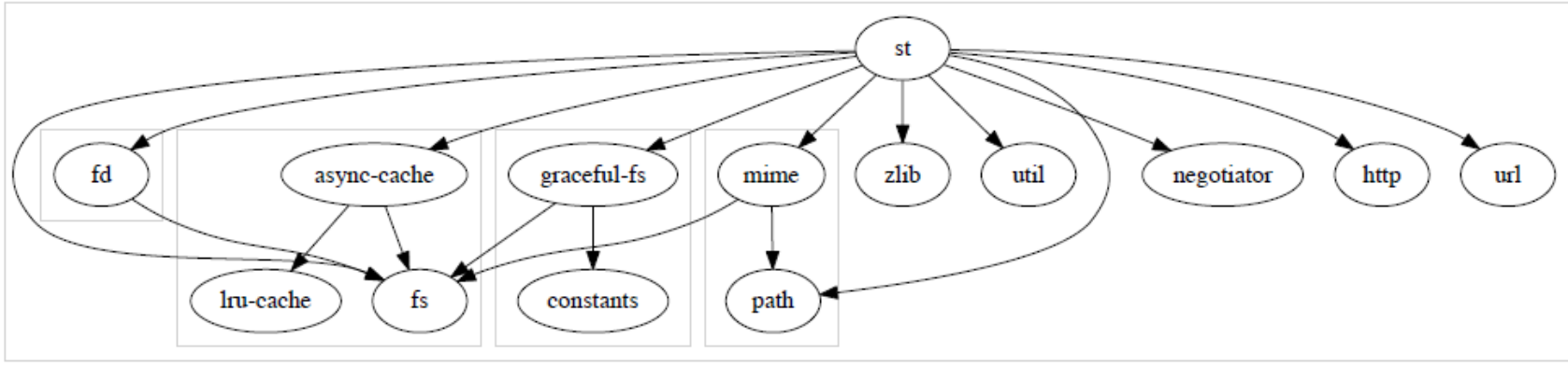
- *st* library downloaded about 400 times each day
- Contained *path traversal vulnerability*

Security Status

Versions prior to 0.2.5 did not properly prevent folder traversal. Literal dots in a path were resolved out, but url encoded dots were not. Thus, a request like `/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/etc/passwd` would leak sensitive data from the server.

As of version 0.2.5, any `'/../'` in the request path, urlencoded or not, will be replaced with `'/'`. If your application depends on url traversal, then you are encouraged to please refactor so that you do not depend on having `..` in url paths, as this tends to expose data that you may be surprised to be exposing.

Problem Statement



Applications depend on many libraries with specific characteristics:

- Non-malicious
- Potentially vulnerable
- Potentially exploitable

Combining JavaScript libraries from **disparate third-parties** to run on a **server** with high privileges

Problem Statement (cont.)

Combining JavaScript libraries from **disparate third-parties** to run on a **server** with high privileges

Researchers proposed a number of solutions for similar *client-side security problems*:

- Static analysis
- Runtime monitoring
- Language-based sandboxing
- Access control
- Information flow control

Problem Statement (cont.)

Combining JavaScript libraries from **disparate third-parties** to run on a **server** with high privileges

None of the current client-side solutions works really well for *server-side JavaScript*.

Research Question

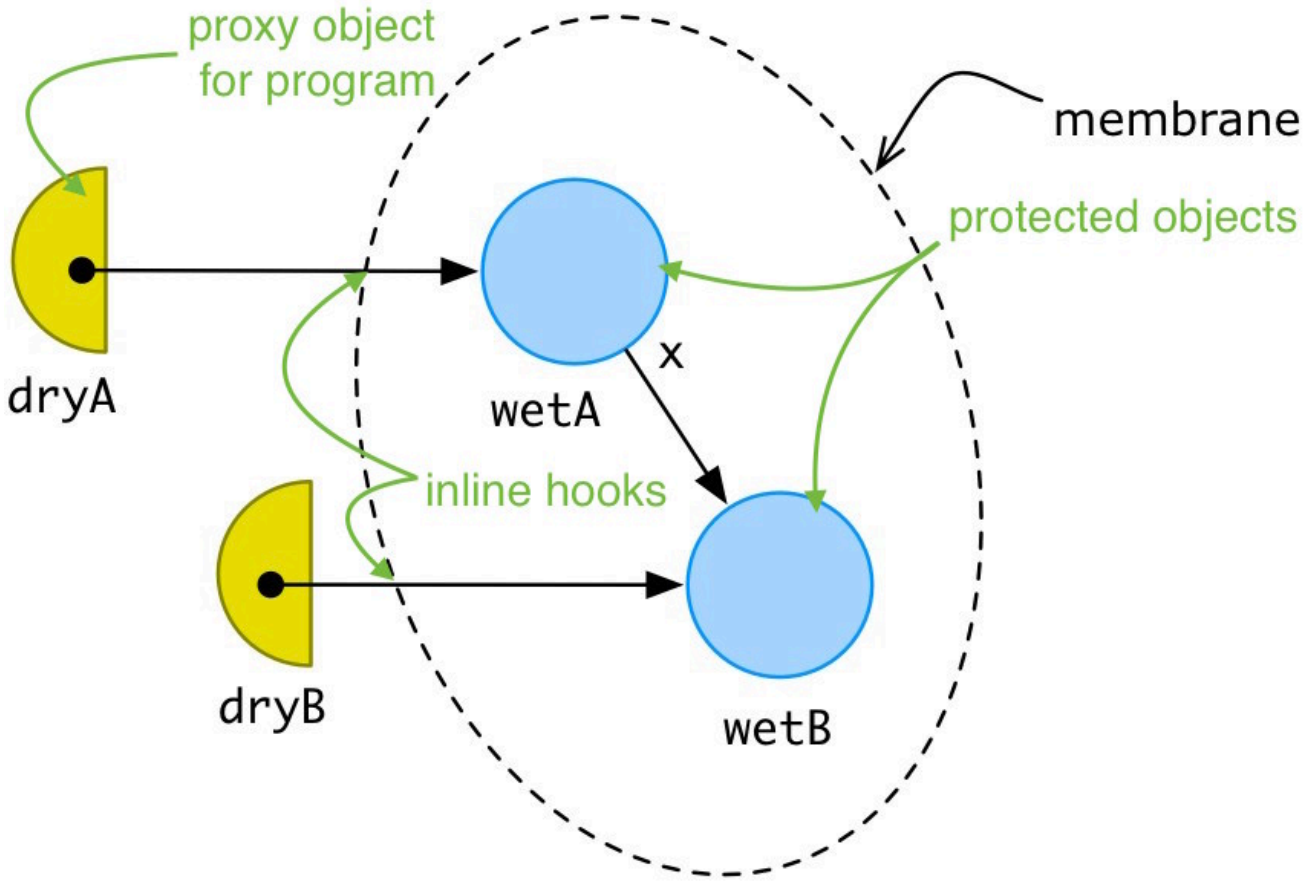
How to *combine* the flexibility of **third-party JavaScript libraries** from a vibrant ecosystem **with strong security guarantees** at an **acceptable performance** at the server-side?

Our solution: NodeSentry

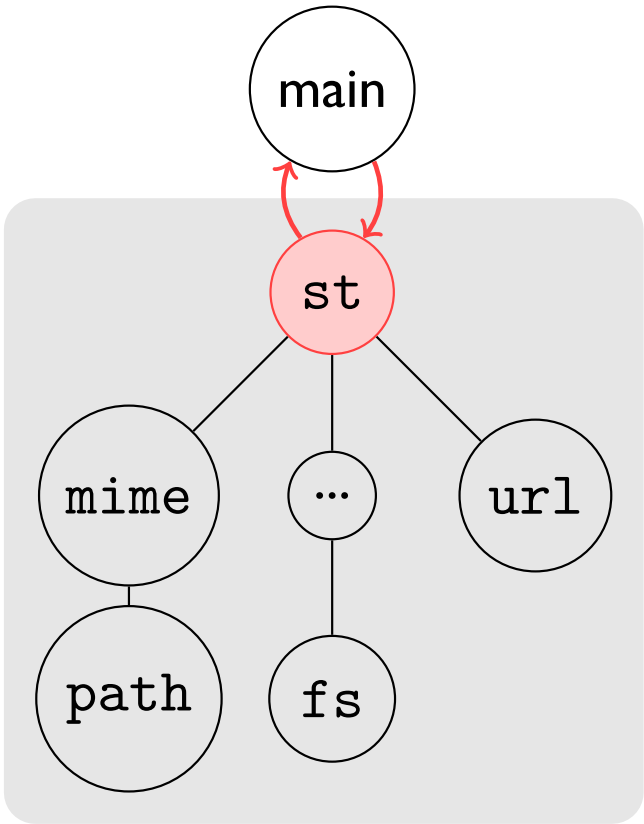
NodeSentry: Least-privilege Integration Framework for Server-side JavaScript Libraries

- Variant of *inline reference monitor* for flexibility
- Wrap each API to intercept interactions (e.g., function calls, ...)
- Relies on *membrane pattern* by Miller and Van Cutsem
- Inline only *hooks* with the *monitor as external component*
- Supports policies that can specify how to *fix executions*

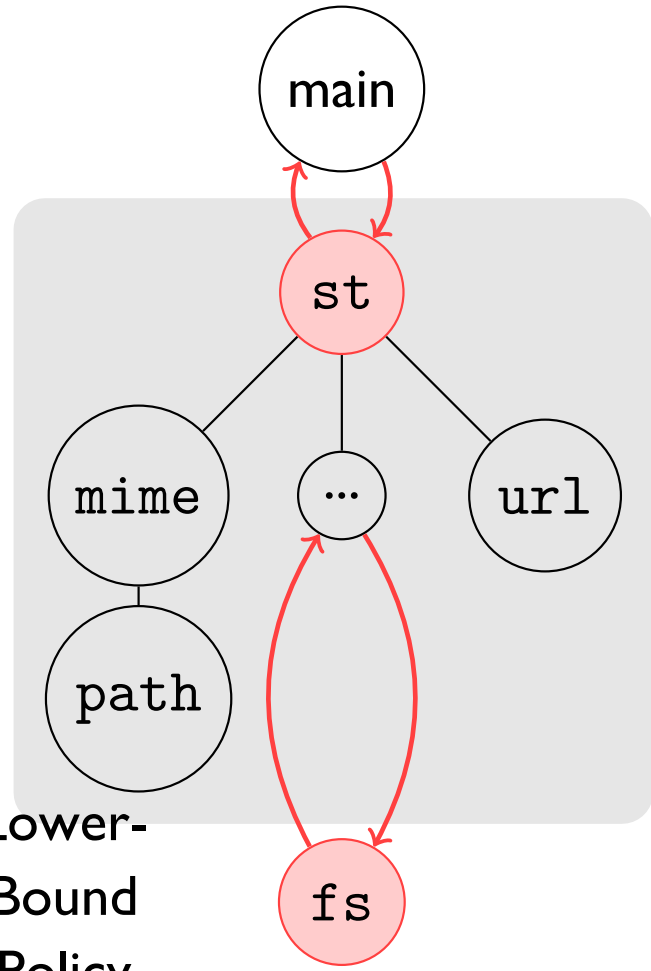
Membrane


proxy object
for program
membrane
protected objects
dryA
wetA
x
inline hooks
dryB
wetB

Security Policies for NodeSentry

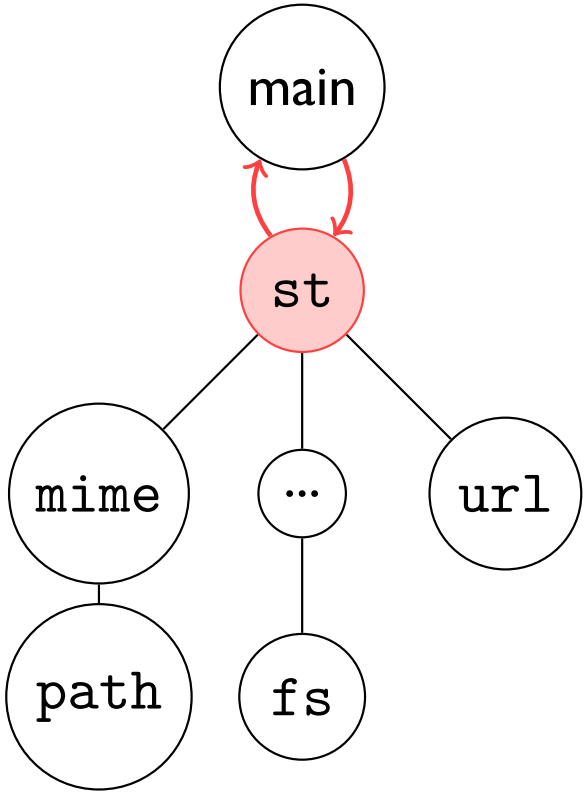


Upper-Bound Policy

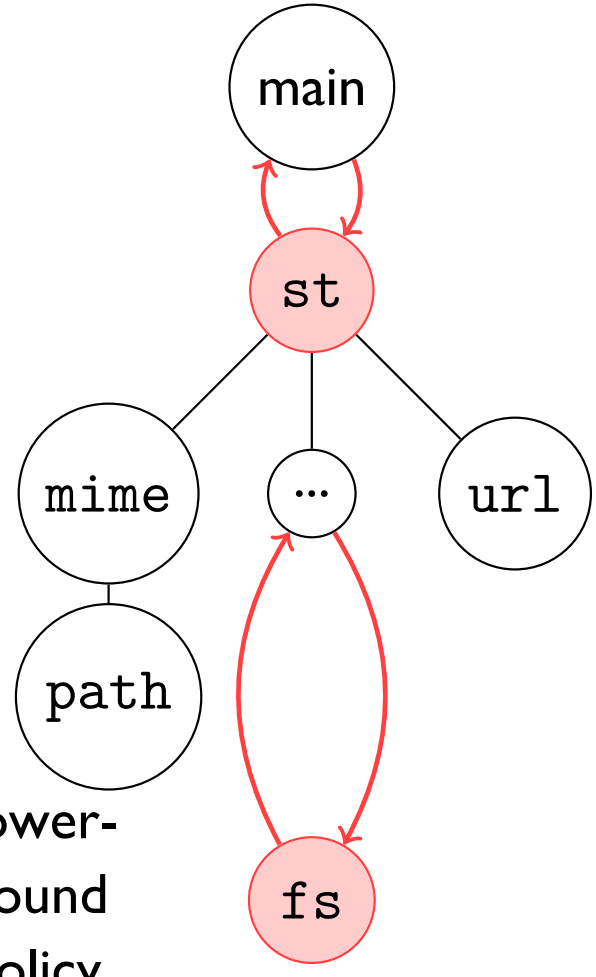


Lower-Bound Policy

Security Policies for NodeSentry (cont.)



Upper-Bound Policy



Lower-Bound Policy

- Implement webapp firewall
- Web-hardening techniques
- Block specific clients

- Application-wide chroot jail
- Fine-grained access control

NodeSentry Demo (1)

```
1 var http = require("http");
2 var st = require ("st");

3 var cwd = process.cwd();
4 var handler = st(cwd);
5 http.createServer(handler).listen(1337);
```

Listing 1: Basic example serving current directory via the `st` library.

NodeSentry Demo (2)

```
1  var Policy = require("nodesentry").Policy;

2  function invalidUrl (reqObj, url) { /* do some checks */ };
3  function errorPage () { return "/404.html"; }

4  var st_policy = new Policy()

5      .on("IncomingMessage.url")
6          .return(errorPage).if(invalidUrl)

7  .build() // build the actual policy object
```

Listing 2: Example policy to secure the `st` library.

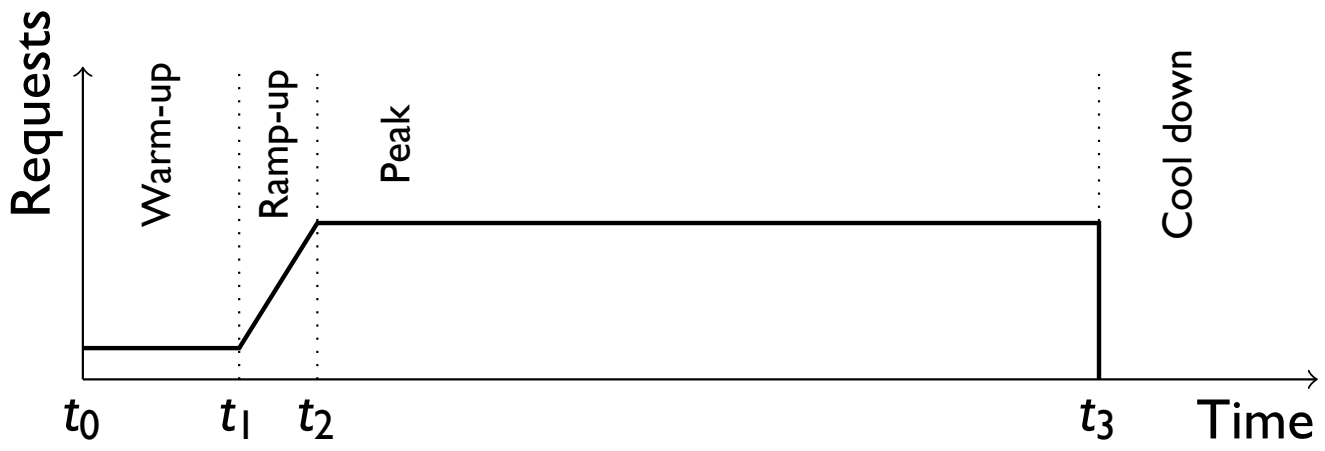
NodeSentry Demo (3)

```
1 require("nodesentry");

2 var http = require("http");
3 var st = safe_require("st", /* policy */);

4 var cwd = process.cwd();
5 var handler = st(cwd);
6 http.createServer(handler).listen(1337);
```

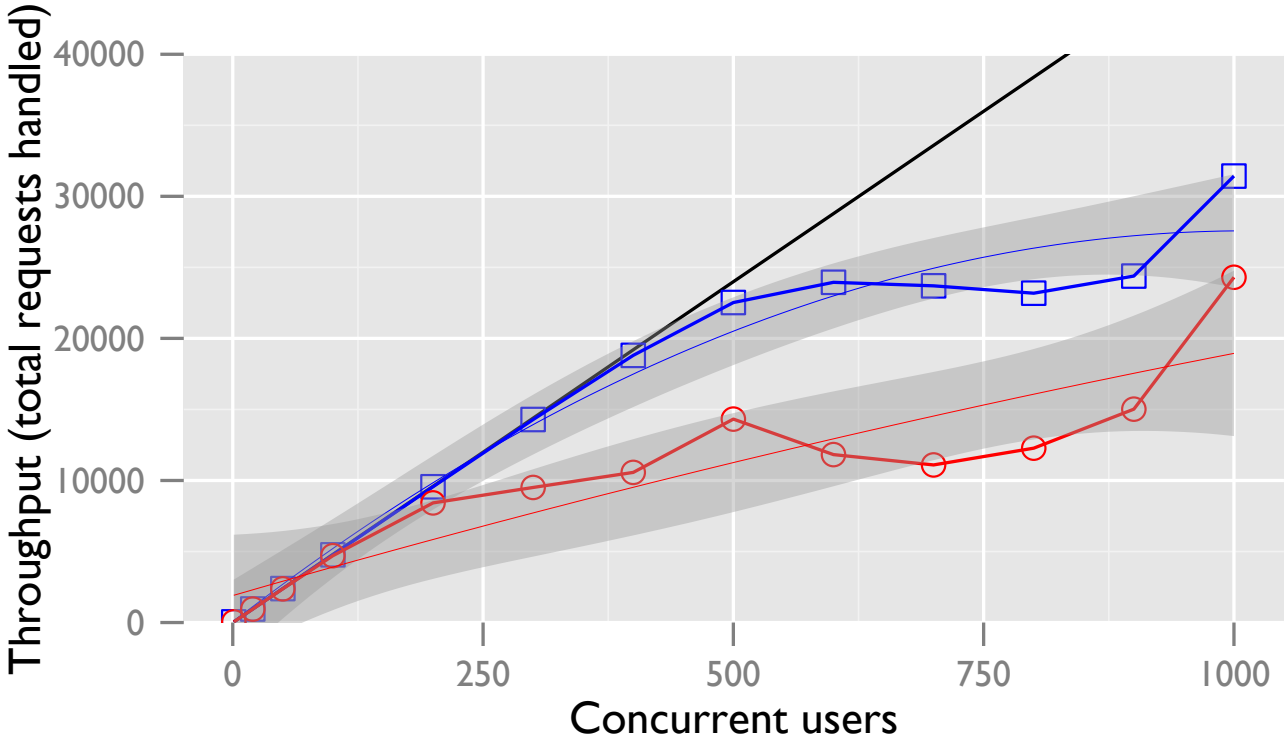
Listing 3: Secured example serving current directory via the `st` library.

Evaluation - Experimental Setup

- Performance is king for server-side JavaScript
- Verify the impact of NodeSentry on two performance drivers:
  1. Throughput, i.e., the amount of tasks or total requests handled
  2. Capacity, i.e., the total amount of concurrent users/requests handled
- Exerpimental setup for 1 to 1000 concurrent users:



Evaluation of *throughput*



- Standard Node.js
- NodeSentry
- 95% confidence interval

Evaluation Summary

- Level of performance *comparable to commercial security events monitoring system* [Gartner, 2014]
  - Small deployment: < 300 event sources; 1,500 events/sec
  - NodeSentry: 300 concurrent users; about 10,000 requests/sec
- Trade-off between performance and security (about 50% overhead)
- Decreases when other conditions stretch the performance
- Overhead is mostly due to the peculiarities of membranes
  - Our implementation of membranes in NodeSentry uses experimental version of proxies
  - Performance will be better once proxies are consolidated in main Node.js development

Conclusions

How to *combine* third-party *JavaScript libraries* on the *server* with *strong security guarantees*?

- Address problem via a *least-privilege integration apprach* based on *membraned libraries*
  - Simple & uniform policy infrastructure to
    - Subsume & combine common web-hardening techniques
    - Common & custom access control policies
  - Performance comparable to commercial security events monitoring system
- More?
  - https://npmjs.org/nodesentry
  - Willem.DeGroef@cs.kuleuven.be