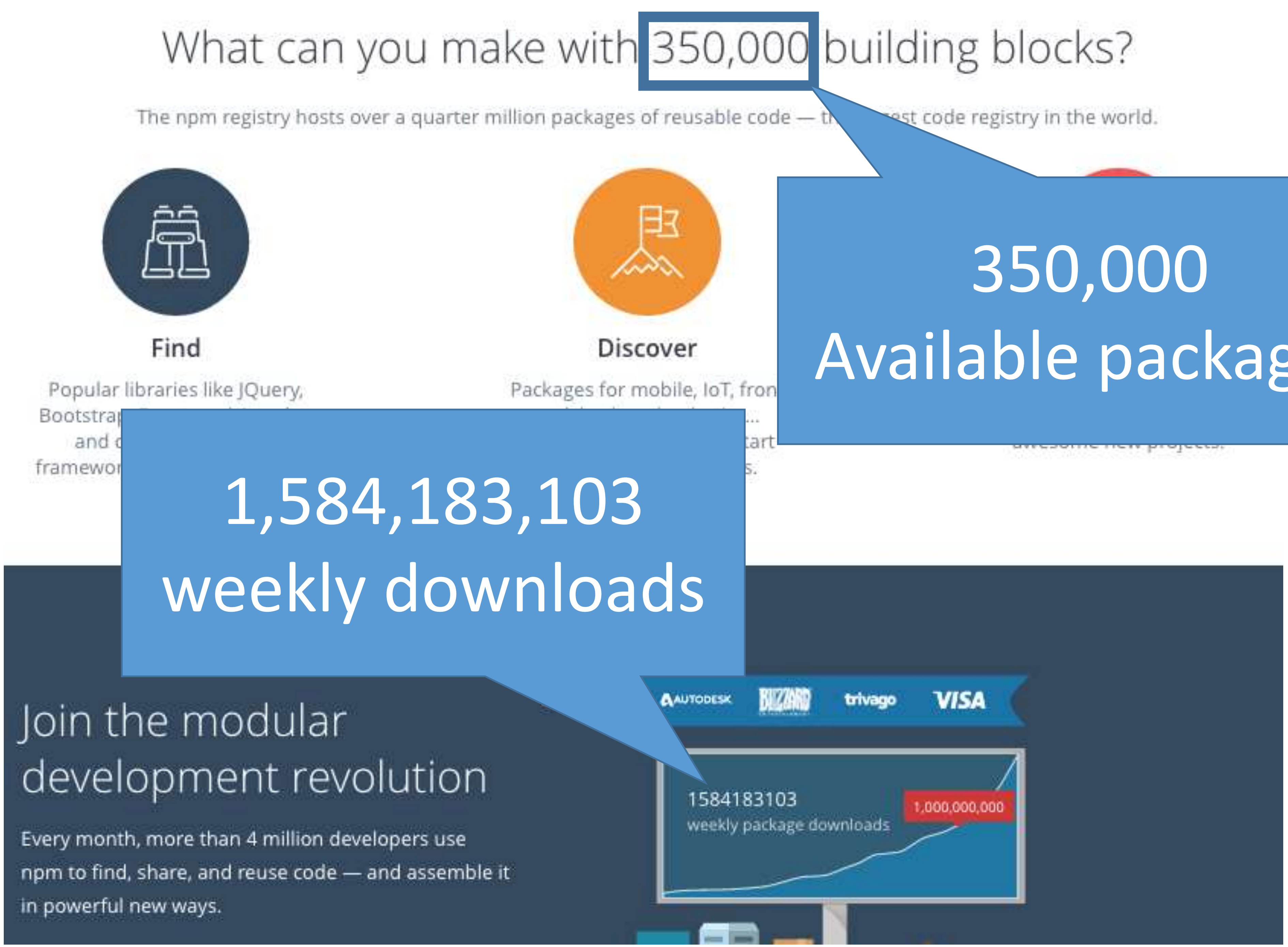
The Tearless server-side JavaScript security architecture

Willem De Groef

Context wrt Tearless

- Tearless project envisions future in which multi-tier web apps are developed as a single artefact that spans all tiers
- One of its objectives is to investigate programming constructs for implementing confinement-related security policies
- This talk is about specific security mechanism (developed at KULeuven)
  - NodeSentry
  - Isolate and restrict untrusted JavaScript libraries at runtime



What can you make with 350,000 building blocks?

The npm registry hosts over a quarter million packages of reusable code — th est code registry in the world.

Find

Popular libraries like JQuery, Bootstrap and framewor

Discover

Packages for mobile, IoT, fron tart s.

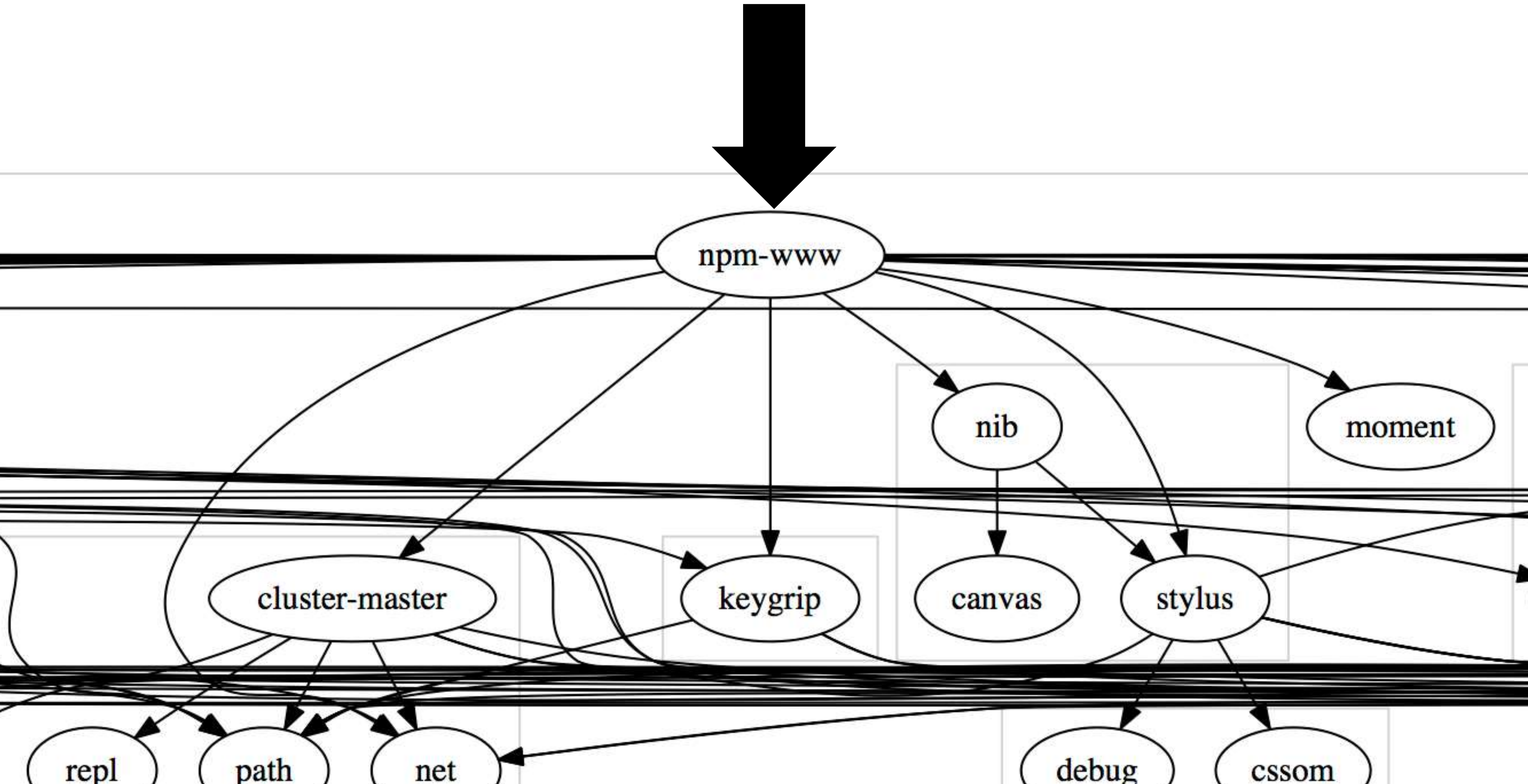
awesome new projects.

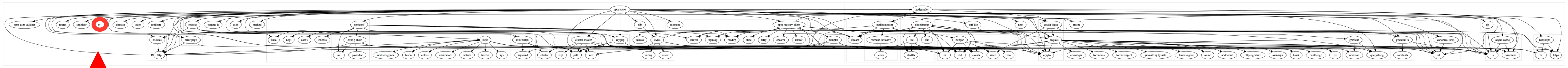
Join the modular development revolution

Every month, more than 4 million developers use npm to find, share, and reuse code — and assemble it in powerful new ways.

350,000
Available packages

1,584,183,103
weekly downloads

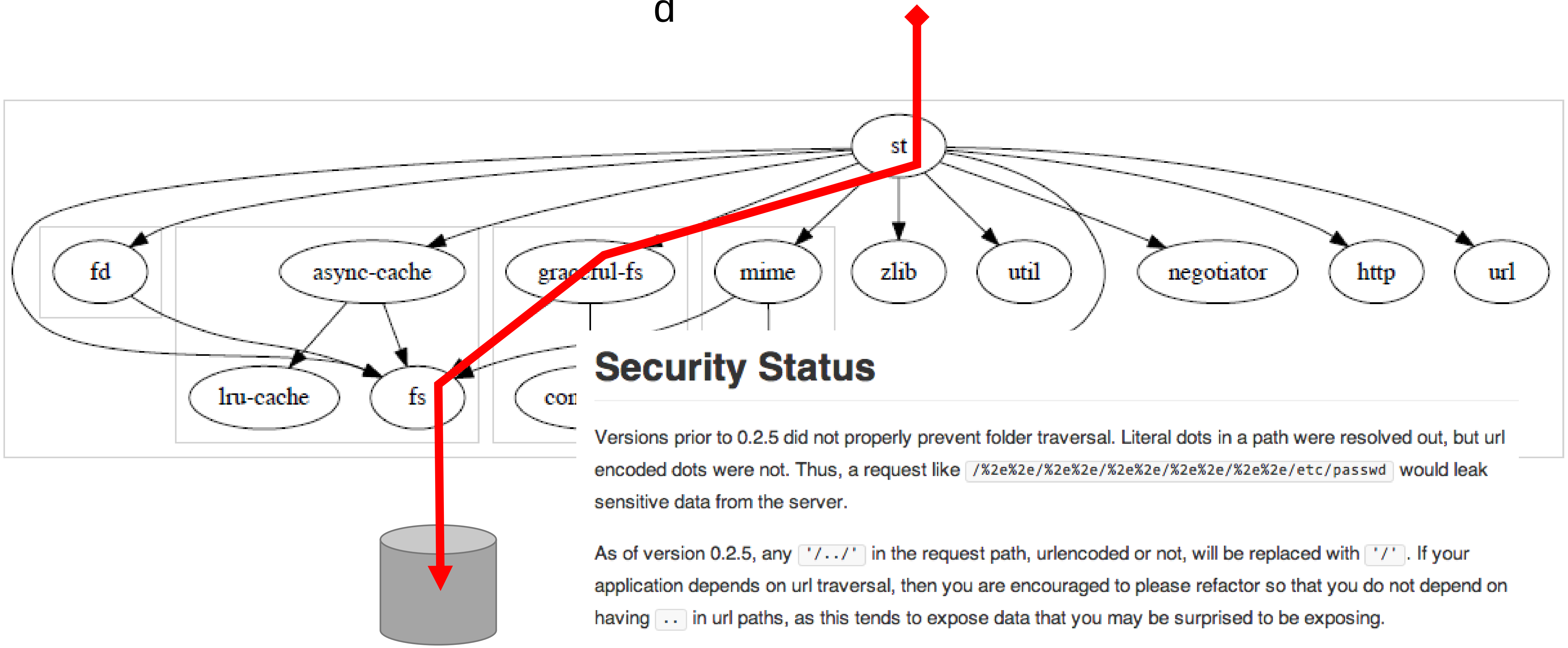

npm-www
nib
moment
cluster-master
keygrip
canvas
stylus
repl
path
net
debug
cssom



The Big Bug

The bug found by Charlie Somerville is a classic "static file leakage" bug: the code that runs the npm website served static files through a module called st. It was possible, through a carefully encoded URL, to get st to serve any file it could see, not just the ones in the static content directory, and you could also list the contents of directories, so it was very easy to go looking for sensitive files.

/%2e%2e/%2e%2e/etc/passw
d



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.

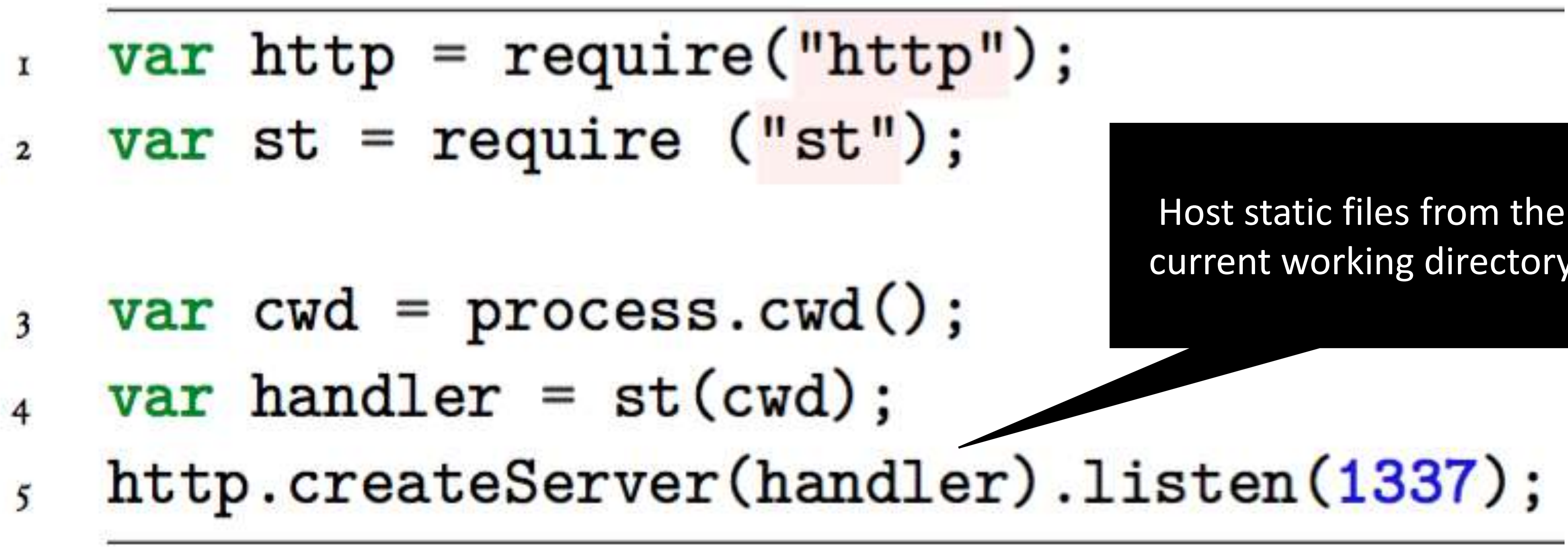
As of version 0.2.5, any `'/../'` in the request path, urlencoded or not, will be replaced with `'/'`. If your application depends on url traversal, then you are encouraged to please refactor so that you do not depend on having `..` in url paths, as this tends to expose data that you may be surprised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

Server-side security framework

- Isolate and restrict (buggy) third-party JavaScript libraries at runtime, without the need for modifications
- NodeJS easily integrates third-party libraries via NPM
  - Without a clear vetting or trust mechanism
- Researchers proposed a number of solutions with common pattern for similar client-side security problems
  - Isolated unit/sandbox, completely cut off from any sensitive functionality
  - Provide mediated access to sensitive parts
  - Mainly vary in policy specification and isolation technique

NodeSentry

- Variant of inline reference monitor for flexibility
- Wrap each API in a membrane to intercept interactions
- Relies on membrane pattern and Trustworthy Proxies *[Van Cutsem & Miller, ECOOP'13]*
- Inline only hooks with the monitor as external component
- Support policies that can specify how to fix executions
  - What to do with a violation of a security policy?



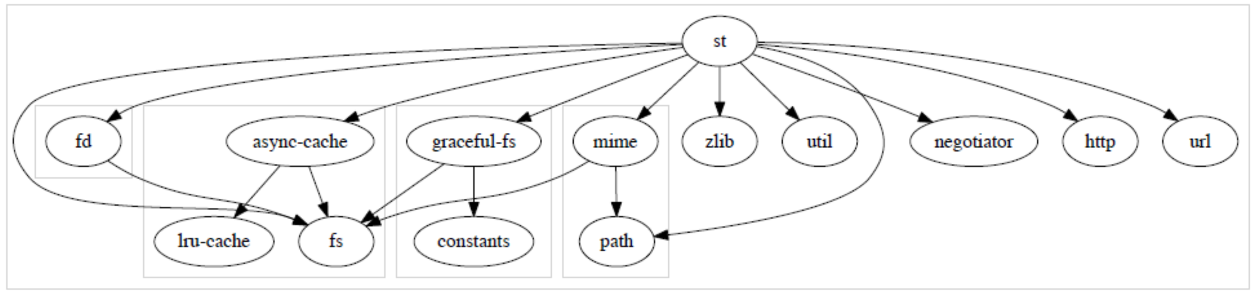
```
var http = require("http");
var st = require ("st");

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

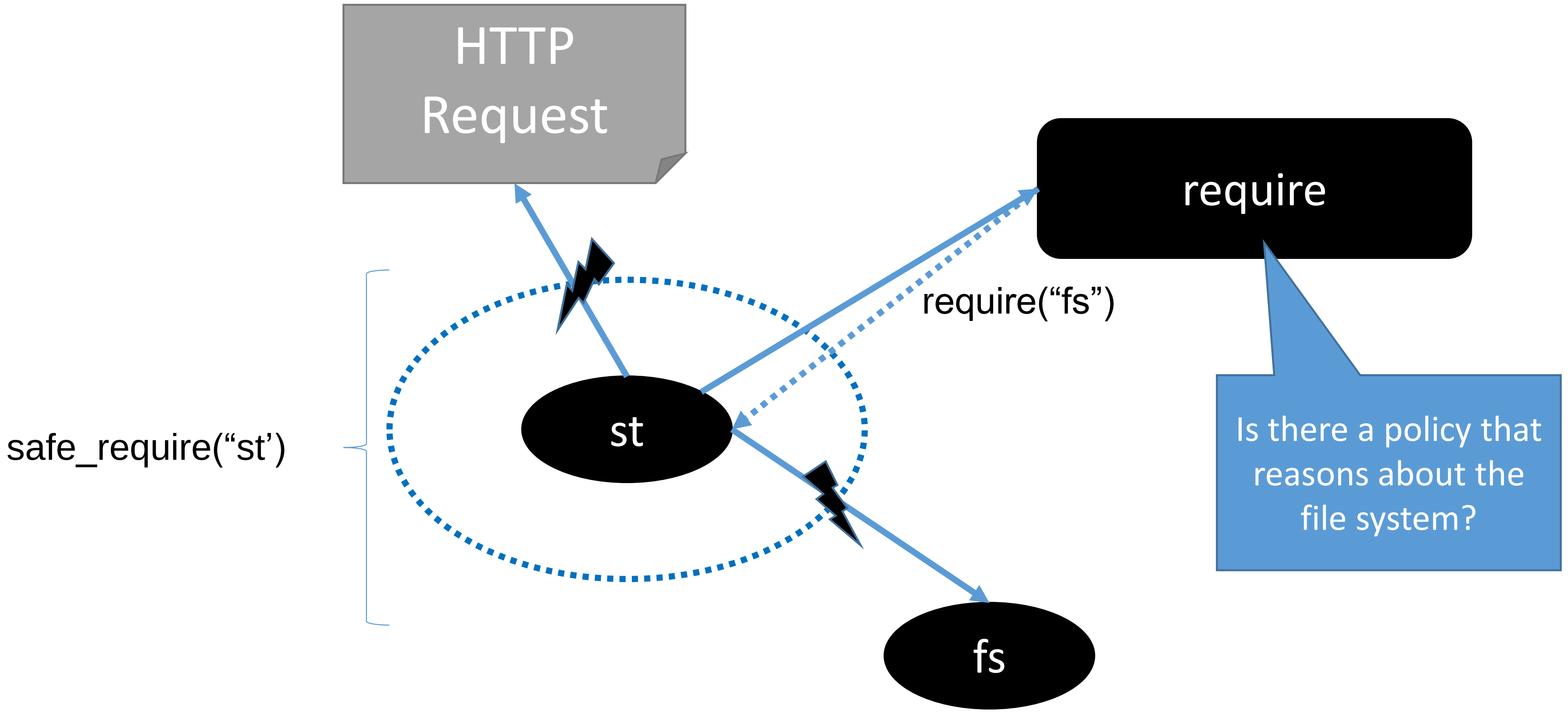
Host static files from the current working directory

Full mediation between library and the environment/application/OS

```
require("nodesentry");

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

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

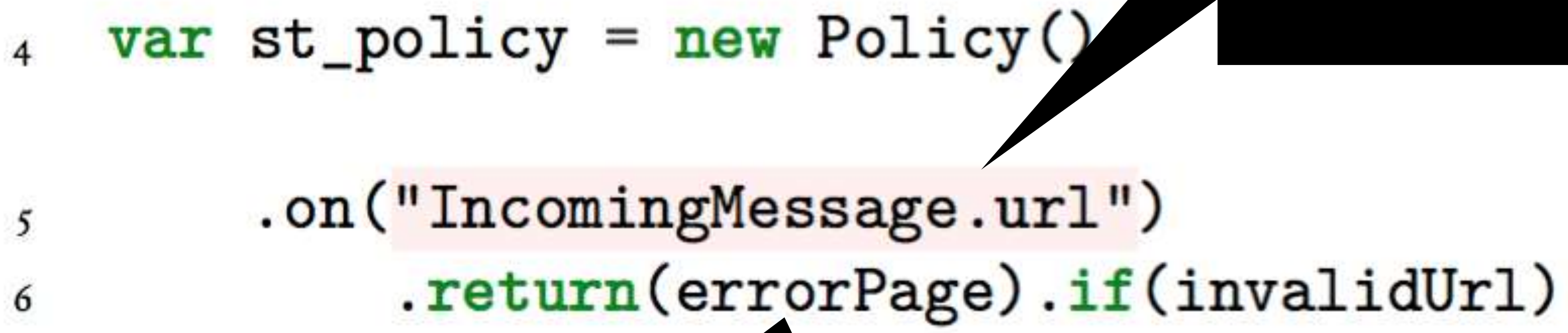
How does it work?





Example policy

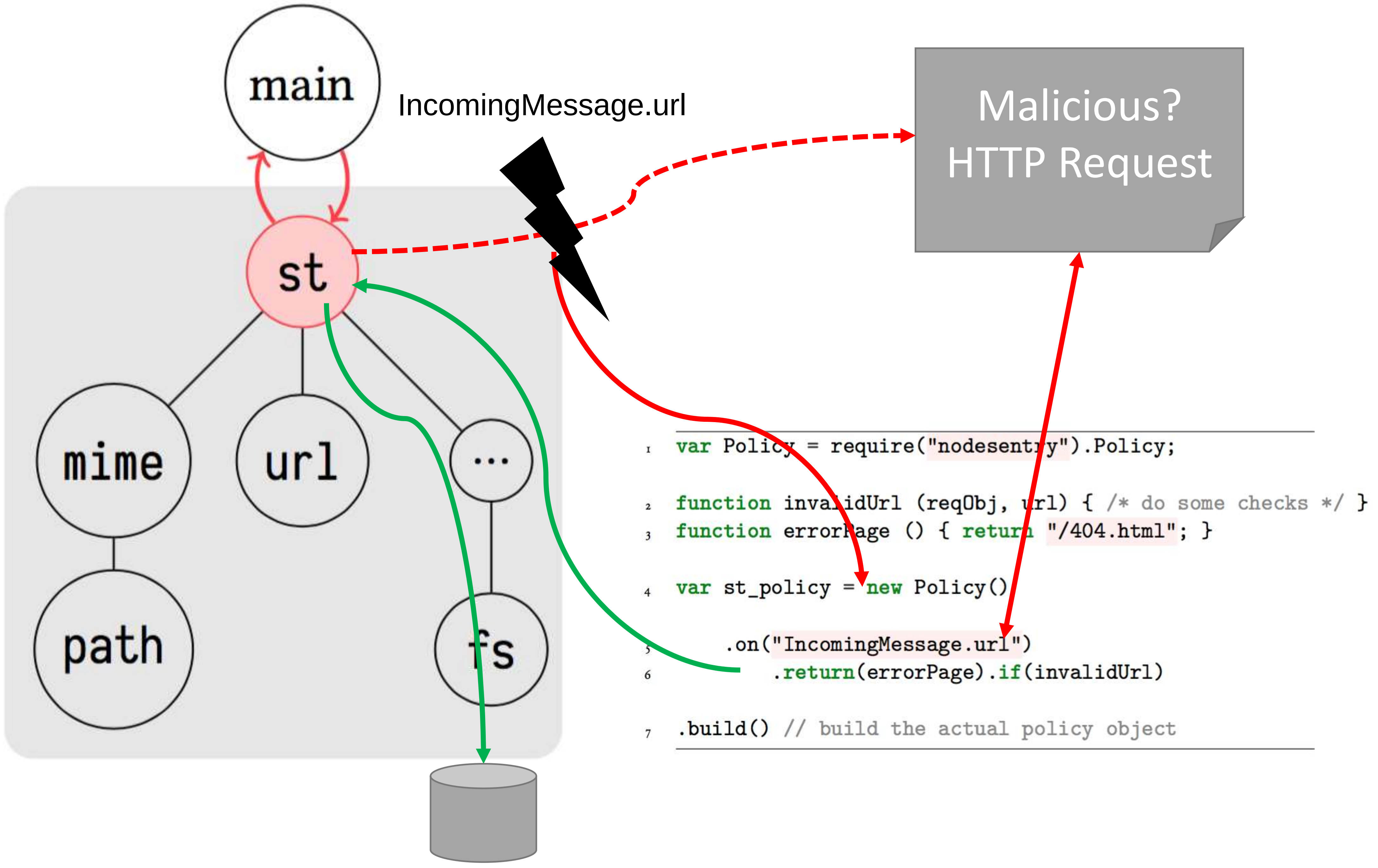
API called when library tries to read the URL property of an incoming HTTP request



```
var st_policy = new Policy()

    .on("IncomingMessage.url")
        .return(errorPage).if(invalidUrl)
```

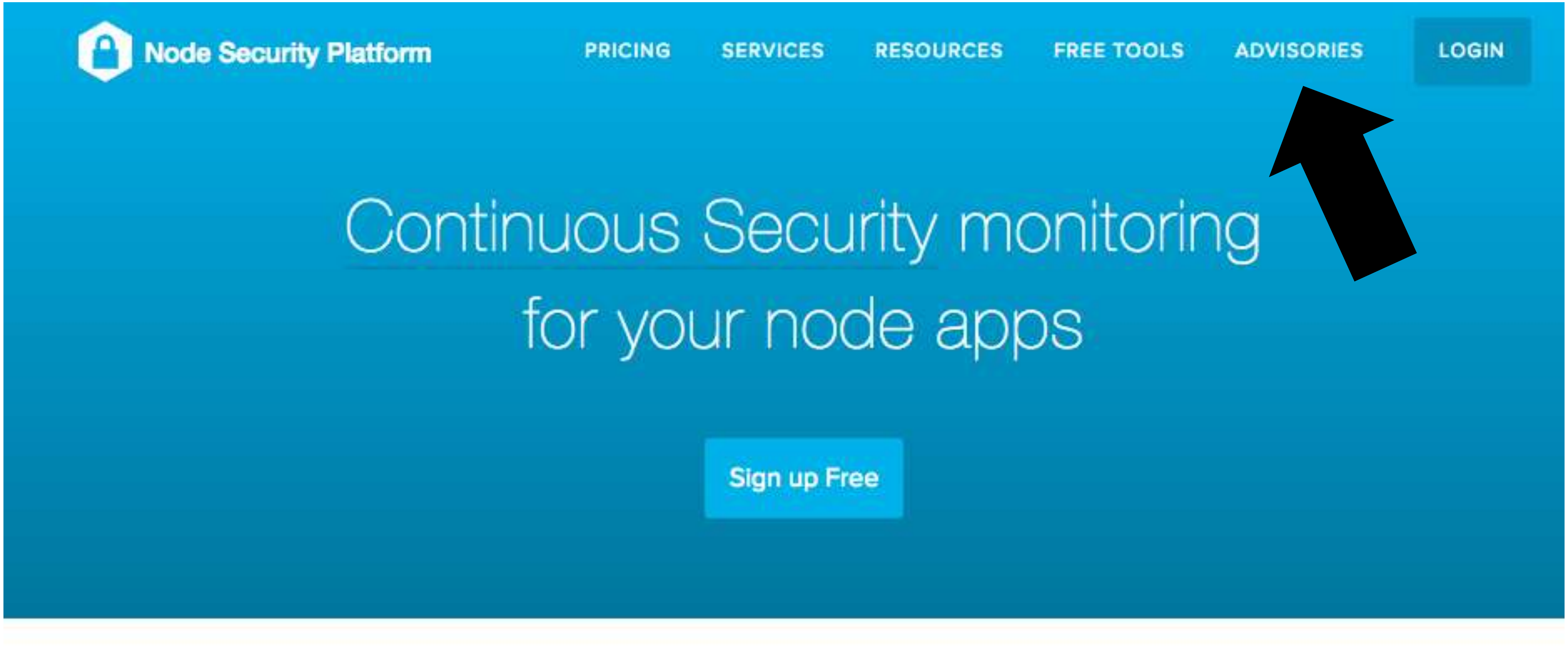
Return /error404.html

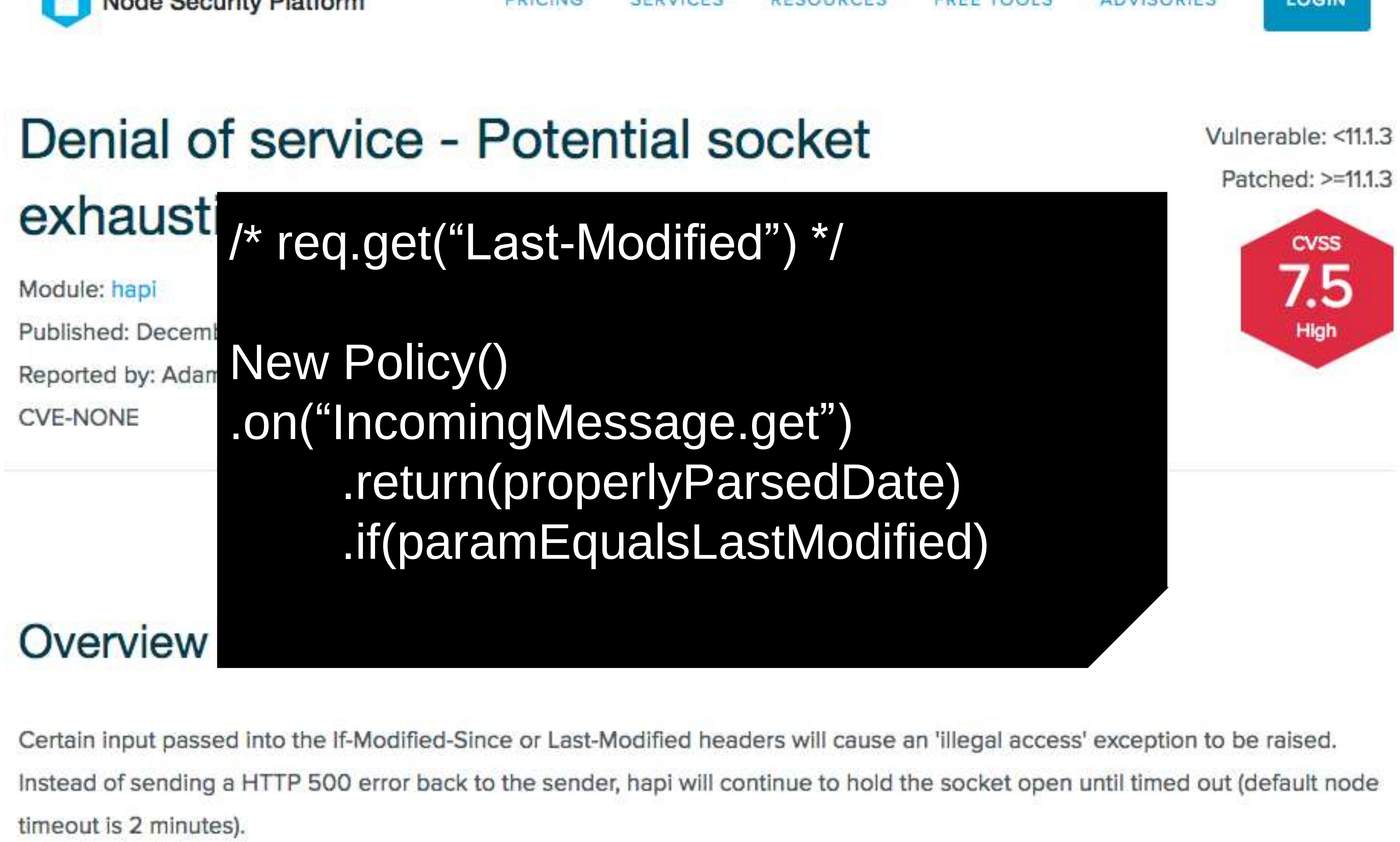
Test for .. and /



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

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

var st_policy = new Policy()

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

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

More experiments





Denial of service - Potential socket exhausti

Vulnerable: <11.1.3

Patched: >=11.1.3

CVSS 7.5 High

Module: hapi

Published: Decem

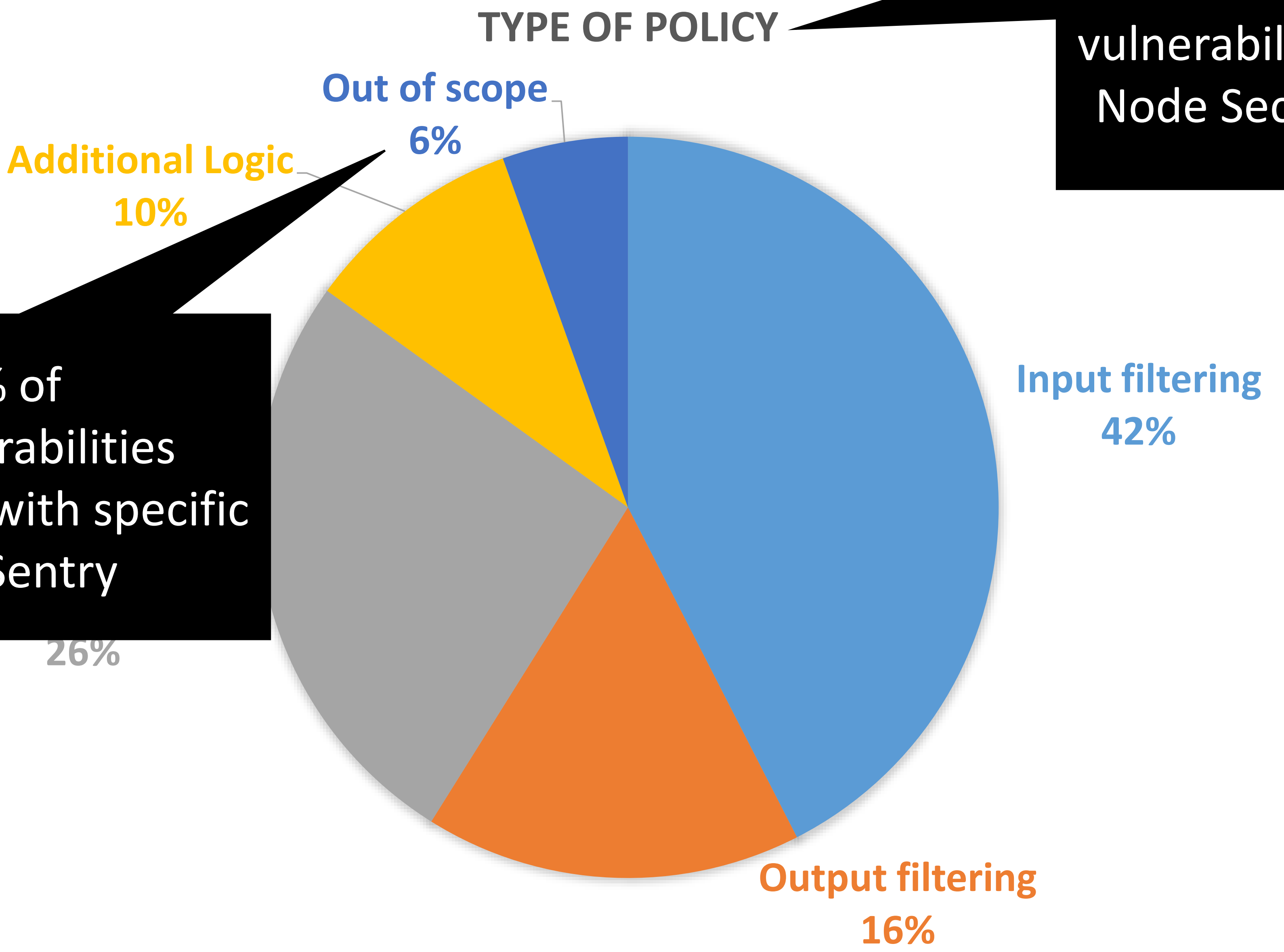
Reported by: Adam

CVE-NONE

Overview

Certain input passed into the If-Modified-Since or Last-Modified headers will cause an 'illegal access' exception to be raised. Instead of sending a HTTP 500 error back to the sender, hapi will continue to hold the socket open until timed out (default node timeout is 2 minutes).

```
/* req.get("Last-Modified") */

New Policy()
.on("IncomingMessage.get")
        .return(properlyParsedDate)
        .if(paramEqualsLastModified)
```



Based on 73 reported vulnerabilities from the Node Security Project

More than 90% of reported vulnerabilities could be fixed with specific policy in NodeSentry

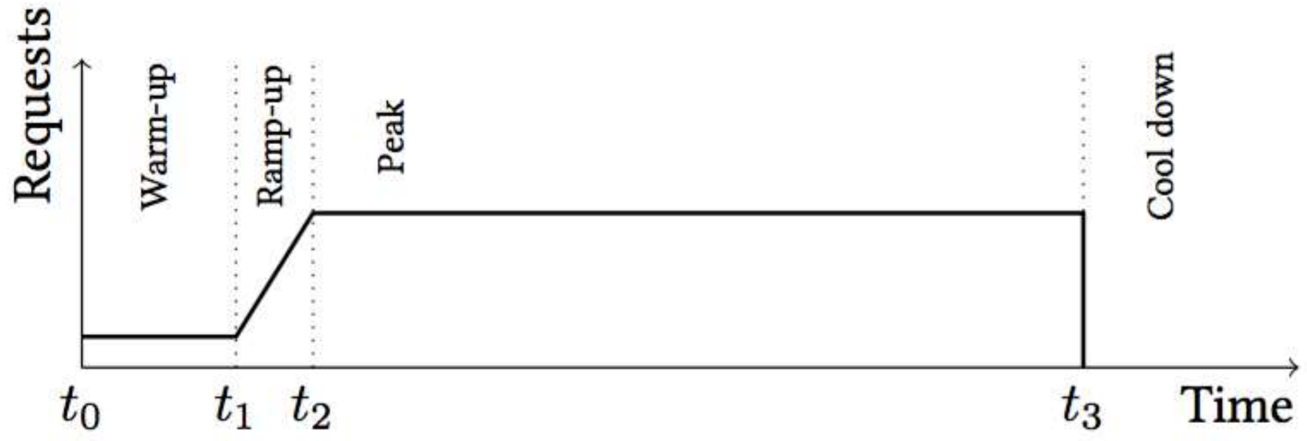
More research needed

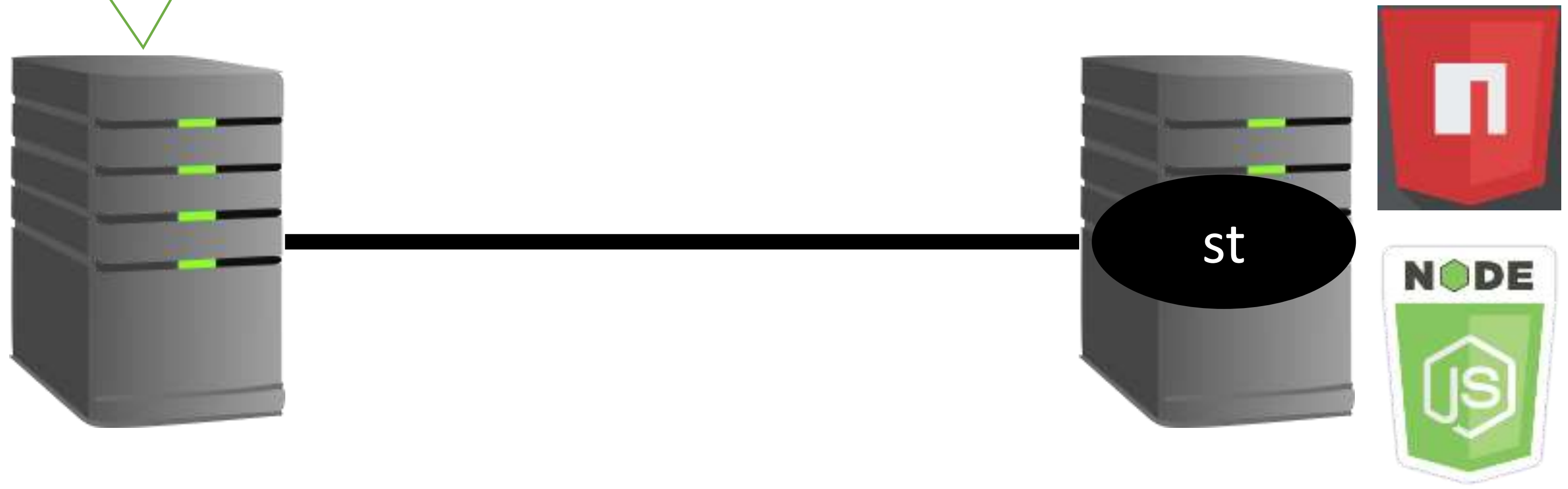
- Other types of web application firewall-like policies
  - Blacklisting clients
  - Enforcing web-hardening techniques like HSTS
- Fine-grained access control
- Access control policies on general ‘capability’ categories (e.g., no file system access) *[Vanacker et al, ACSAC’12]*
  - Indicated via annotation in tearless application

- More research to understand the possibilities of such a general and flexible framework

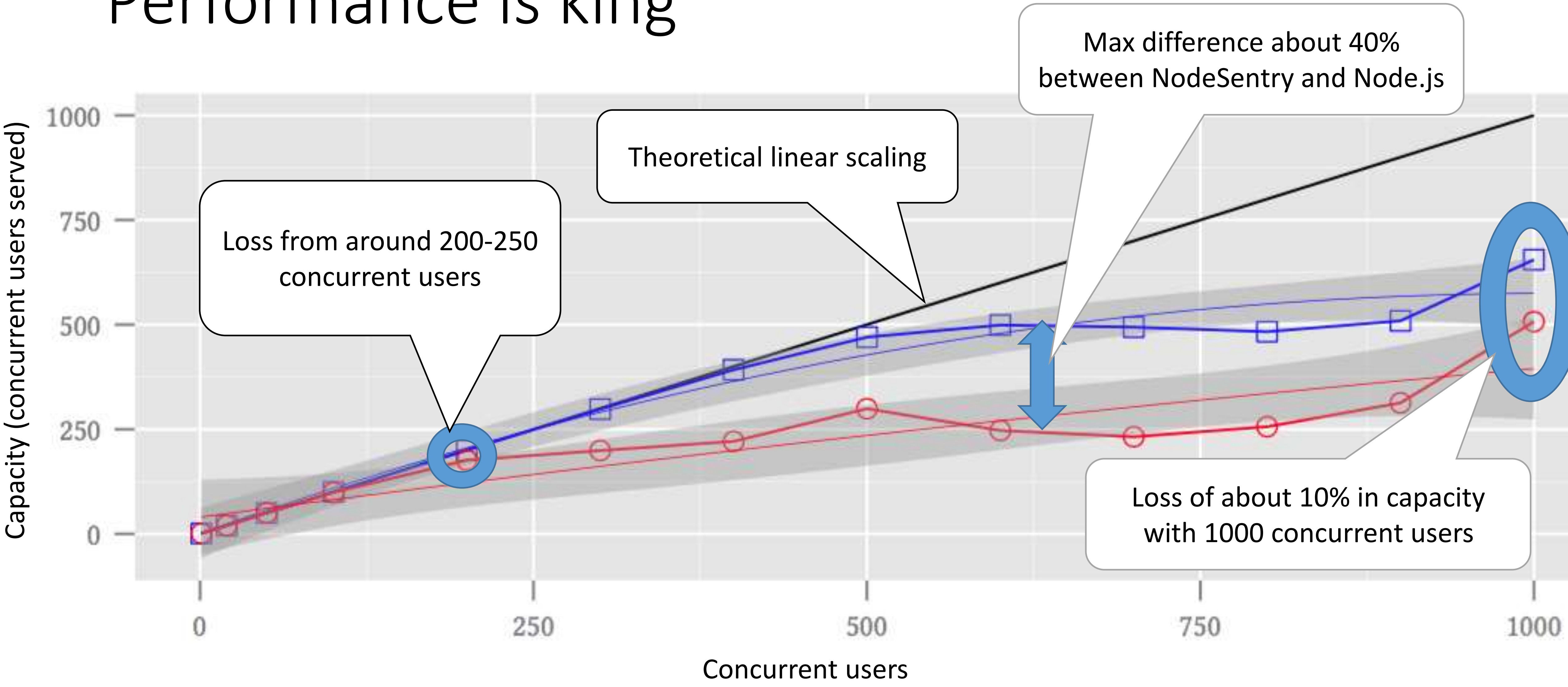
Performance benchmarks setup

Scalar – highly scalable benchmarking framework *[Heyman et al, 2014]*





Performance is king



Performance Benchmarks - Conclusions

- Level of performance comparable to commercial security events monitoring systems [Gartner, 2014]
- Trade-off between performance and security
- Policy impact decreases when other conditions stretch performance

NodeSentry – Wrap up

- Goal: restrict impact of –untrusted? – third-party libraries
  - At runtime
  - Without need for modification of the underlying runtime
- NodeSentry:
  - Provides initial compartmentalization technique
  - Based on membranes and concept of reference monitor
  - Provides platform for enforcing broad range of security policies
  - Performs OK for research prototype

The Tearless server-side JavaScript security architecture

Willem De Groef

Willem.DeGroef@cs.kuleuven.be