node.js security

Maciej Lasyk

SEConference

Kraków, 2014-05-09

Sysadmin about node.js security?

- not only sysadmin ;)
- node needs thorough understanding of whole infra
- 14+ years of exp software dev. / sysop
- currently “noding” 4 prv & Fedora

JS security recap – evals & co

eval() like fncs takes string argument and

evalute those as source code

JS security recap – evals & co

eval() like fncs takes string argument and

evalute those as source code

srsly – who does that?

JS security recap – evals & co

```
var x = req.body.x;
var y = req.body.y;
var sum = eval(a + "+" + b);
```

JS security recap – evals & co

```
var x = req.body.x;
var y = req.body.y;
var sum = eval(a + "+" + b);
```

what if attacker fills 'x' with:

some.super.class.wipe.the.database('now');

LOL :)

JS security recap – evals & co

not only evals:

```
setInterval(code,2)

setTimeout(code,2)

str = new Function(code)
```

Chrome CSP denies those also :)

JS security recap – global namespace pollution

- node.js is single threaded
- all variable values are common
- one could thrtically change bhv of others reqs
- watch out for globals then!

JS security recap – global namespace pollution

some very awful example:

```
var auth = false;
app.get('/auth', function(req, res) {
    if(legit) { auth = true; res.send("success");
});

app.get('/payments-db', function(req, res) {

   if (auth) res.send("legit to see all payments data");

   else res.send("not logged in");
})

app.listen(8080);
```

JS security recap – global namespace pollution

So now imagine..

global namespace pollution + evals & co

JS security recap – global namespace pollution

So now imagine..

global namespace pollution + evals & co

JS security recap – global namespace pollution

So now imagine..

global namespace pollution + evals & co

Watch out who you are hiring

JS security recap – strict mode

- let's throw all errors!
- declare variables!
- global namespaces help

JS security recap – strict mode

```
"use strict";

function testFunction(){
    var testvar = 4;
    return testvar;
}

// This causes a syntax error.
testvar = 5;
```

JS security recap – strict mode

```
// This causes a syntax error:

"use strict";
testvar = 5;


// This is ok:

"use strict";
var testvar = 0;
testvar = 5;
```

JS security recap – strict mode

- evals & co are not that insecure now
- no access to caller and args props
- enable globally or for some scope
- what about strict mode in 3rd party mods?

JS security recap – strict mode

```
"use strict";

function do_smt() {
   do_smt.caller; // no way :)
   do_smt.arguments; // no way :)
}
```

JS security recap – strict mode

```
"use strict";
eval("var smt = 123");
console.log(smt); // sorry – ReferenceError
```

JS security recap – strict mode

```
"use strict";
eval("var smt = 123");
console.log(smt); // sorry – ReferenceError
```

But watch out:

```
"use strict";
var smt = 0;
eval("smt = 123");
console.log(smt); // outputs “123” properly
```

JS security recap – object properties

- writable: RO/RW
- enumerable: no loops enumeration
- configurable: deletion prohibited
- all default set to True so watch out

JS security recap – object properties

```
var obj = {}; obj.prop = "LOL";

// OR:

Object.defineProperty(obj, "prop", {

    writable: true,

    enumerable: true,

    configurable: true,

    value: "LOL"

})
```

JS security recap – object properties

```
// RO & immutable property def:
var obj = {};
Object.defineProperty(obj, "prop", {
    value: "LOL"
});
```

JS security recap – static code analysis

- If not doing it already – just do
- Commit hooks in (D)VCSes
- JSHint / JSLint
- Create policy for static code analysis
- Update & check this policy regularly

about node.js: what's up?

- current stable version 0.10.28

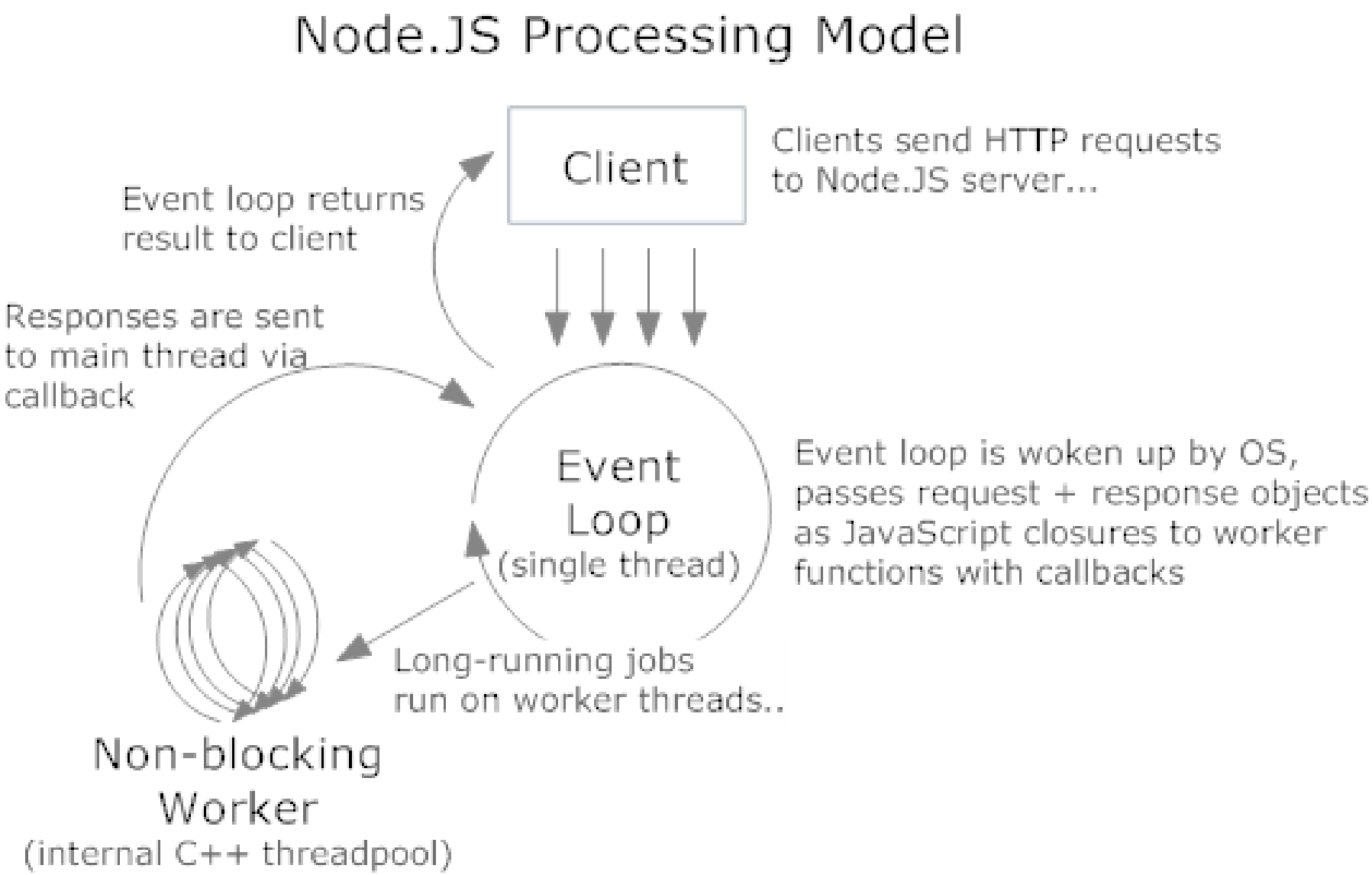
- who is using node?

https://github.com/joyent/node/wiki/Projects,-Applications,-and-Companies-Using-Node

- Operating Node.js in production/Bryan Cantrill (Joyent)

- Bill Scott,“Clash of the Titans: Kraken | Node.js @ paypal”

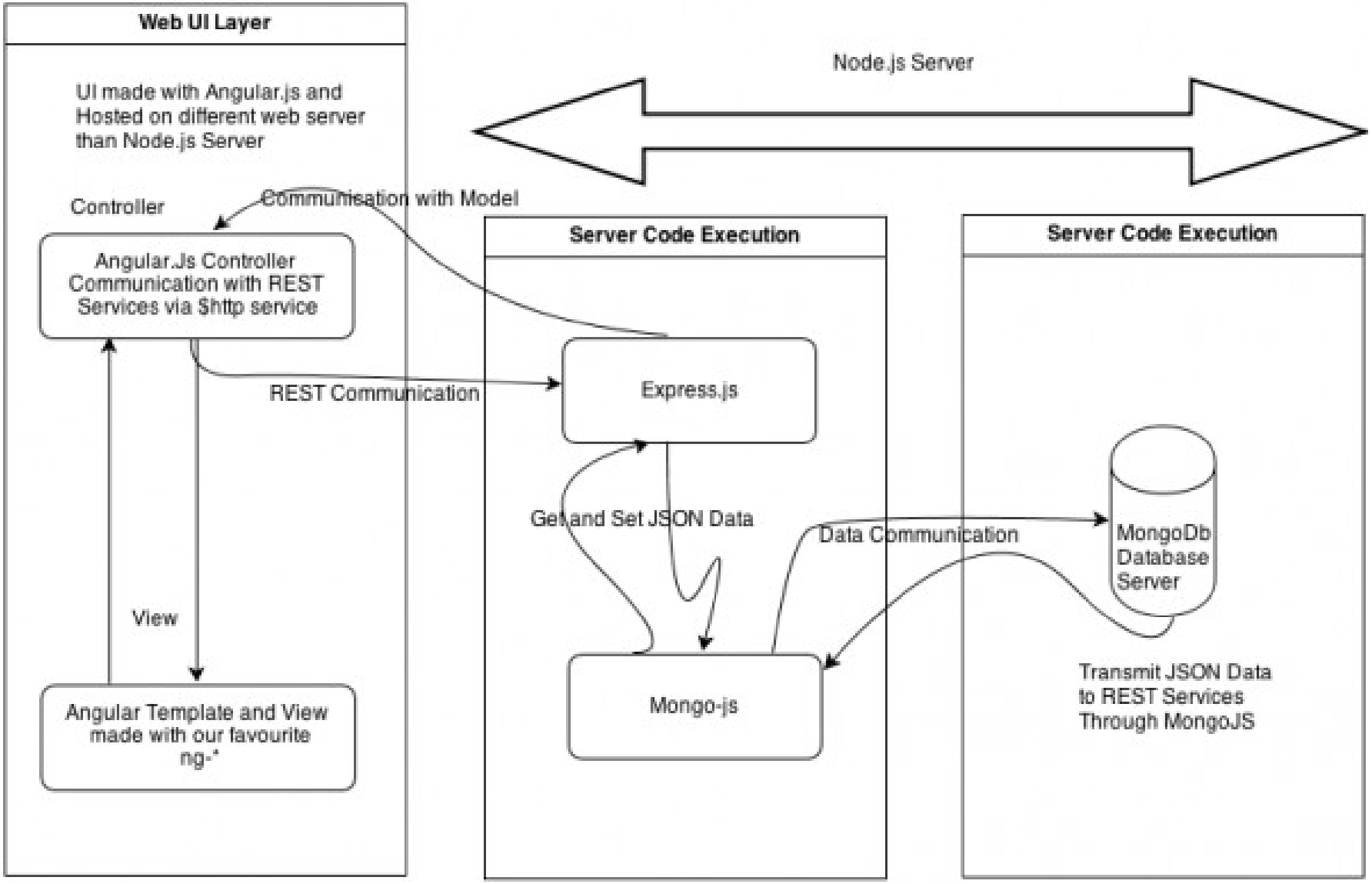
about node.js: model



about node.js: concurrency

- node.js is single threaded (let's say)
- multi – core? child processes (cluster.fork)
- Linux containers ftw!

about node.js: SPA



(thx Sekurak.pl for this image)

node.js sec: exploits anyone?

- http://seclists.org/bugtraq – 0 hits
- http://osvdb.org – 2 hits
- http://1337day.com, http://www.exploitdb.com – 1 hit
- http://nodesecurity.io/advisories – 4 hits

node.js sec: exploits anyone?

- http://seclists.org/bugtraq – 0 hits
- http://osvdb.org – 2 hits
- http://1337day.com, http://www.exploitdb.com – 1 hit
- http://nodesecurity.io/advisories – 4 hits

Such security big?

node.js sec: exploits anyone?

- http://seclists.org/bugtraq – 0 hits
- http://osvdb.org – 2 hits
- http://1337day.com, http://www.exploitdb.com – 1 hit
- http://nodesecurity.io/advisories – 4 hits

Such security big?

not exactly

node.js sec: what's wrong?

node.js security is a blank page

Sessions	NO
Permanent Data Storage	NO
Caching	NO
Database Access	NO
Logging	NO
Default Error Handling	NO
...	Most likely NO

http://www.slideshare.net/ASF-WS/asfws-2012-nodejs-security-old-vulnerabilities-in-new-dresses-par-sven-vetsch

node.js sec: how does sec look like?

node.js sec: how does sec look like?

node.js sec: exceptions / callbacks

callbacks Error object – remember to handle those

```
var fs = require("fs");
fs.readFile("/some/file", "utf8", function (err, contents) {
    // err will be null if no error occured

    // ... otherwise there will be info about error
});
```

forget about handling and die debugging

node.js sec: exceptions / eventemitter

EventEmitter: emitting events 4 async actions

```
var http = require("http");
http.get("http://nodejs.org/", function (res) {
    res.on("data", function (chunk) {
        do_something_with_chunk;
    });
    res.on("error", function (err) {
        // listener handling error
    });
});
```

Attach listeners to errors events or

welcome unhandled exception!

node.js sec: uncaught exceptions

- by default node.js will print stack trace and terminate thread

- EventEmitter / process / uncaughtException

```
// it looks like this by default:

process.on("uncaughtException", function (err) {
    console.error(err);
    console.trace();
    process.exit();
});
```

node.js sec: uncaught exceptions

- by default node.js will print stack trace and terminate thread

- EventEmitter / process / uncaughtException

```
// it looks like this by default:

process.on("uncaughtException", function (err) {
    console.error(err);
    console.trace();
    process.exit();
});
```

So do you <u>really</u> want to comment out the 'process.exit()' line?

node.js sec: clusters

scaling within multi-core envs

```
var cluster = require('cluster');
var http = require('http');
var numCPUs = require('os').cpus().length;
if (cluster.isMaster) {
  for (var i = 0; i < numCPUs; i++) {
    cluster.fork();
  }
  cluster.on('exit', function(worker, code, signal) {
    console.log(worker.process.pid + ' died');
  });
} else {
  http.createServer(function(req, res) {
    res.writeHead(200);
    res.end("hello world\n");
  }).listen(8000);
}
```

node.js sec: domains

Handling multiple different IO operations as a single group

```
// don't do that:

var d = require('domain').create();
d.on('error', function(er) {
  console.log('error, but oh well', er.message);
});
d.run(function() {
  require('http').createServer(function(req, res) {
    handleRequest(req, res);
  }).listen(PORT);
});
```

node.js sec: domains

Rather use cluster & forks and exit gently..:

```
create_cluster;
fork_workers;

if(worker){
    var domain = require('domain');
    var server = require('http').createServer(function(req, res) {
    var d = domain.create();
    d.on('error', function(er) {
      console.error('error', er.stack);
       try {
          // set timeout timer
          // update master && print err msg
          //  close server
       } catch (er2) { console.error('Error 500!', er2.stack); }
}
```

node.js sec: domains

Using Express take look at that:

https://github.com/brianc/node-domain-middleware

Assigning each Express request to a separate domain?

node.js sec: npm modules

- npm install (-g)
- who creates modules?
- who verifies those?
- how to update?
  - semantic versioning in package.json
  - "connect":"~1.8.7" -> 1.8.7 - 1.9

node.js sec: npm modules

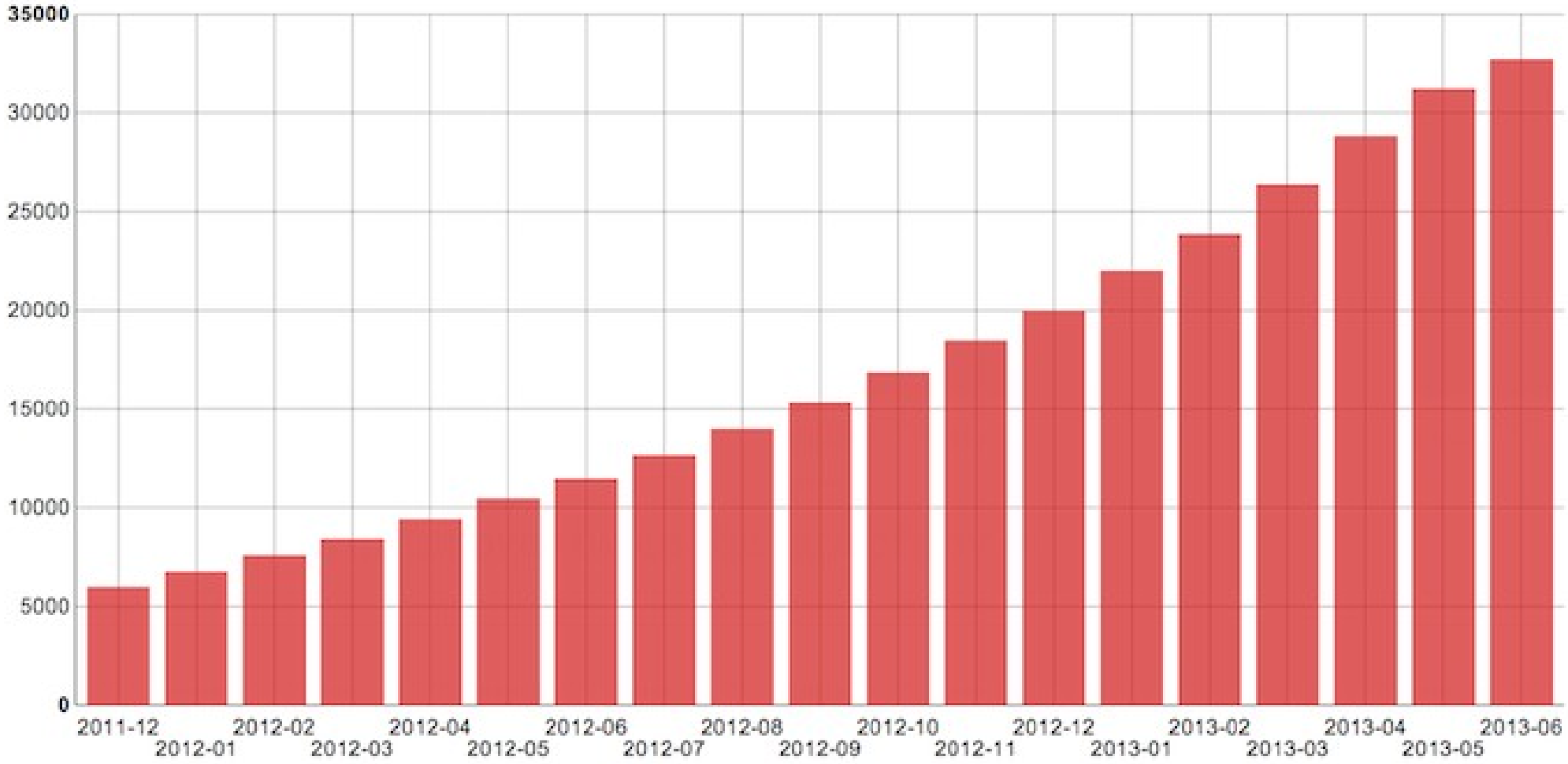
--ignore-scripts

stop preinstall/prepublish scripts

- mods auditing: https://nodesecurity.io/

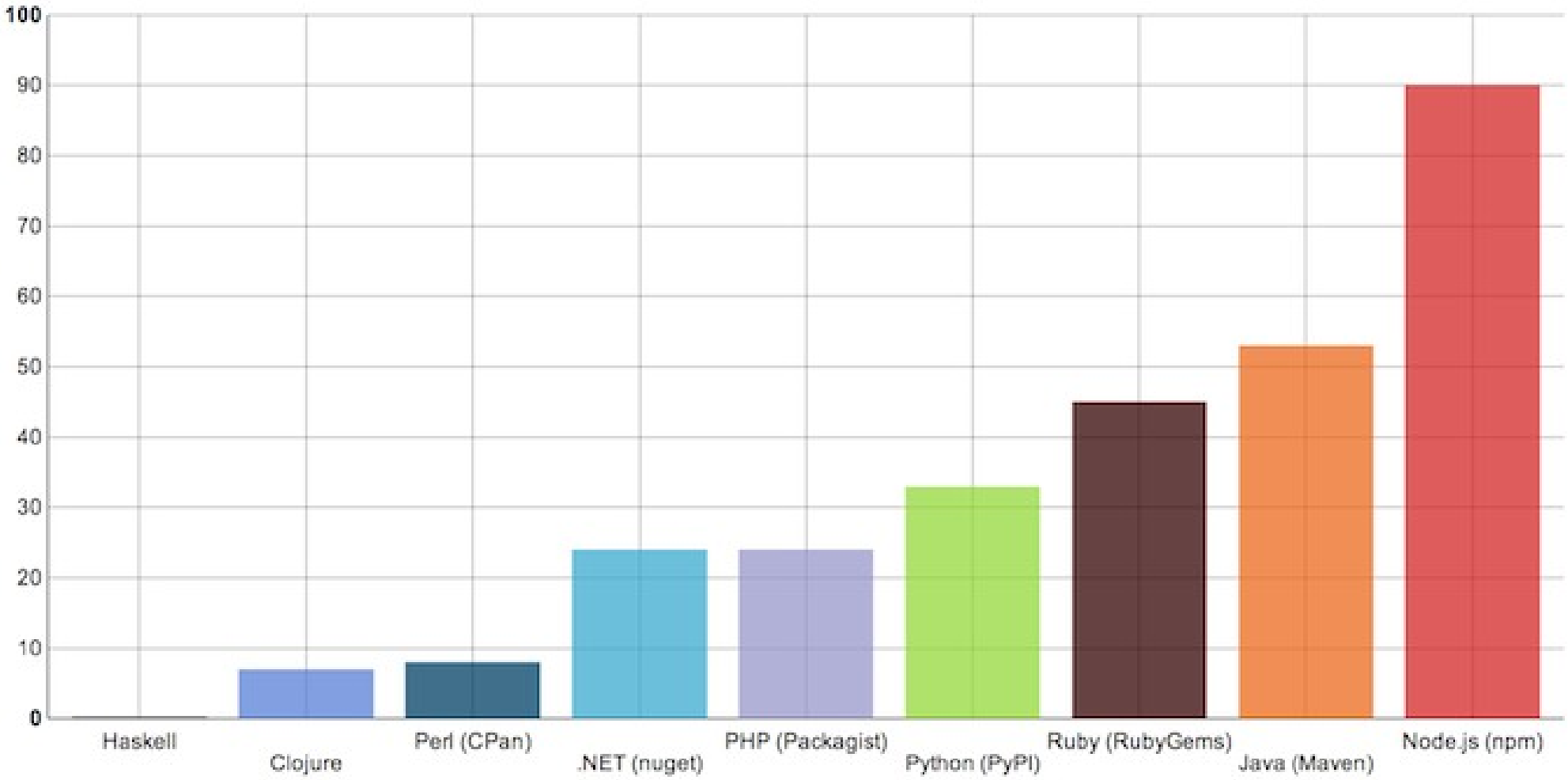
node.js sec: npm modules

The scale of npm modules



node.js sec: npm modules

Comparison to other langs (mods/day):



node.js sec: npm modules

Remember:

- use strict?

- static analysis?

- does include some test suite?

- what is the dependency tree?

node.js.express: connect / express

Express – web dev framework

Built on top of connect

node.js.express: auth.basic_auth

```
var express = require('express'),
app = express();
app.use(express.basicAuth("user", "pwd"));
app.get("/", function (req, res) {
    res.send('Hello World');
});
app.listen(8080);
```

Plain text and simple auth issues

node.js.express: auth.basic_auth

```
var users = {
    admin: "admin123",
    user: "user456"
};

app.use(express.basicAuth(function (user, pass) {
    return users.hasOwnProperty(user) && users[user]
=== pass;
}));
```

node.js.express: auth.SSL

```
var express = require('express'), routes = require('./routes'), fs = require('fs')
var opts = {
        key: fs.readFileSync('ssl/server/keys/server.key'),
        cert: fs.readFileSync('ssl/server/certificates/server.crt'),
        ca: fs.readFileSync('ssl/ca/ca.crt'),
        crl: fs.readFileSync('ssl/ca/ca.crl'),
        requestCert: true,
        rejectUnauthorized: true
        passphrase: "pwd" // <<<< really here?
};

var app = module.exports = express.createServer(opts);

app.configure(function(){
    app.set('views', __dirname + '/views');
    ...
});

app.get('/', routes.index);
app.listen(8443);
```

node.js.express: passport.js

- provides API for authentication and authorization
- authentication:
    - LocalStrategy
    - OpenIDStrategy
    - OAuth / FacebookStrategy

node.js.express: authorization

```
var users = [
    { id: 1, name: "user1", role: "admin" },
    { id: 2, name: "user2", role: "common" },
];
function loadUser(req, res, next) {
    req.userData = users[req.params.user];
    return next();
}
function requireRole(role) {
    return function (req, res, next) {
        if (req.user.role === role) {
        return next();
    } else {
        return next(new Error("Unauthorized"));
    }
};}
```

node.js.express: authorization

```
app.get("/users/:user", loadUser, function (req, res) {
    res.send(req.user.name);
});

app.del("/users/:user", requireRole("admin"), loadUser,
function (req,res) {
    res.send("User deleted");
});
```

node.js.express: logging

OWASP will tell you what should be logged :)

https://www.owasp.org/index.php/Logging_Cheat_Sheet

- authentication & authorisation
- session management
- errors & weirdo events
- events (startups, shutdowns, slowdowns etc)
- high risk functionalities (payments, privileges, admins)

node.js.express: logging

Try Winston module (Github -> flatiron/winston)

- logging to console
- logging to file
- sending logs over HTTP
- CouchDB, Redis, MongoDB, Riak etc

node.js.express: logging

```
var winston = require('winston');
var logger = new (winston.Logger)({
    transports: [
        new (winston.transports.Console)(),
        new (winston.transports.File)({
            filename: 'application.log' })
]
});
```

node.js.express: sessions

```
var express = require('express');
var app = express();
var RedisStore = require('connect-redis')(express);

app.use(express.cookieParser());
app.use(express.session({
  store: new RedisStore({
    host: '127.0.0.2',
    port: 6379,
    db: 3,
    pass: 'pwd'
  }),
  secret: 'this-is-very-secret'
}));

app.get('/somewhere', function(req, res) {
  res.send('In the middle of nowhere');
});

app.listen(process.env.PORT || 8080);
```

node.js.express: sessions

```
app.use(express.session({
    secret: “very-secret”,
    key: “sessionId”,
    cookie: {
            httpOnly: true,
            secure: true
    }
}));
```

node.js.CSI: CSRF

```
var express = require("express"),
app = express();
app.use(express.cookieParser());
app.use(express.bodyParser());
app.use(express.session({ secret: "very-secret" }));
app.use(express.csrf());
```

node.js.CSI: CSRF

```
app.get("/valid", function (req, res) {
    <input type="hidden" name="_csrf" value="+req.CsrfToken()+">
    res.send(output);
});
app.get("/invalid", function (req, res) {
    no_csrf_token_field
    res.send(output);
});
```

Express CSRF ignores CSRF on HTTP, GET, OPTIONS, HEAD reqs

node.js.CSI: input.validation

```
var express = require("express"),
app = module.exports = express();
app.use(express.bodyParser());
app.use(require("express-validator")());

app.get("/", function (req, res) {
    res.sendfile(__dirname + "/tpls/validate.html");
});

app.post("/", function (req, res, next) {
    // validation
    req.checkBody("name").notEmpty().is(/\w+/);
    // filtering
    req.sanitize("name").trim();
    ...
}
```

node.js.CSI: XSS

- XSS allows to access cookies, session tokens etc
- or even redirect user to malicious sites
- Myth: frameworks / tpls does the anti-XSS job
- always encode untrusted data for correct context
- OWASP ESAPI

node.js.CSI: DoS

- Error handling
- Use streams / chunking
- Use monitoring
- Use domains / clusters
- Don't be afraid of SlowLoris :)

node.js.CSI: ReDoS

- Regex could take exponential execution time
- Is regex executed in the event loop thread?
- Regex and user input?

Example: Commonly used URL validator regex

# of Input Characters	Execution Time
30	6 sec
35	3min
36	6 min
37	13 min
38	25 min
39	1hr 28 min
40	3 hr 46 min

https://speakerdeck.com/ckarande/top-overlooked-security-threats-to-node-dot-js-web-applications

node.js.CSI: HPP

HTTP Parameter Pollution

```
// POST firstName=John&firstName=John

req.body.color

//=> [“John”, “John”]
```

node.js.CSI: HPP

Modify app behavior:

DB Shell

```
> db.users.find({userName:"p"}).pretty()
{
        "_id" : ObjectId("53092b495ad132dfd56dbf70"),
        "address" : "",
        "dob" : "",
        "firstName" : [
                "John",
                "John"
        ],
        "lastName" : "Doe",
        "password" : "$2a$10$iZBDC45NulBco7s2GhCkJ.jU1Kr
        "ssn" : "1234",
        "userId" : 39,
        "userName" : "p"
}
```

node.js.CSI: HPP

- TypeErrors, uncaught errs->DoS,
- Check the expected type in the input validation
- Input fuzzing / test suites
- try/catch, domains, clusters

node.js.CSI: HTTP_headers.CSP

Content Security Policy

```
Content-Security-Policy:
    script-src 'self';
    frame-src 'none';
    object-src 'none';
    report-uri /my_csp_report_parser;
```

node.js.CSI: HTTP_headers.CSP

Content Security Policy

```
Content-Security-Policy:
    script-src 'self';
    frame-src 'none';
    object-src 'none';
    report-uri /my_csp_report_parser;
```

```
app.use(helmet.csp.policy({
    defaultPolicy: {
        "script-src": [ "'self'" ],
        "img-src": [ "'self'", "http://example.com/" ]
    }
}));
```

node.js.CSI: HTTP_headers.HSTS

HTTP Strict Transport Security

```
app.use(helmet.hsts(X, true));
```

```
Strict-Transport-Security: max-age=X; includeSubdomains
```

Requires HTTPS also on subdomains,

respects configuration for X seconds

node.js.CSI: HTTP_headers.X-Frame-Options

```
X-Frame-Options: DENY
helmet.xframe('sameorigin');
helmet.xframe('allow-from', 'http://example.com');
```

node.js.CSI: HTTP_headers.others

- X-Content-Type-Options
- Cache-Control
- X-Powered-By

```
X-Content-Type-Options: nosniff
app.use(helmet.contentTypeOptions());
app.use(helmet.cacheControl());
app.disable("x-powered-by");
```

node.js.CSI: request_size

Just set limits

use streams instead of buffering

And request size:

```
app.use(express.limit("5mb"));
```

node.js.CSI: environment.monitoring

- is app functional? :)
- is app overloaded?
- app should provide monitoring interface
- how many errors caught?
- are forks alive and OK?

node.js.CSI: environment.resources

- node.js will eat ;)
- use control groups || containers
- monitor resources usage

node.js.CSI: environment.sandboxing

- running node.js within shared env?
- selinux sandbox?
- libvirt sandbox?
- LXC / Docker?

node.js.CSI: environment.ACLs

Just...

node.js.CSI: environment.ACLs

Just...

Don't run as `root`!!!

node.js.CSI: environment.ACLs

Just...

Don't run as `root`!!!

And also maybe behind some LB?

node.js.CSI: environment.tracing_execution

- SmartOS / Joyent: debugging
- Bunyan / Dtrace
- strace of course...

node.js.testing: testing

- maybe some interface for white-box pentests?
- unit-testing 4 the sake!
- OWASP Zed Attack Proxy

So what else?

sockets.io

node-webkit

node.js.learning

- Node Security Book
- OWASP Node Goat (top10)
- nodesecurity.io (Twitter, RSS)

node.js.hiring?

- does this guy contribute?
- NoSQL somehow?
- not only HTTP: socket.io?
- DevOps & infra?
- Security? :)

Infosec && meet.js meetups

Thank you :)

node.js security

Maciej Lasyk

SEConference

Kraków, 2014-05-09

http://maciek.lasyk.info/sysop

maciek@lasyk.info

@docent-net