OWASP
AppSec Europe
London 2nd-6th July 2018

Patterns in Node Package Vulnerabilities

Chetan

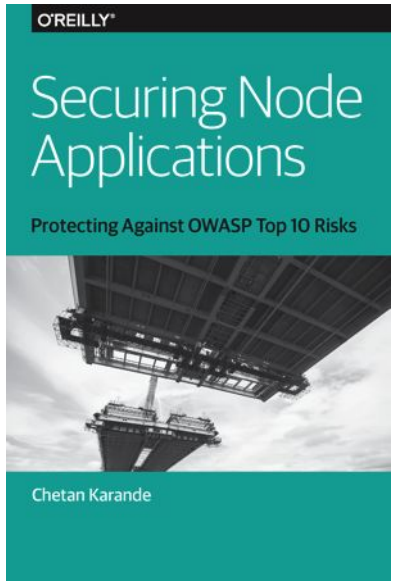
JSON.stringify(me);

{

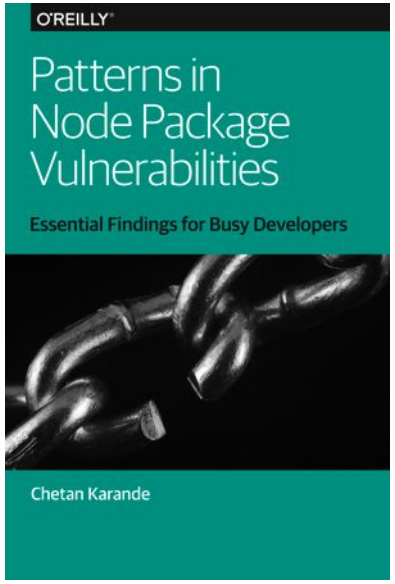
“Principal Software Engineer”: “Depository Trust & Clearing Corporation (DTCC)”,

“Project Leader”: “OWASP NodeGoat Project”,

“Author”: [ ]



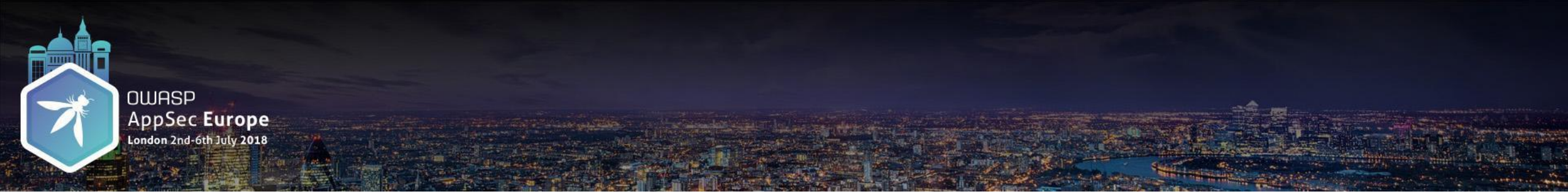
,



}

532 packages/day

~ 700,000 packages

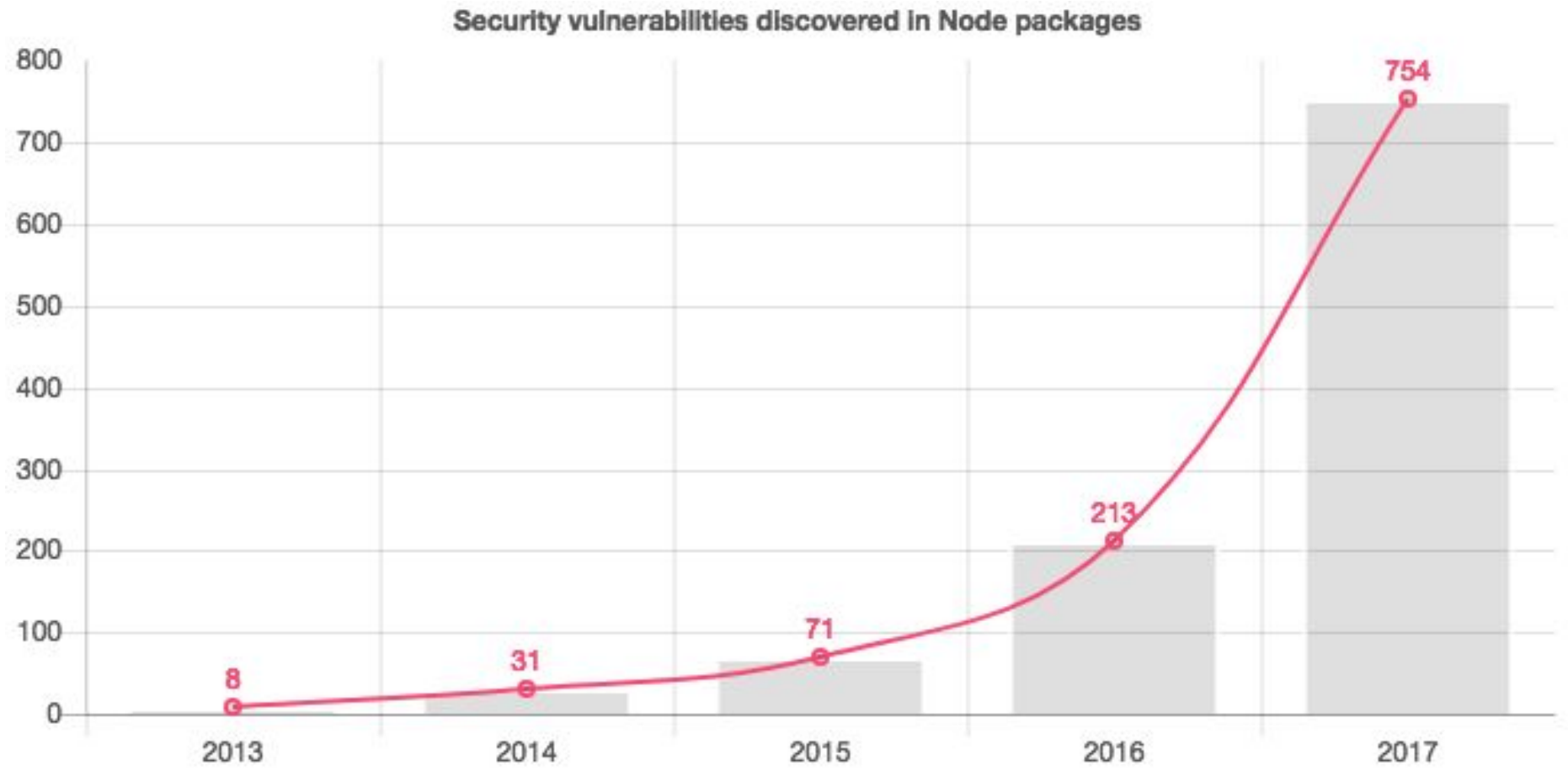


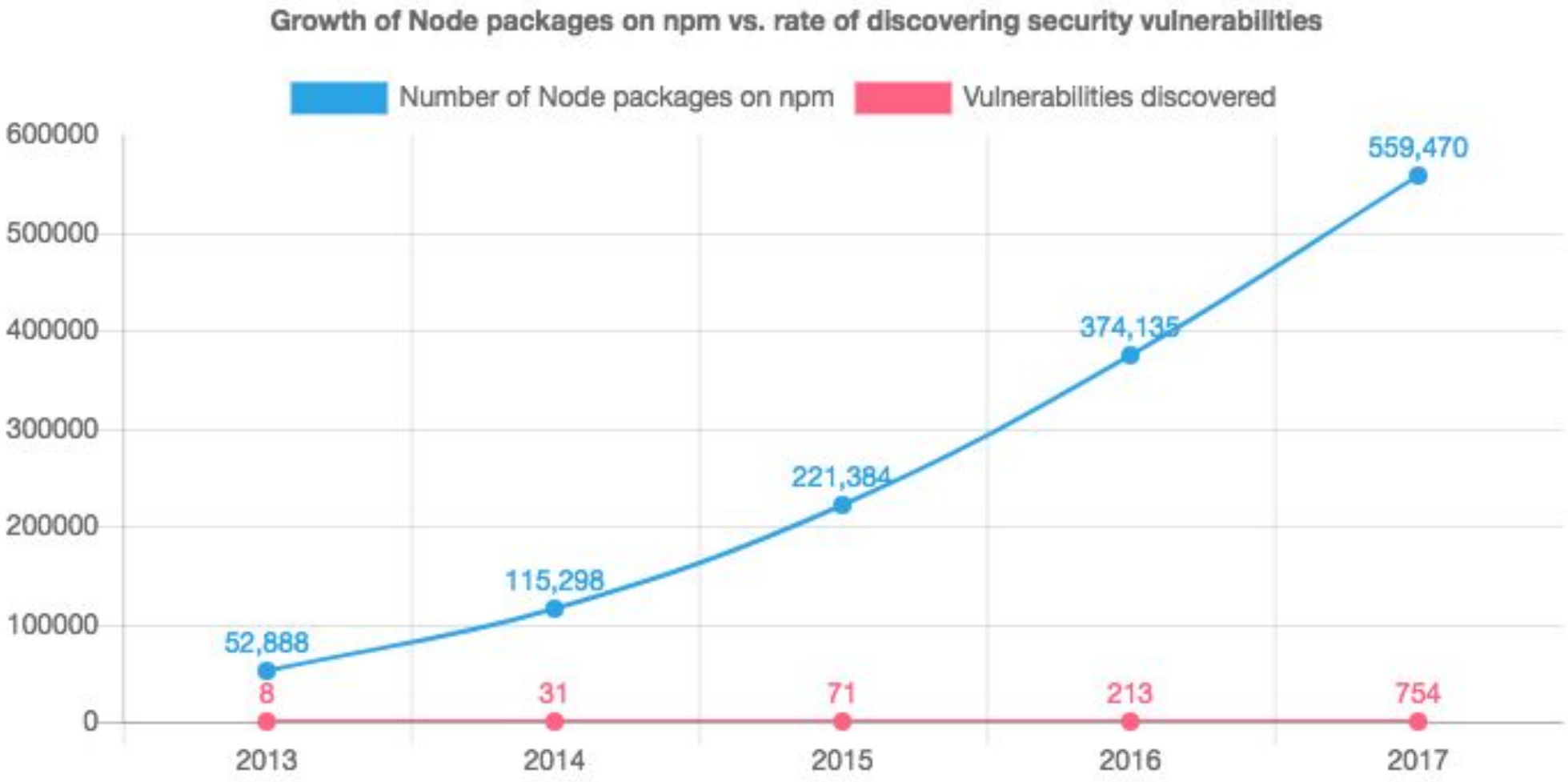
hackerone

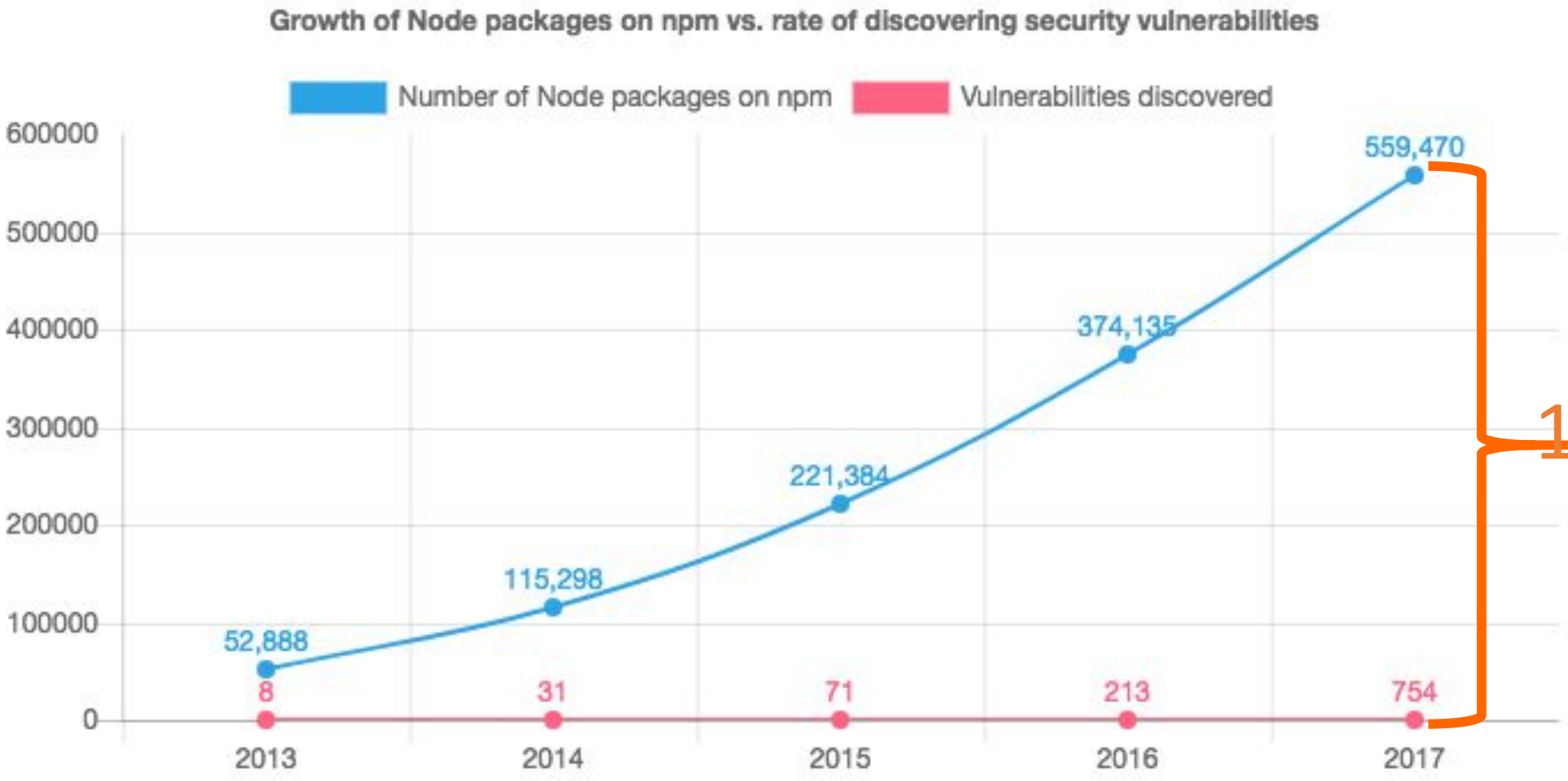
88 Disclosures

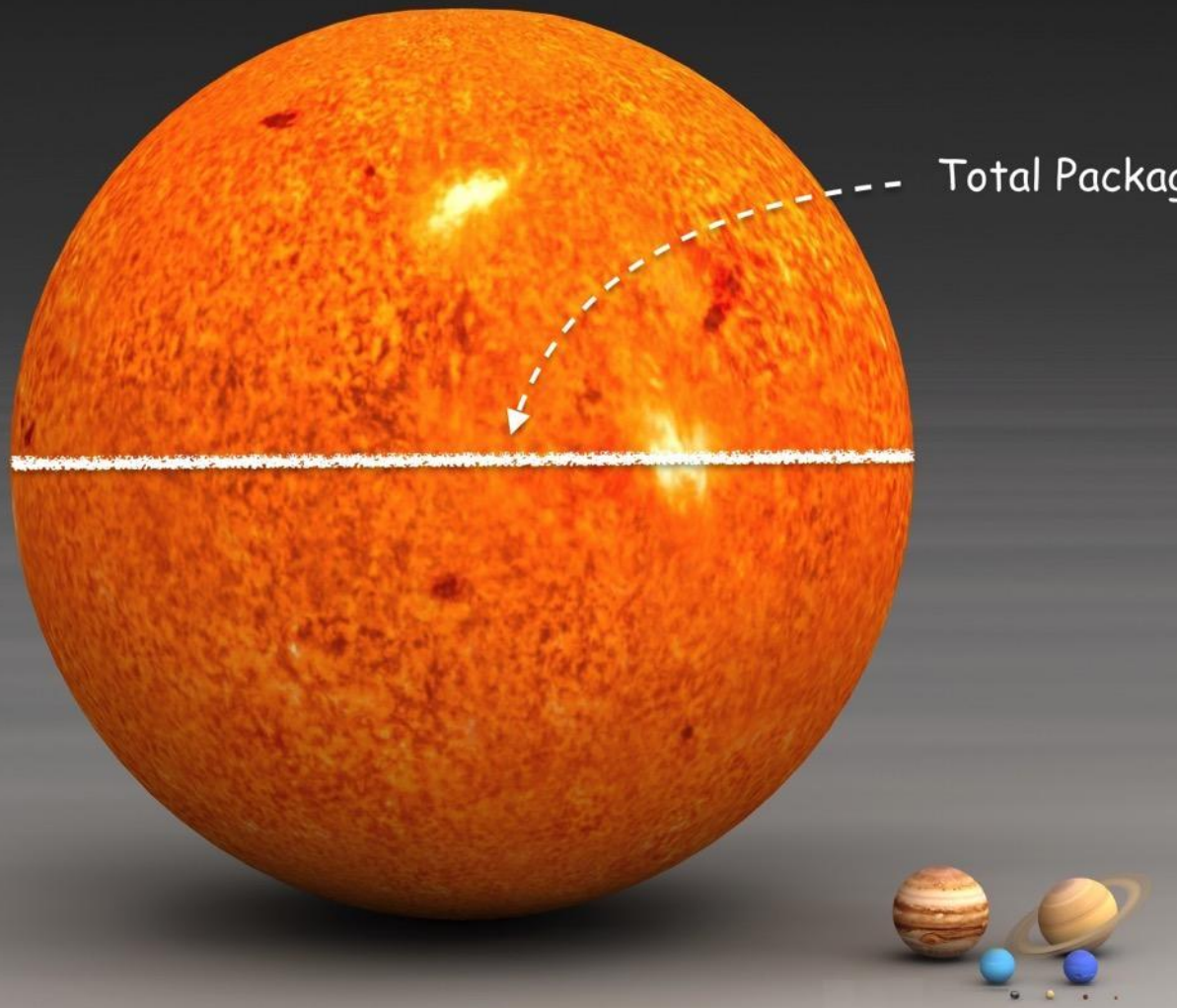
603 Vulnerabilities

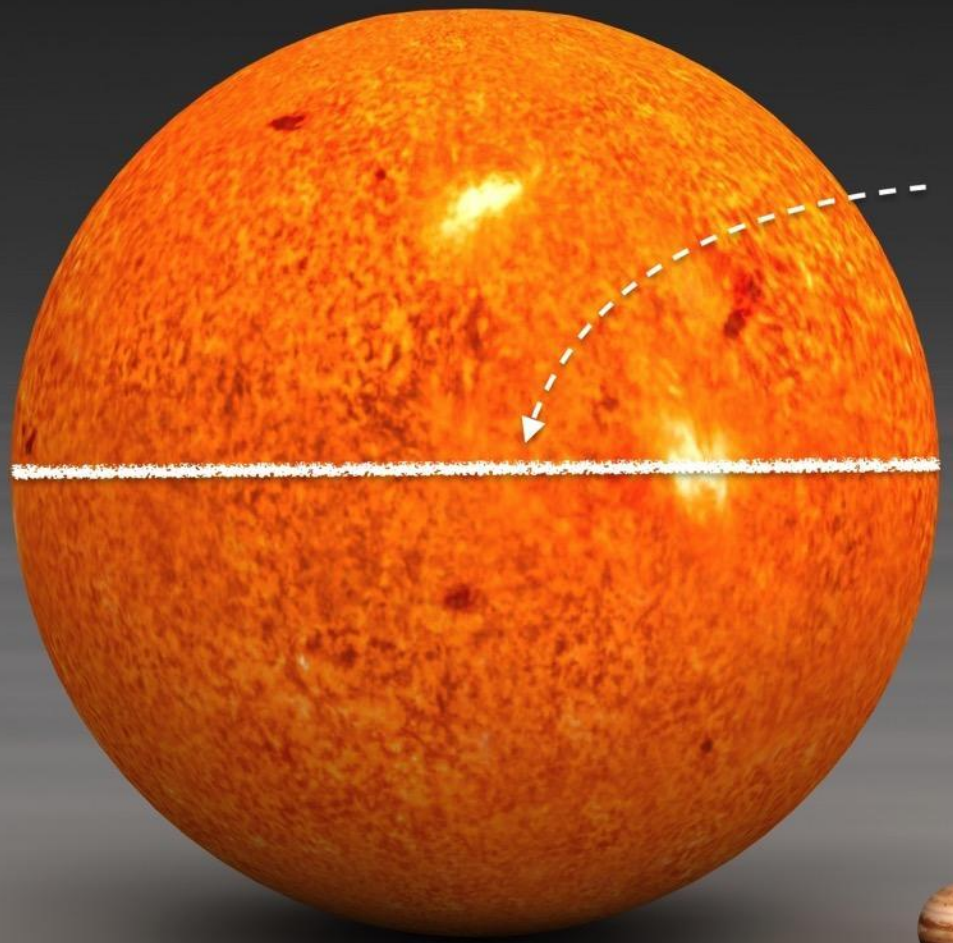
1,098 Advisories


Security vulnerabilities discovered in Node packages
800
700
600
500
400
300
200
100
0
8
31
71
213
754
2013
2014
2015
2016
2017


Growth of Node packages on npm vs. rate of discovering security vulnerabilities
Number of Node packages on npm
Vulnerabilities discovered
600000
500000
400000
300000
200000
100000
0
52,888
115,298
221,384
374,135
559,470
8
31
71
213
754
2013
2014
2015
2016
2017


Growth of Node packages on npm vs. rate of discovering security vulnerabilities
Number of Node packages on npm
Vulnerabilities discovered
600000
500000
400000
300000
200000
100000
0
52,888
115,298
221,384
374,135
559,470
8
31
71
213
754
2013
2014
2015
2016
2017
1 : 600

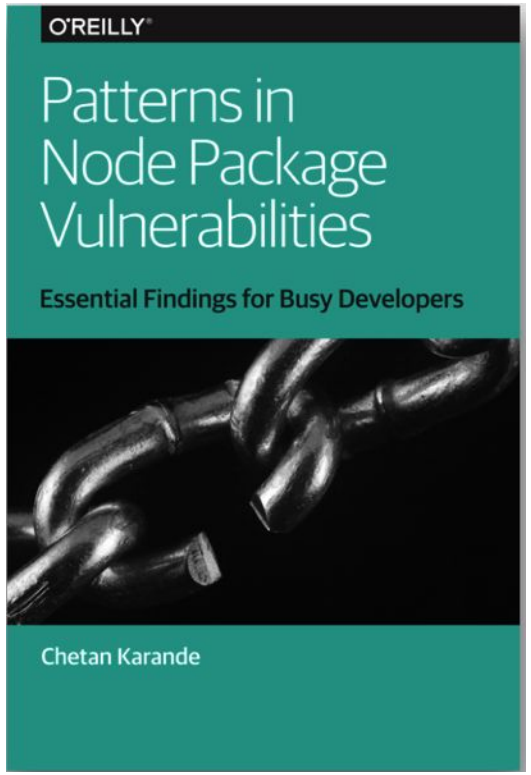

Total Packages on npm

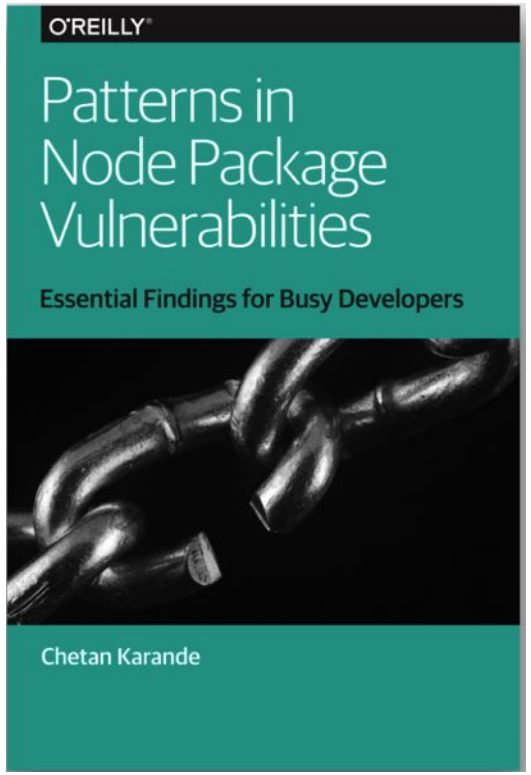

Total Packages on npm
Known npm Package Vulnerabilities
(1/6th of the Earth's Diameter)


=== npm audit security report ===
npm install chokidar@2.0.3 to resolve 1 vulnerability
ARNING: Recommended action is a potentially breaking change
npm audit
ency of chokidar
chokidar > fsevents > node-pre-gyp > rc > deep-extend
nfo https://nodesecurity.io/adv
$ snyk test
✗ High severity vulnerability found on minimatch@0.3.0
- desc: Regular Expression Denial of Service
- info: https://snyk.io/vuln/npm:minimatch:20160620
- from: ionic@2.1.17 > gulp@3.8.8 > liftoff@0.12.1 > findup-sync@0.1.3 > glob@3.2.11 > minimatch@0.3.0
Upgrade direct dependency gulp@3.8.8 to gulp@3.8.11 (triggers upgrades to liftoff@2.2.0 > findup-sync@0.
✗ Medium severity vulnerabil
- desc: Regular Expression D
- info: https://snyk.io/vuln
- from: ionic@2.1.17 > momen
Upgrade direct dependency moment@2.11.1 to moment@2.15.2
Snyk CLI
✗ Medium severity vulnerability found on send@0.10.1
- desc: Root Path Disclosure
- info: https://snyk.io/vuln/npm:send:20151103
- from: ionic@2.1.17 > serve-static@1.7.1 > send@0.10.1
Upgrade direct dependency serve-static@1.7.1 to serve-static@1.8.1 (triggers upgrades to send@0.11.1)

**By seeking and blundering
we learn.**

- Johann Wolfgang von Goethe


O'REILLY
Patterns in Node Package Vulnerabilities
Essential Findings for Busy Developers
Chetan Karande

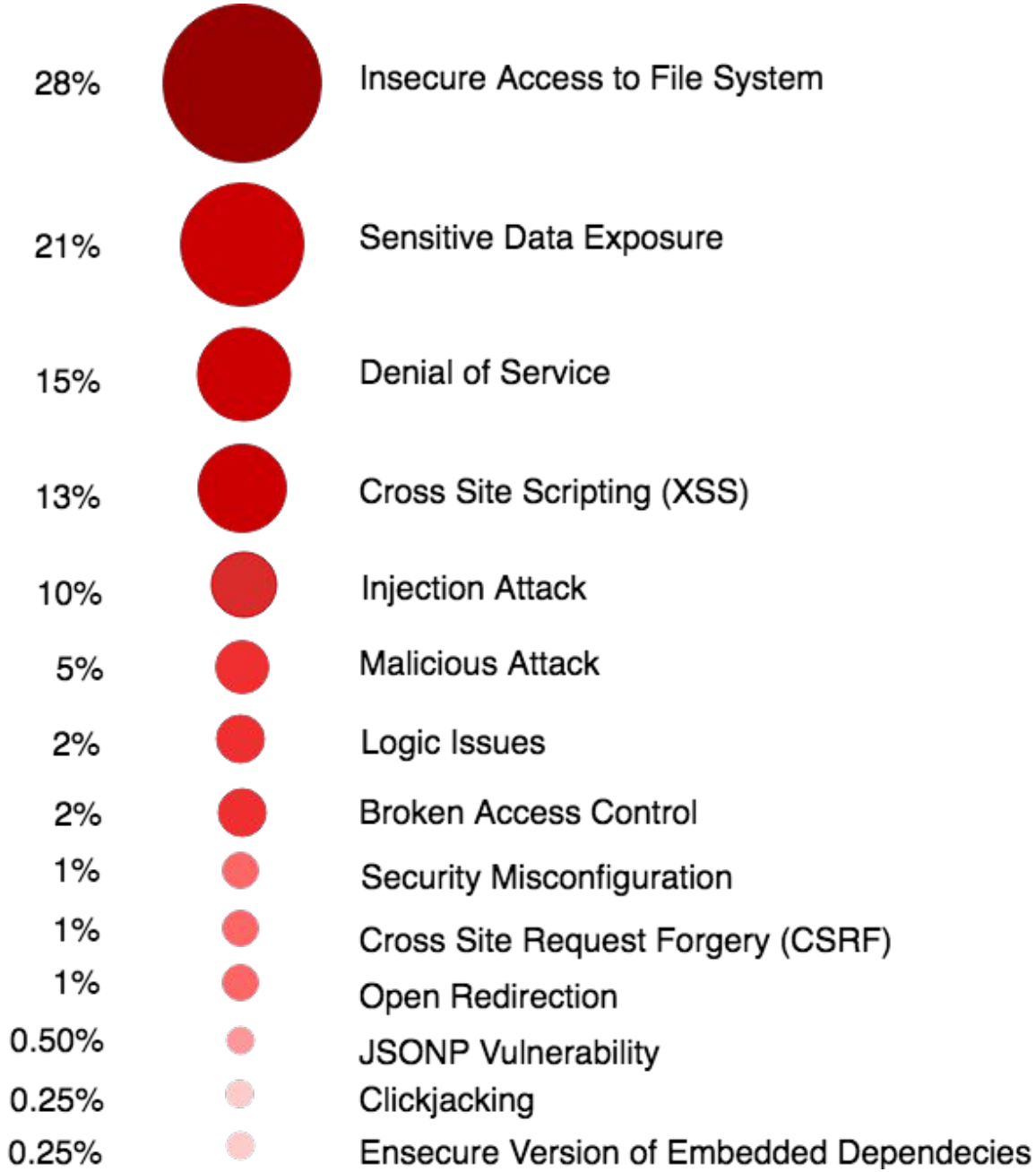


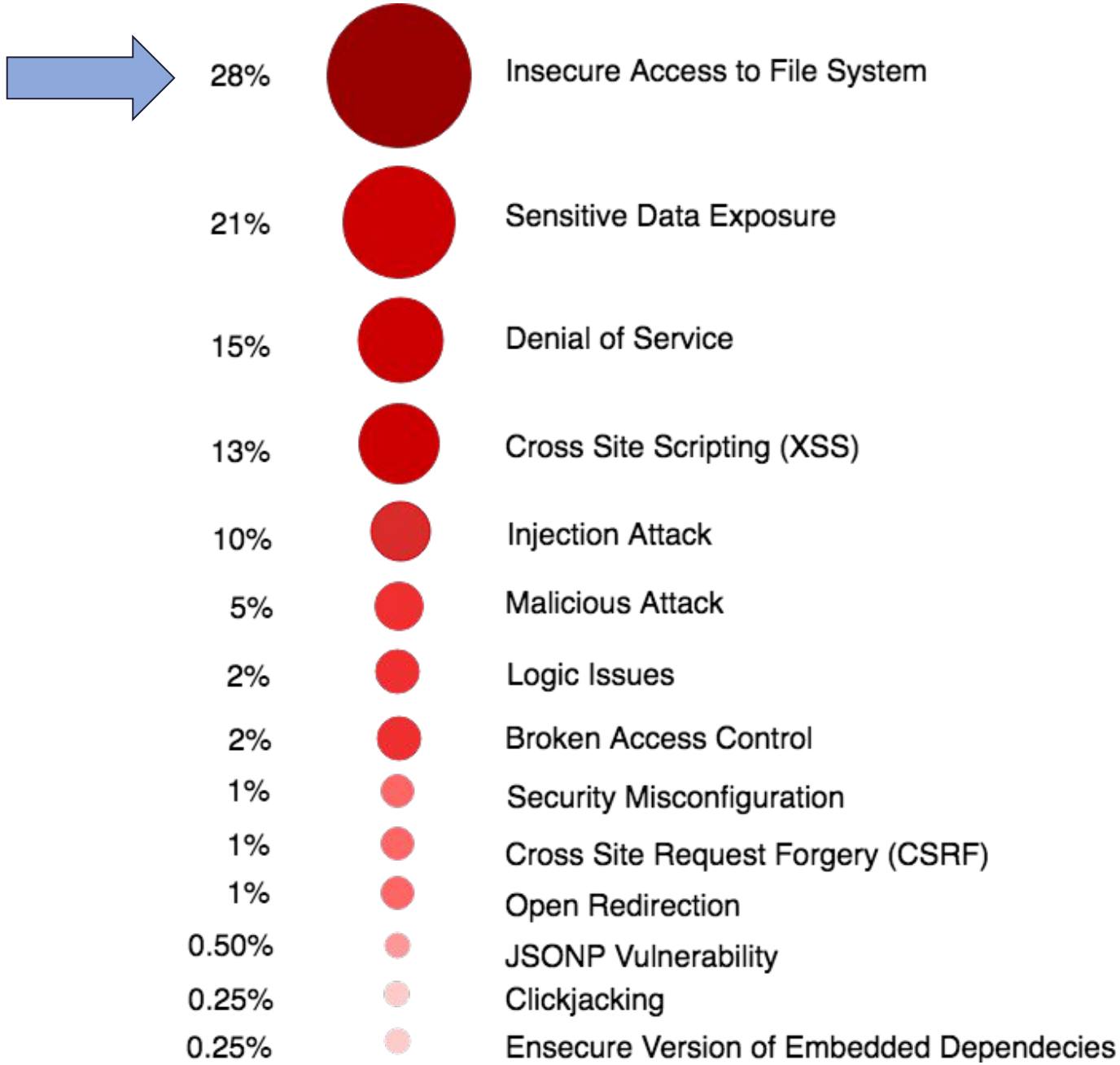
+

1,084

528

1,023 Unique Advisories


28%
Insecure Access to File System
21%
Sensitive Data Exposure
15%
Denial of Service
13%
Cross Site Scripting (XSS)
10%
Injection Attack
5%
Malicious Attack
2%
Logic Issues
2%
Broken Access Control
1%
Security Misconfiguration
1%
Cross Site Request Forgery (CSRF)
1%
Open Redirection
0.50%
JSONP Vulnerability
0.25%
Clickjacking
0.25%
Ensecure Version of Embedded Dependecies


28%
Insecure Access to File System
21%
Sensitive Data Exposure
15%
Denial of Service
13%
Cross Site Scripting (XSS)
10%
Injection Attack
5%
Malicious Attack
2%
Logic Issues
2%
Broken Access Control
1%
Security Misconfiguration
1%
Cross Site Request Forgery (CSRF)
1%
Open Redirection
0.50%
JSONP Vulnerability
0.25%
Clickjacking
0.25%
Ensecure Version of Embedded Dependecies

Insecure Access to File System

Pattern #1 Directory Traversal

The npm Blog

Blog about npm things.

Newly Paranoid Maintainers

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.

The files that could have been potentially accessed included a ton of sensitive information: SSL keys, database passwords with read/write access to our production databases, basically everything you never want a third party to see. Somebody with access to the database could replace npm modules with malicious payloads. I don’t want to blur the truth here: this could have been a disaster. It is of very

The npm Blog

Blog about npm things.

Newly Paranoid Maintainers

Caused by an insecure dependency vulnerable to Directory Traversal

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.

The files that could have been potentially accessed included a ton of sensitive information: SSL keys, database passwords with read/write access to our production databases, basically everything you never want a third party to see. Somebody with access to the database could replace npm modules with malicious payloads. I don't want to blur the truth here: this could have been a disaster. It is of very

Directory Traversal
Common Coding Mistakes

Missing or insufficient user input validation for path traversal characters before using it in a URL to serve contents on the server.

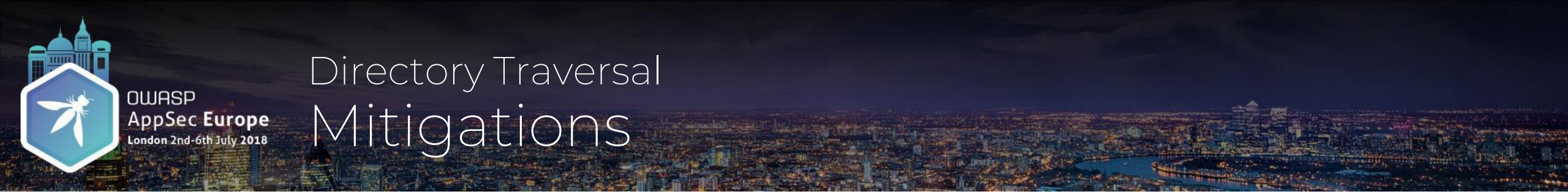
Directory Traversal
Common Coding Mistakes

Missing or insufficient user input validation for path traversal characters before using it in a URL to serve contents on the server.

- /
- ../
- %2f
- %2e%2e/
- %2e%2e%2f

Directory Traversal
Common Coding Mistakes

```
1  const http = require('http');
2  const fs = require('fs');
3  const path = require('path');
4  http.createServer(function (req, res) {
5
6   // Get resource path from the user input
7   let userInput = req.url;
8
9   // Prevent serving files outside public folder
10  let fullPath = (path.join(__dirname, 'public', userInput));
11
12  // Open a file on the server and return its content
13  fs.readFile(fullPath, function (err, data) {⟺});
19 }).listen(8080);
```



Directory Traversal
Mitigations

✓ If the path needs to be supplied from the user input, sanitize the input to remove path traversal characters (./ and ../ as well as encoded variations)

Insecure Access to File System

Pattern # 2 Symlink Attack /Arbitrary File Writ

Symlink Attack

Symlink Attack

Application sharing the
host server with external users

Symlink Attack

Application sharing the host server with external users

Shared folders

Symlink Attack

A malicious user sharing the host, could exploit this vulnerability to:

Symlink Attack

A malicious user sharing the host, could exploit this vulnerability to:

Corrupt or destroy vital system or application files to which only the target application has the access.

Symlink Attack
Common Coding Mistakes

Using predictable file or folder names when writing to shared directories on a host server shared with external users.

Arbitrary File Write

Vulnerable: <1.0.0

Patched: >=1.0.0

Module: cli

Published: June 15th, 2016

Reported by: Steve Kemp

CVE-NONE

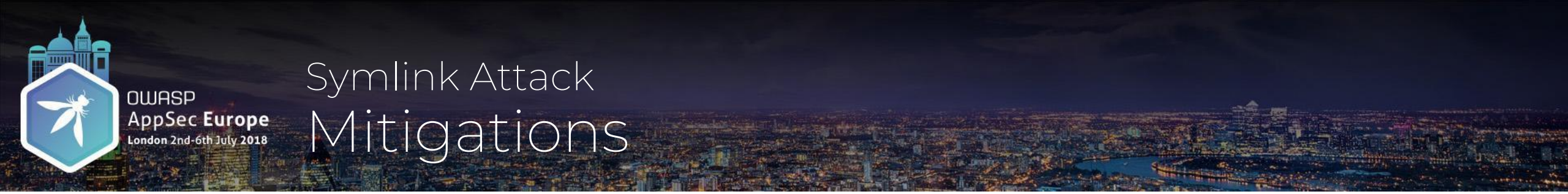
CWE-22

Example: The package writing logs to the shared /tmp directory with a predictable file name

Overview

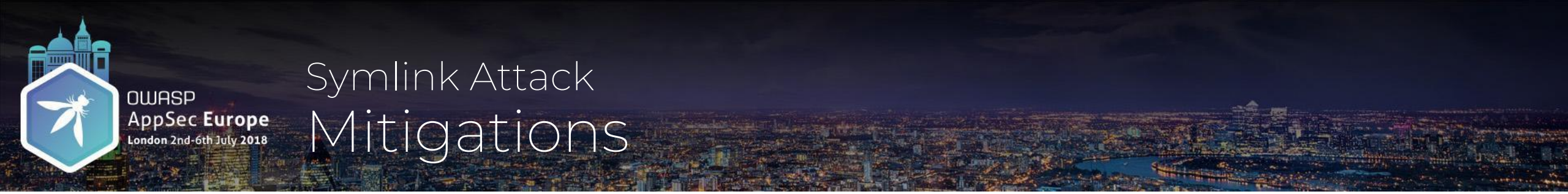
Affected versions of `cli` use predictable temporary file names. If an attacker can create a symbolic link at the location of one of these temporarly file names, the attacker can arbitrarily write to any file that the user which owns the `cli` process has permission to write to.

> ln –s <source file> <target file>



Symlink Attack
Mitigations

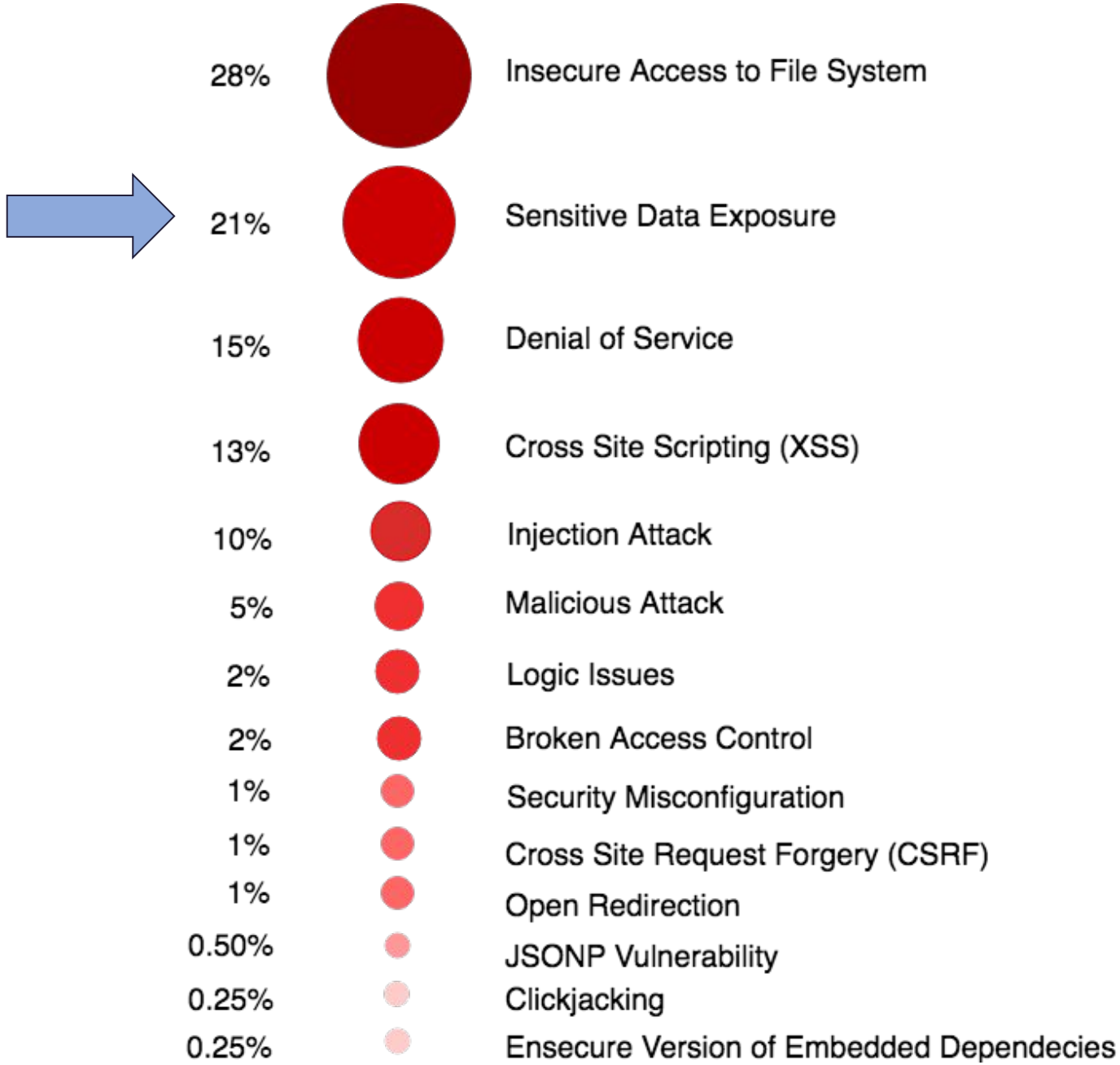
✓ Avoid using shared system folders.



Symlink Attack
Mitigations

✓ Avoid using shared system folders.

✓ If you have to use a shared folder for writing non-sensitive data, use crypto module's *randomBytes* method to generate random filenames.


28%
Insecure Access to File System
21%
Sensitive Data Exposure
15%
Denial of Service
13%
Cross Site Scripting (XSS)
10%
Injection Attack
5%
Malicious Attack
2%
Logic Issues
2%
Broken Access Control
1%
Security Misconfiguration
1%
Cross Site Request Forgery (CSRF)
1%
Open Redirection
0.50%
JSONP Vulnerability
0.25%
Clickjacking
0.25%
Ensecure Version of Embedded Dependecies

The more you leave out,

the more you highlight

what you leave in.

- Henry Green

Sensitive Data Exposure

Pattern # 3 Leaking Application Secrets

Leaking Application Secrets
Common Coding Mistakes

Application-specific secrets appearing at insecure places such as as:

- code repositories,
- log files,
- client-side storage,
- URLs,
- application global namespace

snyk Test Vulnerability DB Docs Blog Features

Vulnerability DB › npm

Man-in-the-Middle (MitM)

Affecting **hotel** package, **ALL** versions

Example: Leaking the SSL private key in the code repository

Overview

hotel is local domains for everyone. Affected versions of the package are vulnerable to Man-in-the-Middle (MitM) attacks. Hotel contained a self-signed certificate built-in, the private key being in the repo. This allows any user to use that key and listen in on the traffic.

Remediation

Vulnerability DB › npm

Credentials saved as clear-text in log

Affecting **grunt-gh-pages** package, versions **<=0.9.1**

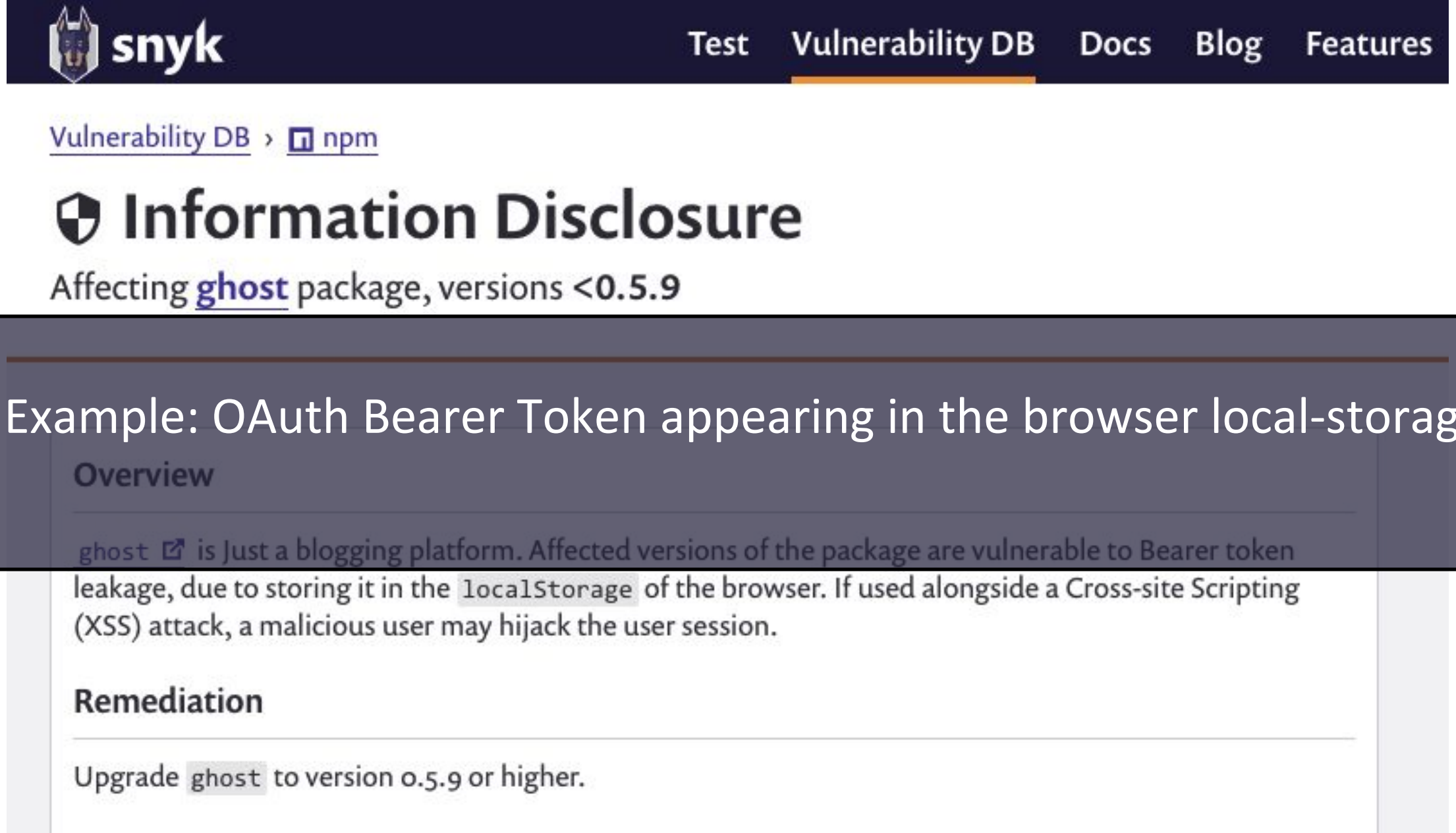
Example: URLs with authentication tokens appearing in the logs

Overview

grunt-gh-pages writes the repository url to log without redacting the github authentication token. The token can be compromised if the logs become publicly available.

Remediation

Upgrade to version 1.0.0 or greater and consider revoking previously used credentials with the module.


snyk
Test
Vulnerability DB
Docs
Blog
Features
Vulnerability DB › npm
Information Disclosure
Affecting ghost package, versions <0.5.9
Example: OAuth Bearer Token appearing in the browser local-storage
Overview
ghost is Just a blogging platform. Affected versions of the package are vulnerable to Bearer token leakage, due to storing it in the localStorage of the browser. If used alongside a Cross-site Scripting (XSS) attack, a malicious user may hijack the user session.
Remediation
Upgrade ghost to version 0.5.9 or higher.

Leaking Application Secrets
Mitigations

✓ Securely store applications secrets in Hardware Security Module (HSM) or Key Management Services.

Leaking Application Secrets
Mitigations

✓ Securely store applications secrets in Hardware Security Module (HSM) or Key Management Services.

✓ Mask any sensitive data before it appears in the log files.

Leaking Application Secrets
Mitigations

✓ Securely store applications secrets in Hardware Security Module (HSM) or Key Management Services

✓ Mask any sensitive data before it appears in the log files

✓ To reduce impact of a leak, use short-lived tokens.

Sensitive Data Exposure

Predictable Secrets

Sensitive Data Exposure

Predictable Secrets

Pattern # 4 Insecure Randomness

Insecure Randomness
Common Coding Mistakes

- Using Math.random() method is to generate random values in a security-sensitive context (random tokens, resource IDs, or UUIDs).

- Math.random() is cryptographically insecure. It can produce predictable values.

Insecure Entropy Source - Math.random()

Vulnerable: <1.4.4

Patched: >=1.4.4

Module: node-uuid

Published: March 28th, 2016

Reported by: Fedot Praslov

CVE-2015-885

CWE-330

Example: Using Math.random() to generate UUID

Overview

Affected versions of `node-uuid` consistently fall back to using `Math.random` as an entropy source instead of `crypto`,

Vulnerability DB › npm

Insecure Randomness

Affecting socket.io package, versions <0.9.7

Example: Using Math.random() to generate Socket IDs

Overview

socket.io is a node.js realtime framework server. Affected versions of the package are vulnerable to Insecure Randomness due to the cryptographically insecure `Math.random` function which can produce predictable values and should not be used in security-sensitive context.

Remediation

Upgrade `socket.io` to version 0.9.7 or higher.

Insecure Randomness
Mitigations

✓ Use *crypto* module to generate random numbers instead of Math.random()

Insecure Randomness
Mitigations

```
1  const crypto = require('crypto');
2  crypto.randomBytes(256, (err, buf) => {⟷});
```

Insecure Randomness
Mitigations

```
1  const crypto = require('crypto');
2  crypto.randomBytes(256, (err, buf) => {
3    if (err) throw err;
4    // use the generated random value
5    console.log(`${buf.length} bytes of random data: ${buf.toString('hex')}`);
6  });
```

Sensitive Data Exposure

Predictable Secrets

Pattern # 5 Non-constant Time Comparison

Non-constant Time Comparison
Common Coding Mistakes

Using fail-fast comparison logic to match user inputs against sensitive values.

Non-constant Time Comparison
Common Coding Mistakes

Using fail-fast comparison logic to match user inputs against sensitive values.

Example: JavaScript native string comparison operators (=== , ==)

Node Security Platform

REPORT VULNERABILITY ADVISORIES LOGIN

Non-Constant Time String Comparison

Vulnerable: <=0.1.1

Patched: >=0.1.2

CVSS 4.3 Medium

Module: csrf-lite

Published: April 22nd, 2016

Reported by: Todd Wolfson

CWE-208

Example: Using Fail Fast operators to compare csrf tokens

Overview

Affected versions of `csrf-lite` are vulnerable to timing attacks as a result of testing CSRF tokens via a fail-early comparison instead of a constant-time comparison.

snyk Test Vulnerability DB Docs Blog Features

Vulnerability DB › npm

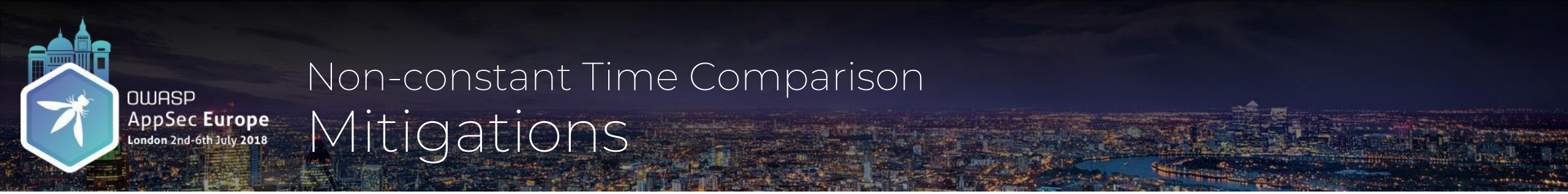
Timing Attack

Affecting **node-forge** package, versions **<0.6.33**

Example: Using a Fail Fast iterator to compare byte arrays

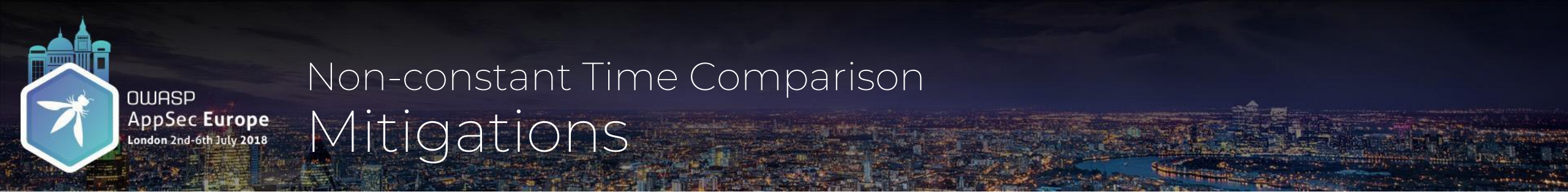
Overview

node-forge is a JavaScript implementation of network transports, cryptography, ciphers, PKI, message digests, and various utilities. Affected versions of the package are vulnerable to a Timing Attack due to unsafe HMAC comparison. The HMAC algorithm produces a keyed message by pairing a hash function with a cryptographic key. Both the key and a message serve as input to this algorithm, while it outputs a fixed-length digest output which can be sent with the message. Anyone who knows the key can repeat the algorithm and compare their calculated HMAC with one they have received, to verify a message originated from someone with knowledge of the key and has not been tampered with.



Non-constant Time Comparison
Mitigations

✓ Use a constant-time comparison logic that takes the same amount of time regardless of the input values.



Non-constant Time Comparison
Mitigations

✓ Use a constant-time comparison logic that takes the same amount of time regardless of the input values.

```
1  function constantTimeEquals(strA, strB) {
2   if (strA.length !== strB.length) {
3    return false;
4   } else {
5    let equal = 0;
6    for (let i = 0; i < strA.length; i++) {
7     equal |= strA.charAt(i) ^ strB.charAt(i);
8    }
9    return equal === 0;
10  }
11 }
```

Sensitive Data Exposure

Pattern # 6 Remote Memory Exposure

Remote Memory Exposure
Common Coding Mistakes

- Prior to Node.js 8, the Buffer constructor that takes a number as an argument, generates a Buffer instance with uninitialized underlying memory.

```
// Uninitialized Buffer of length 1000
var buffer = new Buffer(1000);
```

- The contents of a newly created Buffer remain unknown and might contain sensitive data.

Remote Memory Exposure

Affecting mongoose package, versions <3.8.39 >=3.5.5 || <4.3.6 >=4.0.0

Overview

Example: Using unsafe Buffer constructor

A potential memory disclosure vulnerability exists in mongoose. A `Buffer` field in a MongoDB document can be used to expose sensitive information such as code, runtime memory and user data into MongoDB.

Details

Initializing a `Buffer` field in a document with integer `N` creates a `Buffer` of length `N` with non zero-ed out memory. **Example:**

```
var x = new Buffer(100); // uninitialized Buffer of length 100
// vs
var x = new Buffer('100'); // initialized Buffer with value of '100'
```

Example: Using unsafe Buffer constructor

Overview

Versions of `ws` prior to 1.0.1 are affected by a remote memory disclosure vulnerability.

In certain rare circumstances, applications which allow users to control the arguments of a `client.ping()` call will cause `ws` to send the contents of an allocated but non-zero-filled buffer to the server. This may disclose sensitive information that still exists in memory after previous use of the memory for other tasks.

snyk Test Vulnerability DB Docs Blog Features

Remote Memory Exposure

Affecting request package, versions <2.68.0 >2.2.5

Overview

request is a simplified http request client. A potential remote memory exposure vulnerability exists in request . If [illegible] X bytes of uninitialized memory will be sent in the body of the request.

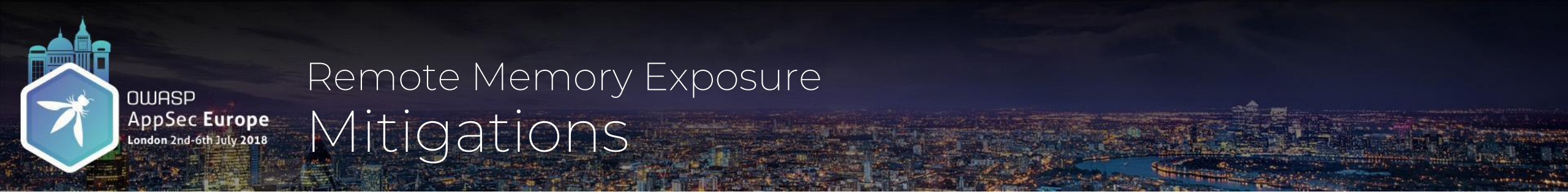
Note that while the impact of this vulnerability is high (memory exposure), exploiting it is likely difficult, as the attacker needs to somehow control the body type of the request. One potential exploit scenario is when a request is composed based on JSON input, including the body type, allowing a malicious JSON to trigger the memory leak.

Details

Constructing a `Buffer` class with integer `N` creates a `Buffer` of length `N` with non zero-ed out memory.
Example:

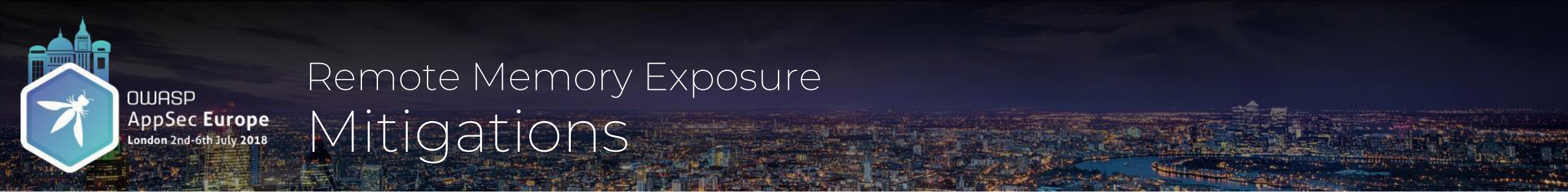
```
var x = new Buffer(100); // uninitialized Buffer of length 100
// vs
var x = new Buffer('100'); // initialized Buffer with value of '100'
```

Example: Using unsafe Buffer constructor



Remote Memory Exposure
Mitigations

✓ Upgrade to Node.js version 8.11.3 or later (also fixes DoS Vulnerability related to Buffer)



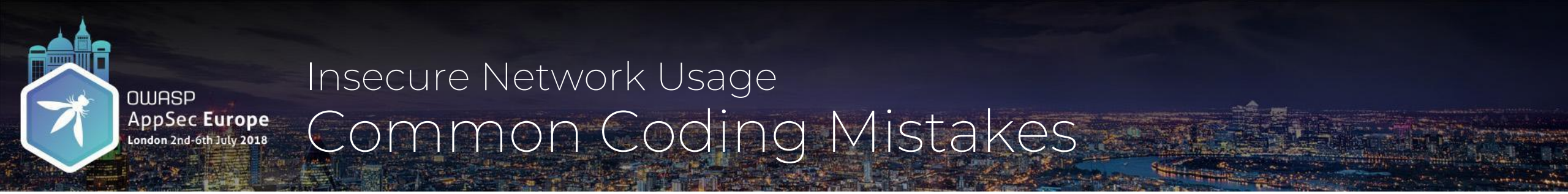
Remote Memory Exposure
Mitigations

- ✓ Upgrade to Node.js version 8.11.3 or later (also fixes DoS Vulnerability related to Buffer)
- ✓ Use a safe method Buffer.alloc(size) to create a buffer that is initialized with zeroes:

```
1  const buf = Buffer.alloc(5);
2  console.log(buf);
3  // Prints: <Buffer 00 00 00 00 00>
```

Sensitive Data Exposure

Pattern # 7 Insecure Network Usage

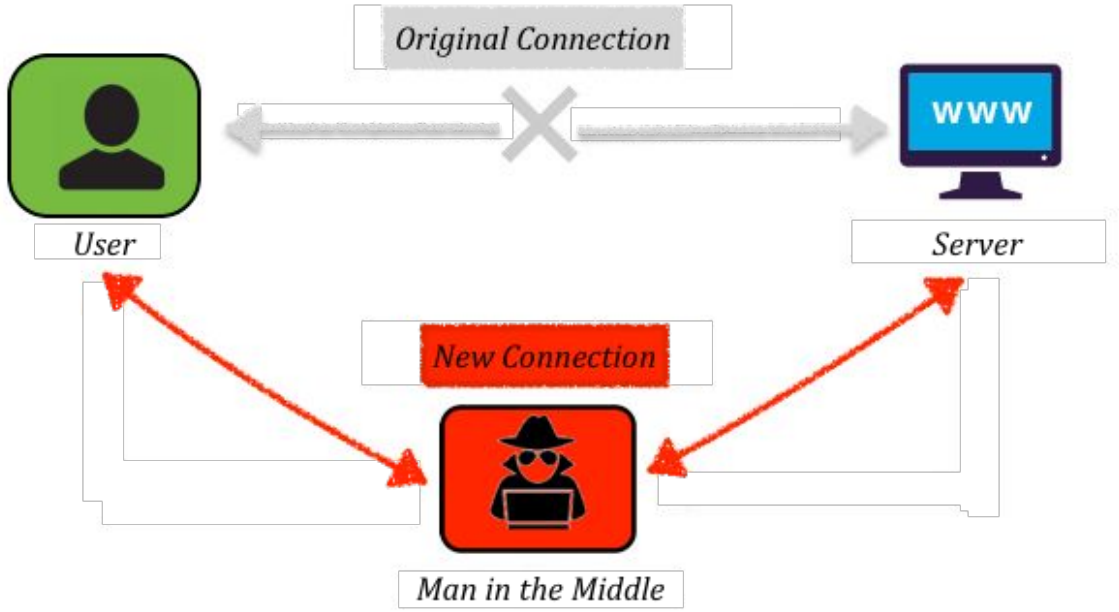


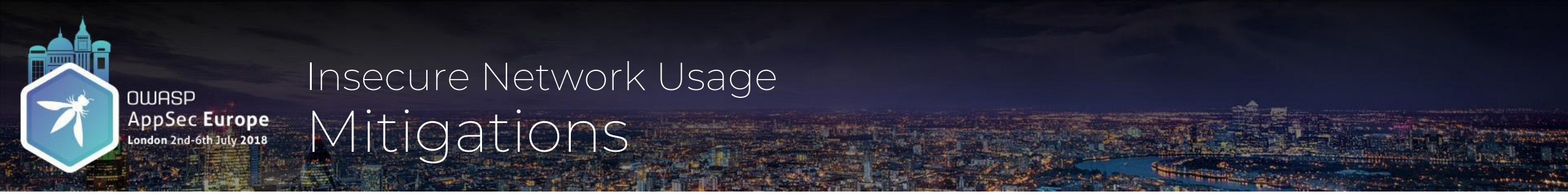
Insecure Network Usage
Common Coding Mistakes

- Using insecure HTTP protocol to download resources as part of install scripts or at runtime.

Insecure Network Usage
Common Coding Mistakes

- Using insecure HTTP protocol to download resources as part of install scripts or at runtime.

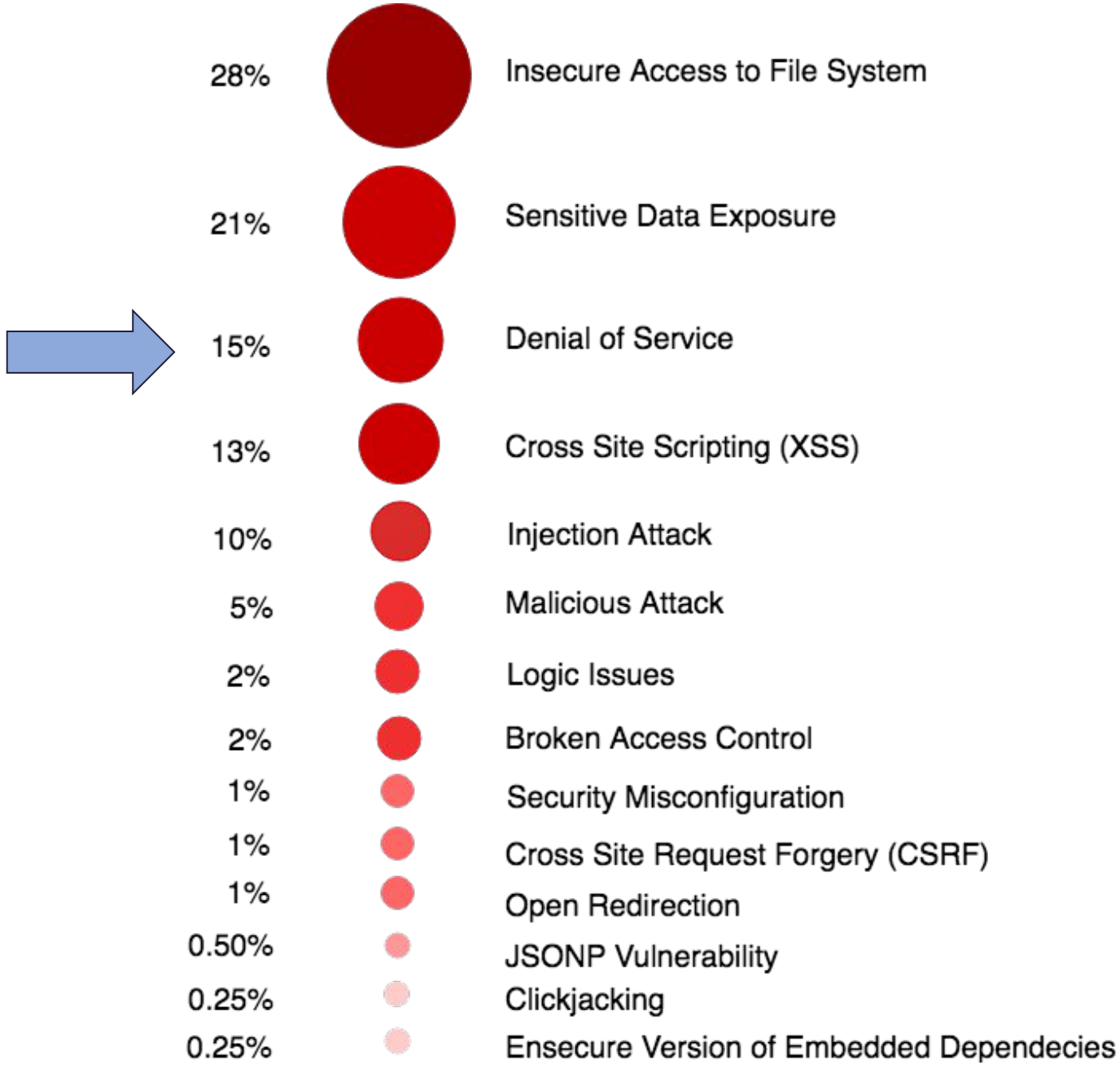




Insecure Network Usage
Mitigations

✓ Download resources over secure HTTPS connection.

✓ Provide an option for users to download dependencies in advance and specify the location path.


28%
Insecure Access to File System
21%
Sensitive Data Exposure
15%
Denial of Service
13%
Cross Site Scripting (XSS)
10%
Injection Attack
5%
Malicious Attack
2%
Logic Issues
2%
Broken Access Control
1%
Security Misconfiguration
1%
Cross Site Request Forgery (CSRF)
1%
Open Redirection
0.50%
JSONP Vulnerability
0.25%
Clickjacking
0.25%
Ensecure Version of Embedded Dependecies

Denial of Service (DoS)

Pattern # 8 Exhausting System Resources

Denial of Service

Module: uws

Published: October 17th, 2016

Reported by: Luigi Pinca

CVE-NONE

CWE-730

Vulnerable: >=0.10.0

<=0.10.8

Patched: >=0.10.9

Example: Exceeding V8’s maximum string size limit

Overview

Affected versions of `uws` do not properly handle large websocket messages when `permessage-deflate` is enabled, which may result in a denial of service condition.

If `uws` recieves a 256Mb websocket message when `permessage-deflate` is enabled, the server will compress the message prior to executing the length check, and subsequently extract the message prior to processing. This can result in a situation where an excessively large websocket message passes the length checks, yet still gets cast from a Buffer to a string, which will exceed v8's maximum string size and crash the process.

Vulnerability DB › npm

Denial of Service (DoS)

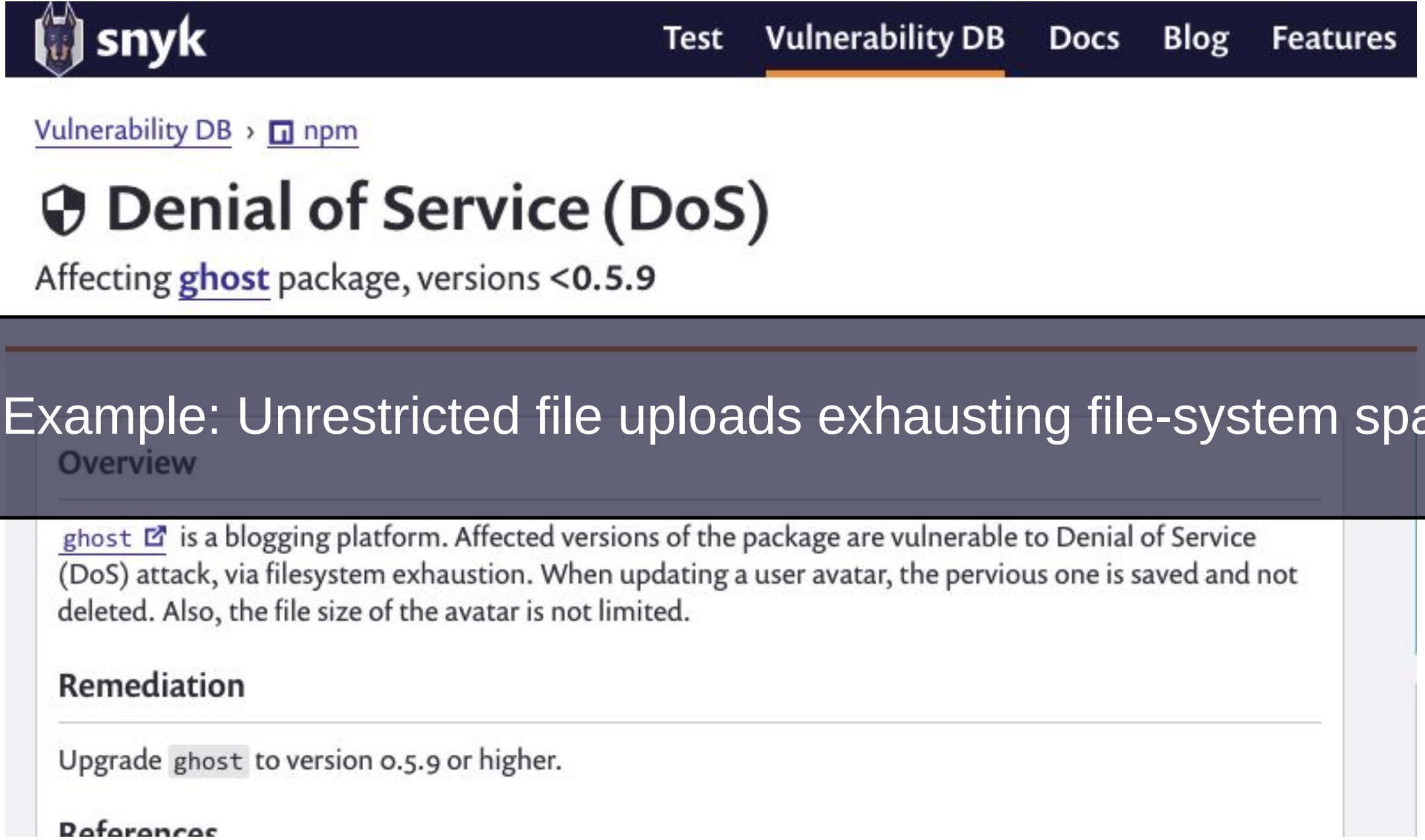
Affecting **websocket-driver** package, versions **<0.3.1**

Example: Exceeding V8’s maximum buffer size limit

Overview

websocket-driver is WebSocket protocol handler with pluggable I/O.

Affected versions of this package are vulnerable to Denial of Service (DoS) attacks. The `Buffer` length is immediately allocated after reading the frame, up to a length that is no more that `MAX_LENGTH`, which is 2^53 - 1 (the largest precisely representable JS integer), and rejects larger frames with a 1009 error before creating the new Buffer. But Node buffers have a max length of 1GB (0x3fffffff). Parsing an incoming frame with length between 1GB and `MAX_LENGTH`, the parser will throw (and perhaps crash your whole server). Attackers can use this to their advantage and cause a Denial of Service on the servers.



Example: Unrestricted file uploads exhausting file-system space

DoS by Exhausting System Resources
Common Coding Mistake

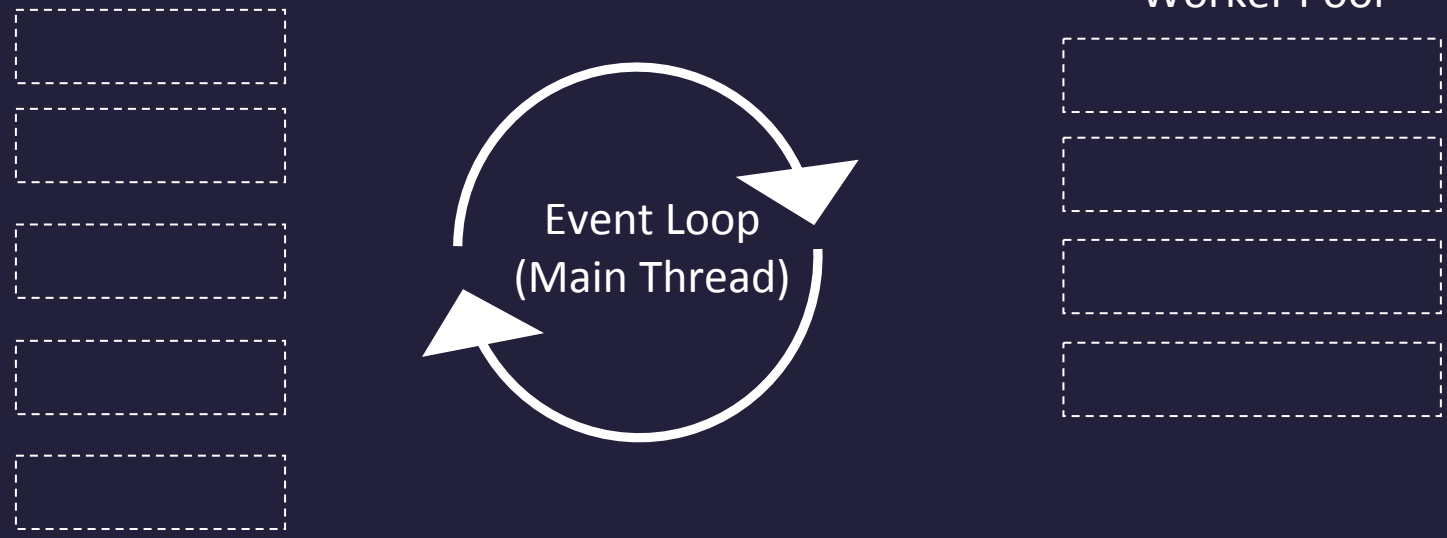
- Allocating unrestricted amount of system resources based on the size of a user input.

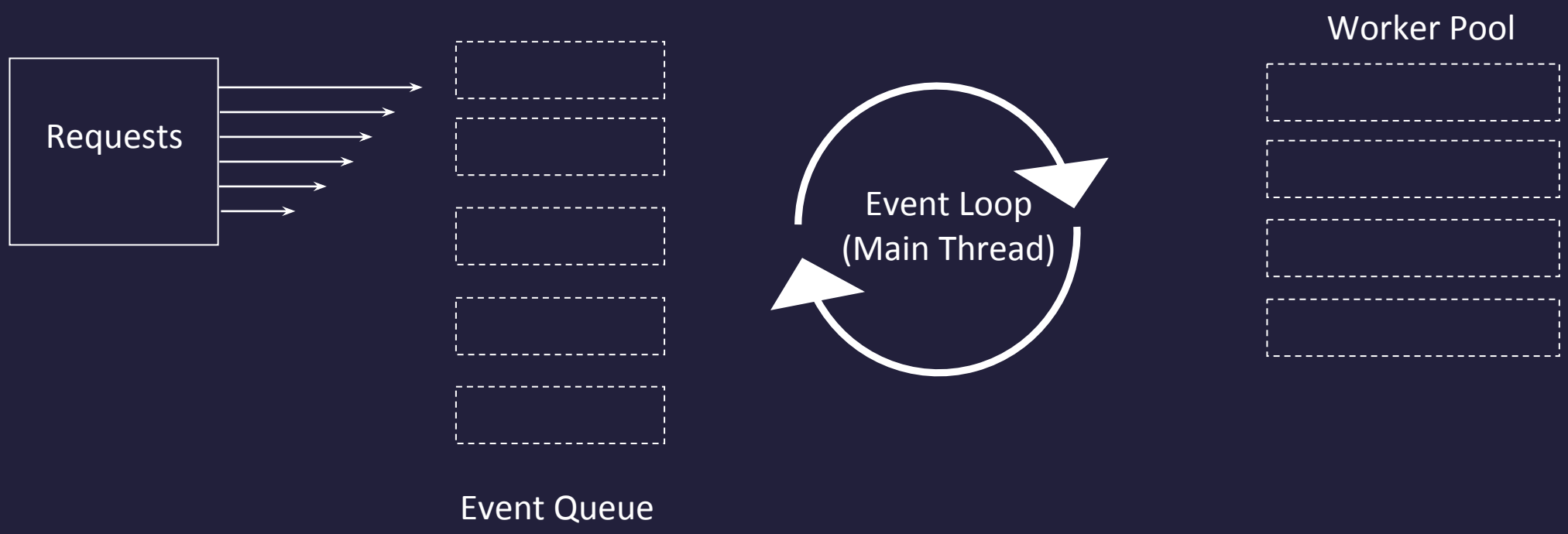
DoS by Exhausting System Resources
Mitigations

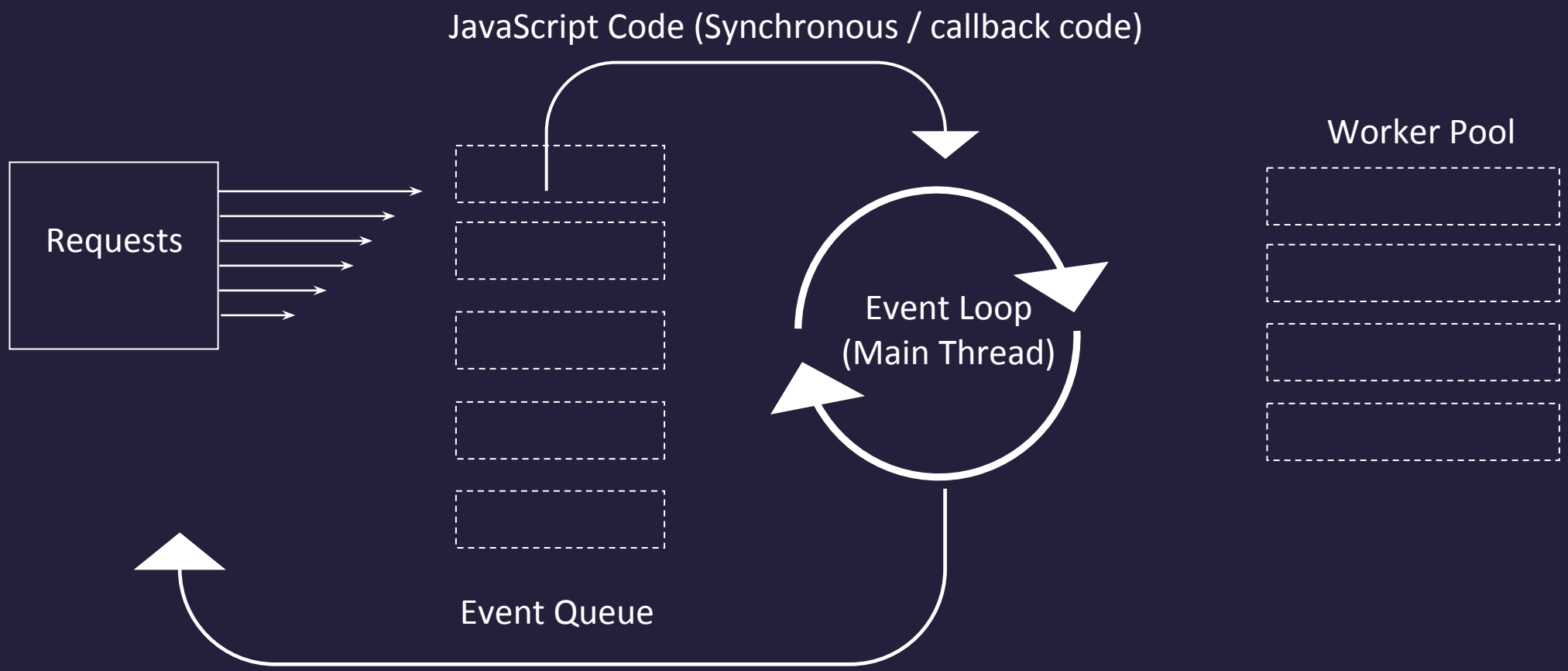
✓ Validate size of a user input before processing it

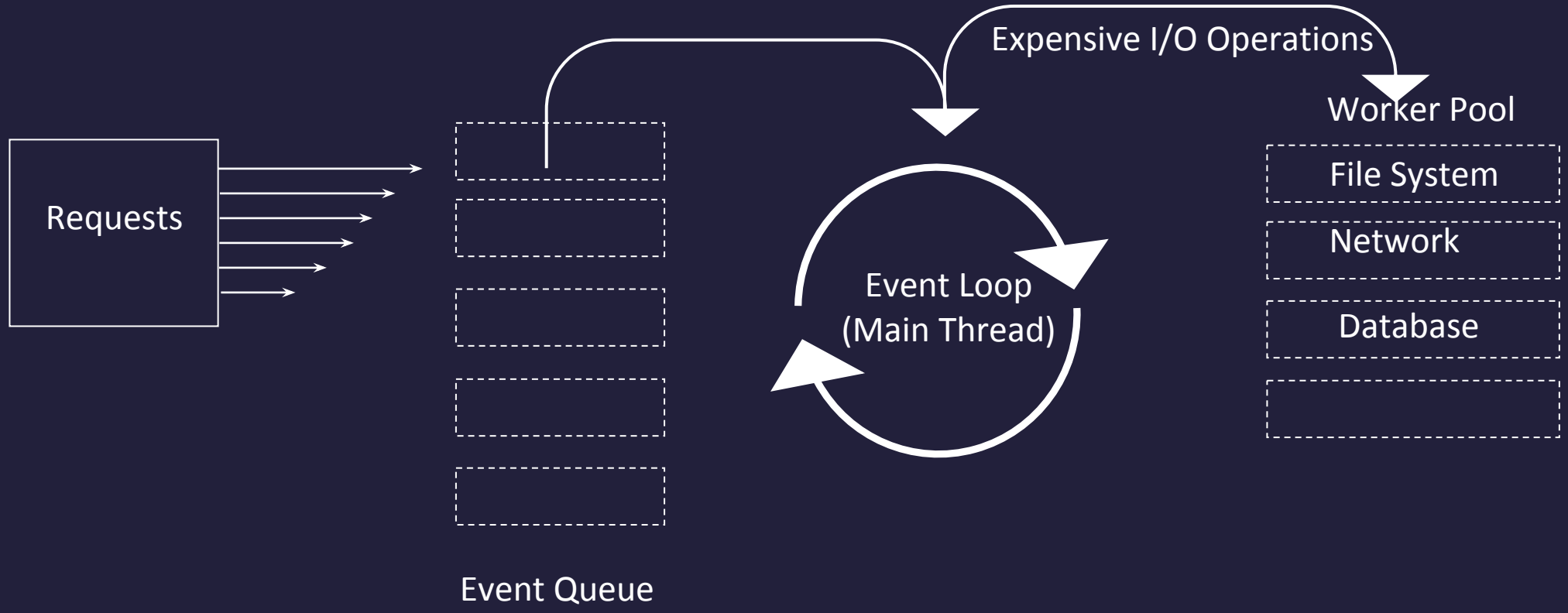
Denial of Service (DoS)

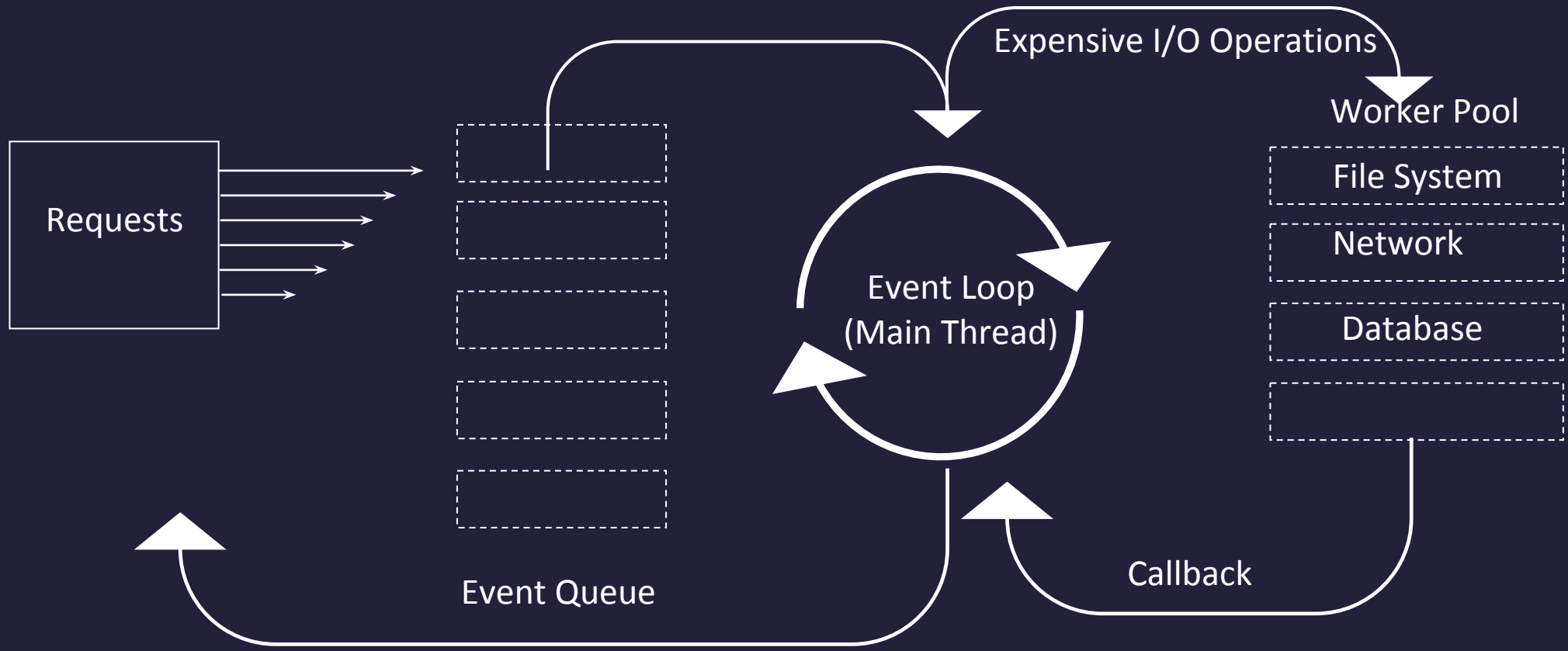
By Small Targeted Inputs

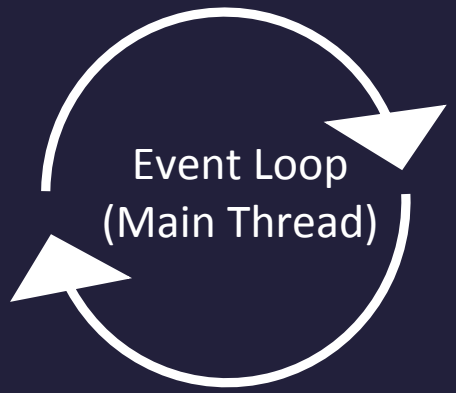

Worker Pool
Event Loop
(Main Thread)
Event Queue

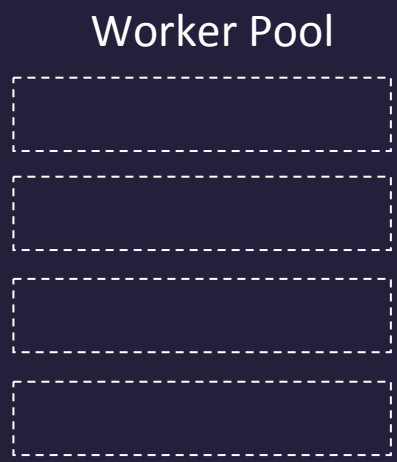

Requests
Event Queue
Event Loop
(Main Thread)
Worker Pool

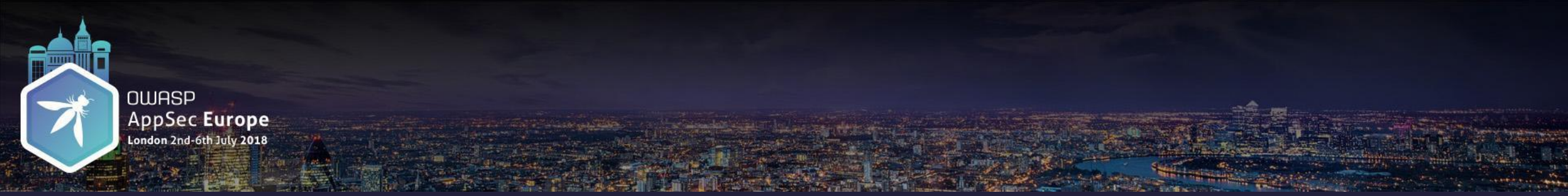

JavaScript Code (Synchronous / callback code)
Requests
Event Loop
(Main Thread)
Worker Pool
Event Queue


Expensive I/O Operations
Worker Pool
File System
Network
Database
Requests
Event Loop
(Main Thread)
Event Queue


Expensive I/O Operations
Worker Pool
File System
Network
Database
Requests
Event Loop
(Main Thread)
Event Queue
Callback


Event Loop
(Main Thread)

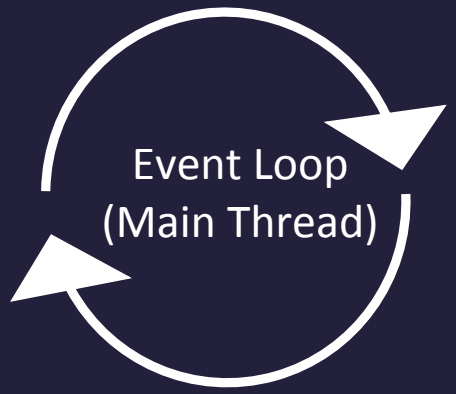

Worker Pool



A malicious client could submit an "evil input", make your threads block, and keep them from working on other clients.

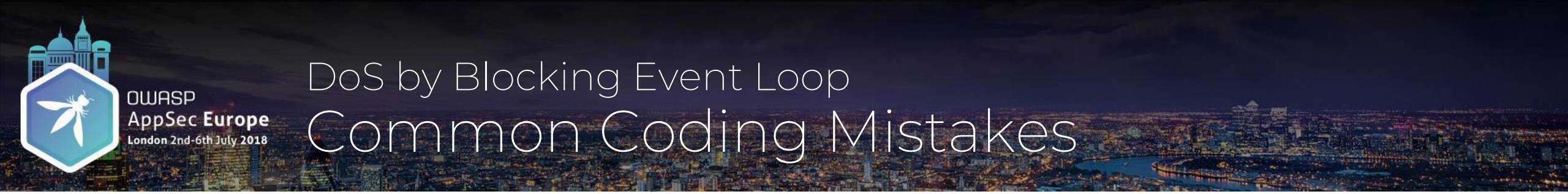
This would be a Denial of Service attack.

- Node.js Docs


Event Loop
(Main Thread)

Denial of Service (DoS)

Pattern # 9 Blocking Event Loop



DoS by Blocking Event Loop
Common Coding Mistakes

- Running an execution loop whose iterations depend on the length of a user input.

Denial of Service

Vulnerable: < 2.0.0

Patched: >=2.0.0

Module: ecstatic

Published: December 13th, 2017

Reported by: Checkmarx

CVE-2016-10703

CWE-400

Overview

`ecstatic` , a simple static file server middleware, is vulnerable to denial of service. If a payload with a large number of null bytes (`%00`) is provided by an attacker it can crash ecstatic by running it out of memory.

Node Security Platform

REPORT VULNERABILITY ADVISORIES LOGIN

Denial of Service

Vulnerable: < 2.0.0

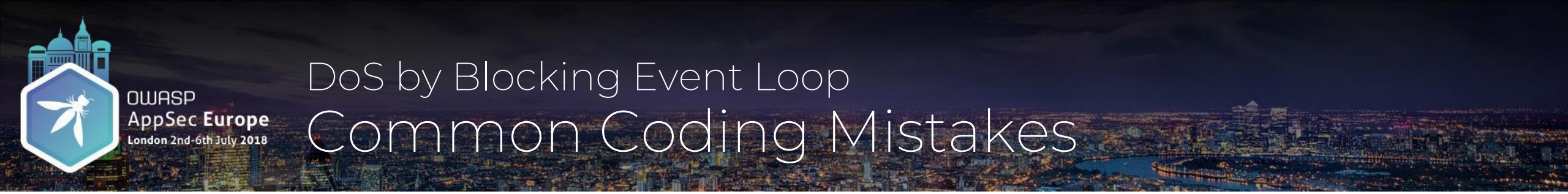
Patched: >=2.0.0

Module: ecstatic

```
54        // Strip any null bytes from the url
55        while(req.url.indexOf('%00') !== -1) {
56          req.url = req.url.replace(/\%00/g, '');
57        }
```

Overview

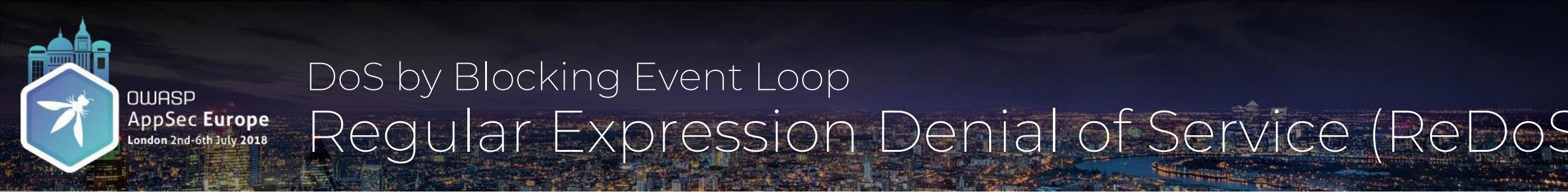
`ecstatic` , a simple static file server middleware, is vulnerable to denial of service. If a payload with a large number of null bytes (`%00`) is provided by an attacker it can crash ecstatic by running it out of memory.



DoS by Blocking Event Loop
Common Coding Mistakes

- Running an execution loop whose iterations depend on the length of a user input.

- Using unsafe Regular Expressions



DoS by Blocking Event Loop
Regular Expression Denial of Service (ReDoS

- By default, regular expressions get executed in the main event loop thread

- Evil regex can take exponential execution time when applied to certain non-matching inputs.

ReDoS

Vulnerable: <=1.1.6

Patched: >=1.1.7

CVSS 6.2 Medium

Module: brace-expansion

Published: April 25th, 2017

Reported by: myvyang

CVE-NONE

CWE-400

Overview

Affected versions of `brace-expansion` are vulnerable to a regular expression denial of service condition.

Proof of Concept

```
var expand = require('brace-expansion');
expand
('{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
```

ReDoS

Vulnerable: <=1.1.6

Patched: >=1.1.7

CVSS 6.2 Medium

Module: brace-expansion

Published: April 25th, 2017

Reported by: myvyang

CVE-NONE

CWE-400

^(.*,)+(.+)?$/

Overview

Affected versions of `brace-expansion` are vulnerable to a regular expression denial of service condition.

Proof of Concept

```
var expand = require('brace-expansion');
expand
('{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
```

ReDoS

Vulnerable: <=1.1.6

Patched: >=1.1.7

CVSS 6.2 Medium

Module: brace-expansion

Published: April 25th, 2017

Reported by: myvyang

CVE-NONE

CWE-400

Input format: ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,\n

Input Length	Execution Time
25	2 sec
26	4 sec
27	9 sec
28	15 sec
30	1 minute
35	34 minutes

Overview

Affected versions of `brace-expansion` are vulnerable to a regular expression denial of service condition.

Proof of Concept

```
var expand = require('brace-expansion');
expand
('{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
```

Denial of Service (DoS)

Pattern # 10 Crashing Event Loop By Unhandled Operational Errors

DoS by Crashing Event Loop by Unhandled Operational Errors

Common Coding Mistakes

1. Failing to handle Invalid User Inputs

Vulnerability DB › npm

Denial of Service (DoS)

Affecting **connect** package, versions **>=1.4.0 <2.0.0**

Invalid Character

Root Cause: Unexpected Trailing \ in URL localhost:3000/index.html\

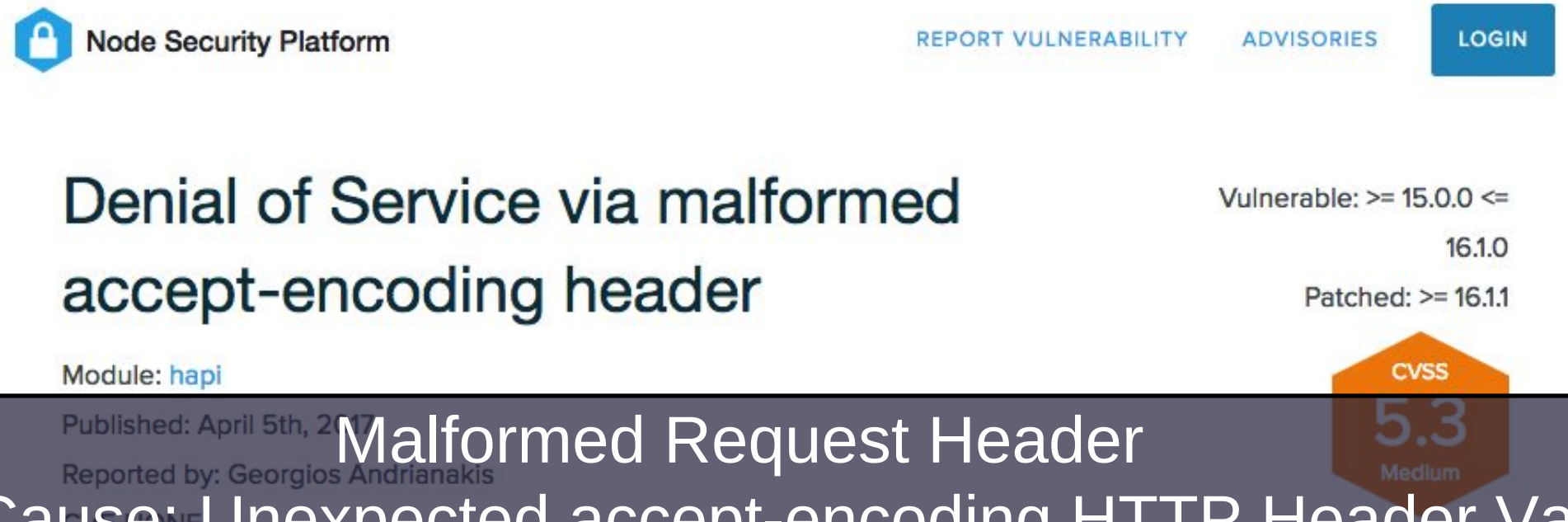
Overview

connect is a high performance middleware framework.

Affected versions of the package are vulnerable to Denial of Service (DoS) attacks. It is possible to crash the node server by requesting a url with a trailing backslash in the end.

Remediation

Upgrade `connect` to version 2.0.0 or higher.



Malformed Request Header

Root Cause: Unexpected accept-encoding HTTP Header Value

Overview

Affected versions of `hapi` will crash or lock the event loop when a malformed `accept-encoding` header is recieved.

snyk Test Vulnerability DB Docs Blog Features

Affecting nunjucks package, versions <2.4.3

Overview

`nunjucks` is a powerful templating engine.

Like many templating engines, it automatically HTML encodes any `string` value included in the template using the `{{ some-variable }}` notation. These variables are often user generated, but the HTML Encoding protects the application from Cross-site Scripting (XSS) attacks.

However, if the variable passed in is an array, no HTML encoding is applied, exposing an easy path to XSS. The risk of exploit is especially high given the fact `express`, `koa` and many other Node.js servers allow users to force a query parameter to be an array using the `param[]=value` notation.

Details

The issue opened by Matt Austin explains the vulnerability very well:

> *The following string works as expected:*

Invalid Object Shape

Root Cause: Type coercion of HTTP Request Parameters

DoS by Crashing Event Loop by Unhandled Operational Errors

Common Coding Mistakes

- User input coercion via HTTP Request Parameters in qs, Express, Koa

```
// GET /search?conference=appSecEU
request.query.conference
//=> ”appSecEU”
```

DoS by Crashing Event Loop by Unhandled Operational Errors

Common Coding Mistakes

- User input coercion via HTTP Request Parameters in qs, Express, Koa

```
// GET /search?conference=appSecEU&conference=appSecUSA
request.query.conference
//=>
```

DoS by Crashing Event Loop by Unhandled Operational Errors
Common Coding Mistakes

- User input coercion via HTTP Request Parameters in qs, Express, Koa

```
// GET /search?conference=appSecEU&conference=appSecUSA
request.query.conference
//=> [”appSecEU”, “appSecUSA”]
```

DoS by Crashing Event Loop by Unhandled Operational Errors

Common Coding Mistakes

- User input coercion via HTTP Request Parameters in qs, Express, Koa

```
// GET /search?conference[]=appSecEU
request.query.conference
//=> [" appSecEU"]
```

DoS by Crashing Event Loop by Unhandled Operational Errors
Common Coding Mistakes

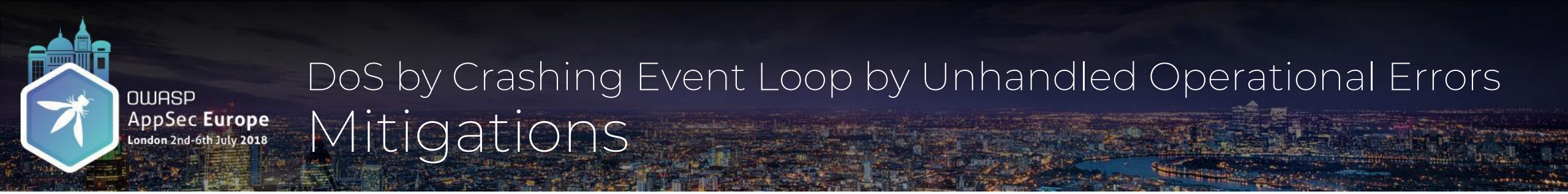
- User input coercion via HTTP Request Parameters in qs, Express, Koa

```
// GET /search?conference[appSecEU][year]=2018
request.query.conference
//=>
```

DoS by Crashing Event Loop by Unhandled Operational Errors
Common Coding Mistakes

- User input coercion via HTTP Request Parameters in qs, Express, Koa

```
// GET /search?conference[appSecEU][year]=2018
request.query.conference
//=> {appSecEU: { year: '2018' }}
```



DoS by Crashing Event Loop by Unhandled Operational Errors

Mitigations

✓ Validate user inputs for expected value, type, or shape before processing it. (using joi package, for example)

DoS by Crashing Event Loop by Unhandled Operational Errors

Common Coding Mistakes

1. Failing to handle Unexpected User Inputs
2. Missing or incorrect operational error handling

DoS by Crashing Event Loop by Unhandled Operational Errors

Common Coding Mistakes

Mechanisms to communicate Operational Errors

```
throw new Error('something bad happened!');
```

DoS by Crashing Event Loop by Unhandled Operational Errors

Common Coding Mistakes

Mechanisms to communicate Operational Errors

```
throw new Error('something bad happened!');
```

```
callback(new Error('something bad happened!'));
```

DoS by Crashing Event Loop by Unhandled Operational Errors
Common Coding Mistakes

Mechanisms to communicate Operational Errors

```
throw new Error('something bad happened!');
```

```
callback(new Error('something bad happened!'));
```

```
return Promise.reject(new Error('something bad happened!'));
```

DoS by Crashing Event Loop by Unhandled Operational Errors

Common Coding Mistakes

Mechanisms to communicate Operational Errors

```
throw new Error('something bad happened!');
```

```
callback(new Error('something bad happened!'));
```

```
return Promise.reject(new Error('something bad happened!'));
```

```
myEmitter.emit('error', new Error(something bad happened!'));
```

Node Security Platform

REPORT VULNERABILITY ADVISORIES LOGIN

Denial of Service

Vulnerable: <=6.4.0

Patched: >=6.4.1

Module: nes

Published: April 14th, 2017

Reported by: iipokypatop

CVE-NONE

CWE-730

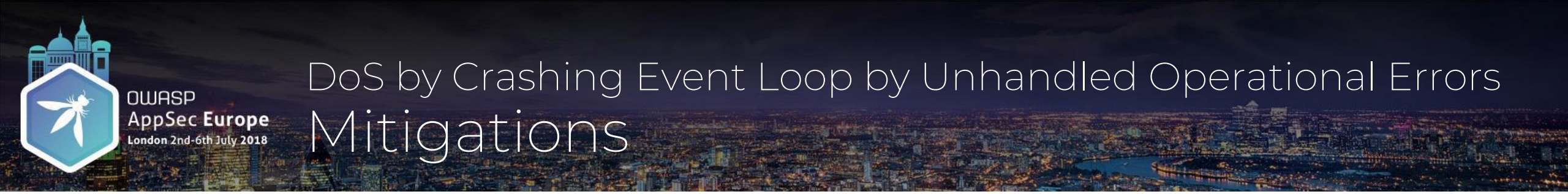
CVSS 7.5 High

Example: Failure to handle error object passed in the callback

Overview

Affected versions of `nes` are vulnerable to denial of service when given an invalid `cookie` header, and websocket authentication is set to `cookie`. Submitting an invalid cookie on the websocket upgrade request will cause the node process to throw and exit.

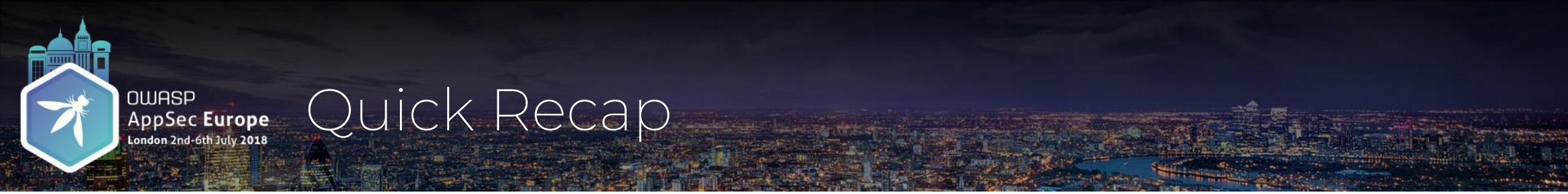
Remediation



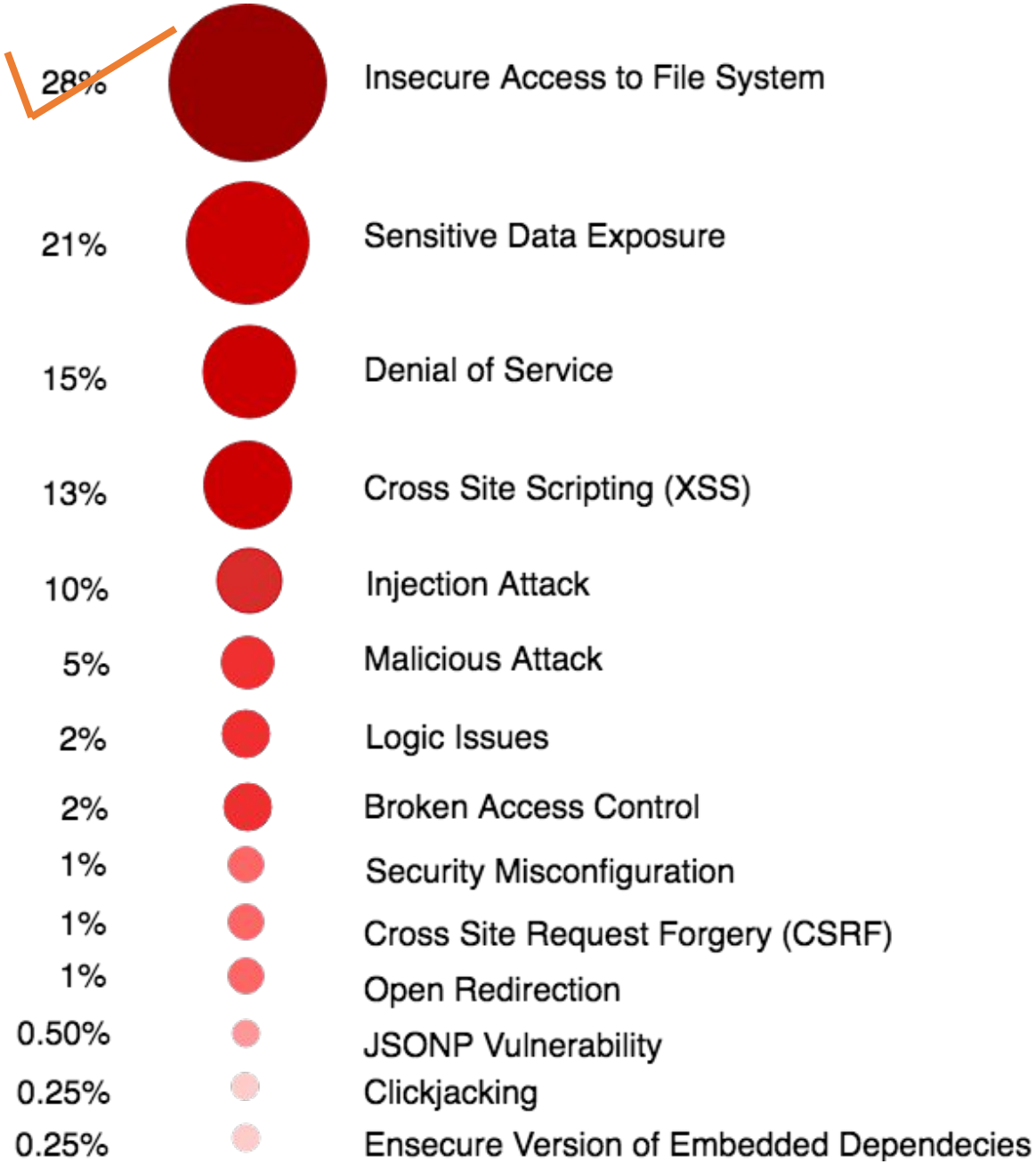
DoS by Crashing Event Loop by Unhandled Operational Errors

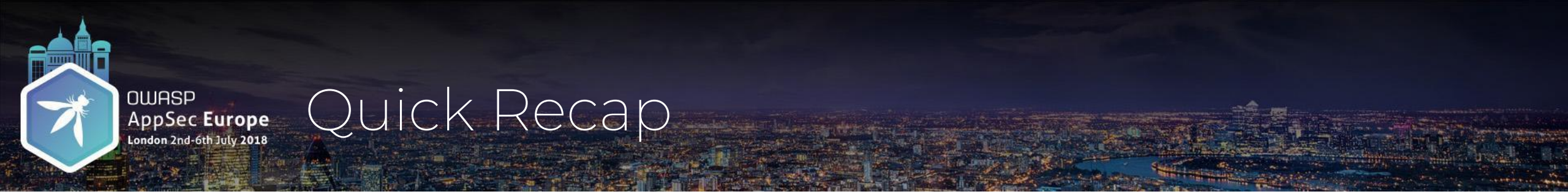
Mitigations

✓ Be aware of the error delivery mechanism used by the invoked function and handle errors accordingly.



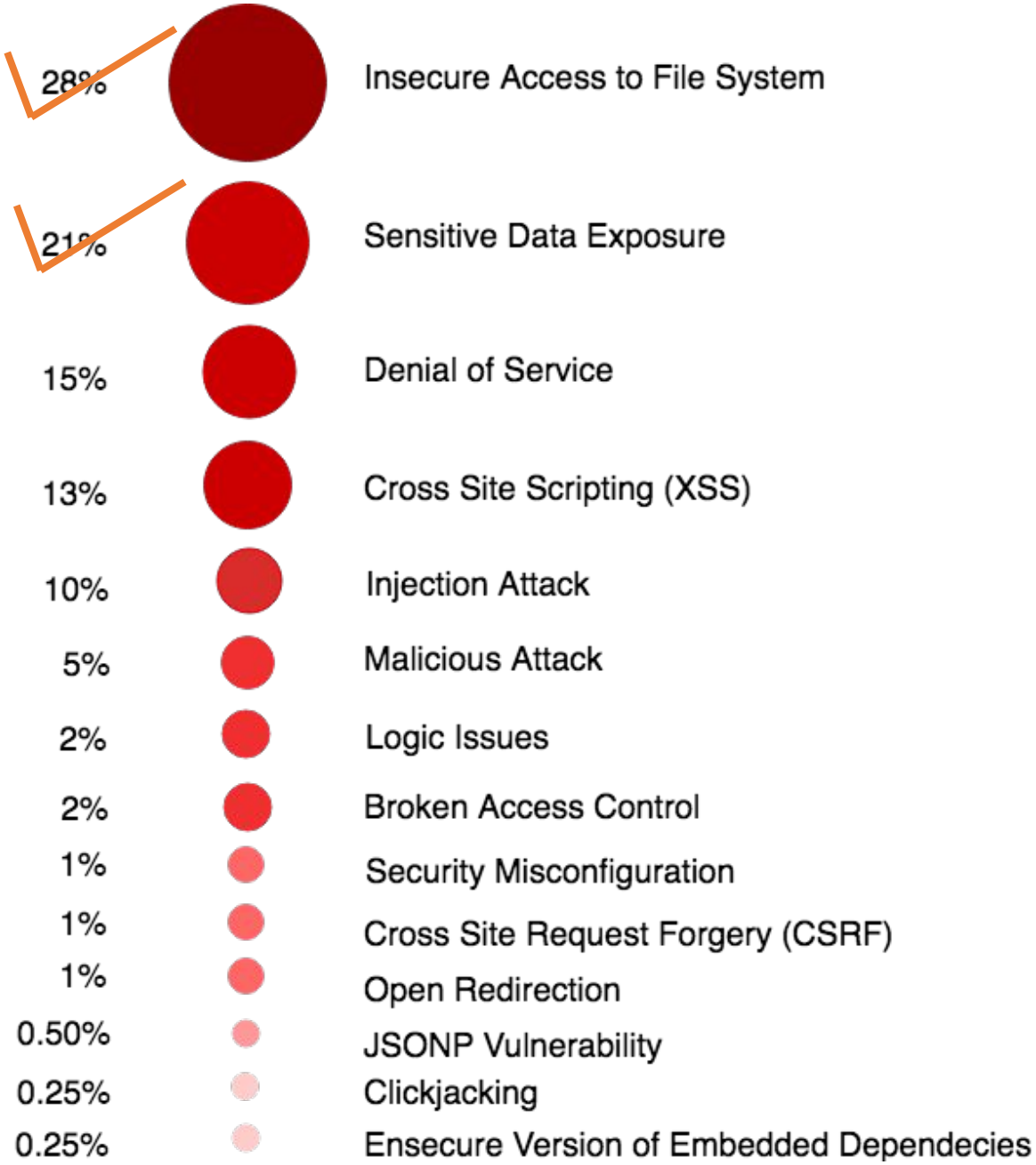
Quick Recap

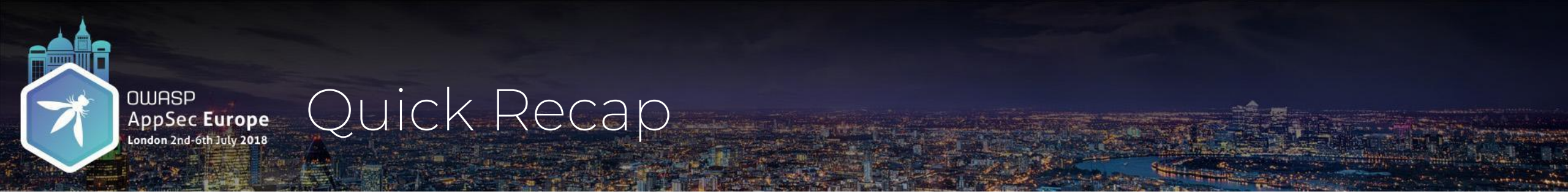

28%
Insecure Access to File System
21%
Sensitive Data Exposure
15%
Denial of Service
13%
Cross Site Scripting (XSS)
10%
Injection Attack
5%
Malicious Attack
2%
Logic Issues
2%
Broken Access Control
1%
Security Misconfiguration
1%
Cross Site Request Forgery (CSRF)
1%
Open Redirection
0.50%
JSONP Vulnerability
0.25%
Clickjacking
0.25%
Ensecure Version of Embedded Dependecies



Quick Recap

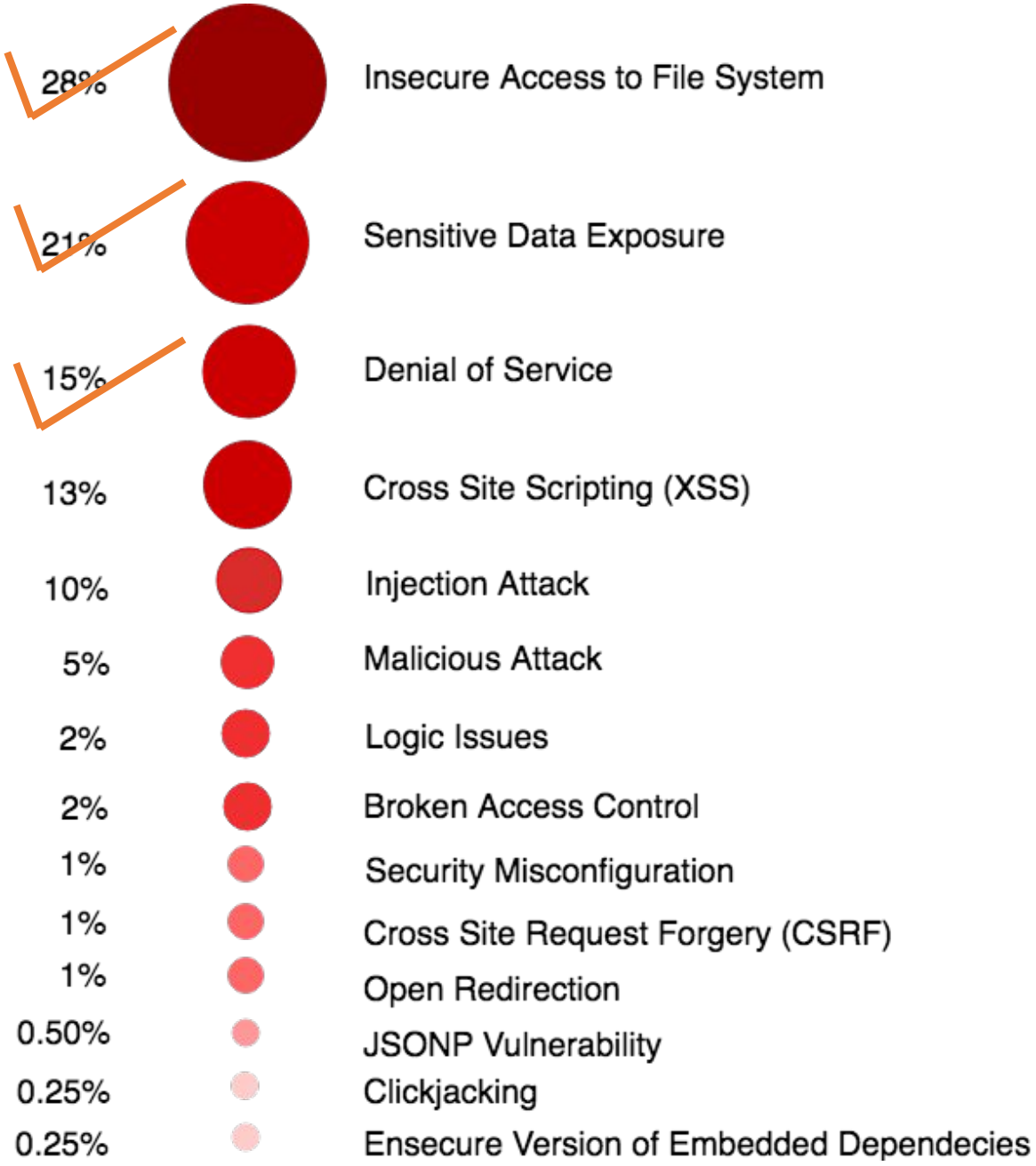
- Insecure Access to File System
  - Pattern #1 Directory Traversal
  - Pattern #2 Symlink Attack

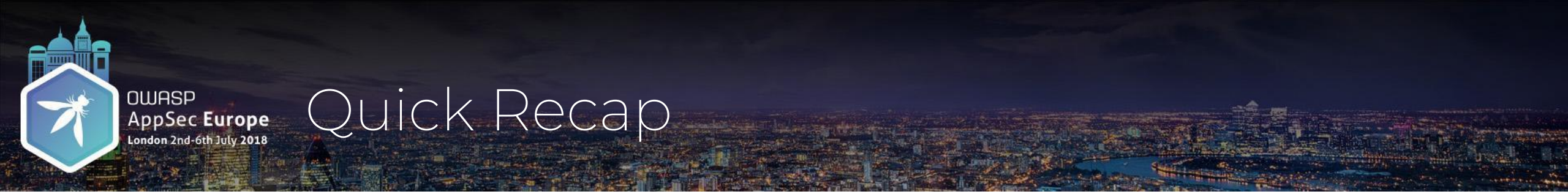

28%
Insecure Access to File System
21%
Sensitive Data Exposure
15%
Denial of Service
13%
Cross Site Scripting (XSS)
10%
Injection Attack
5%
Malicious Attack
2%
Logic Issues
2%
Broken Access Control
1%
Security Misconfiguration
1%
Cross Site Request Forgery (CSRF)
1%
Open Redirection
0.50%
JSONP Vulnerability
0.25%
Clickjacking
0.25%
Ensecure Version of Embedded Dependecies



Quick Recap

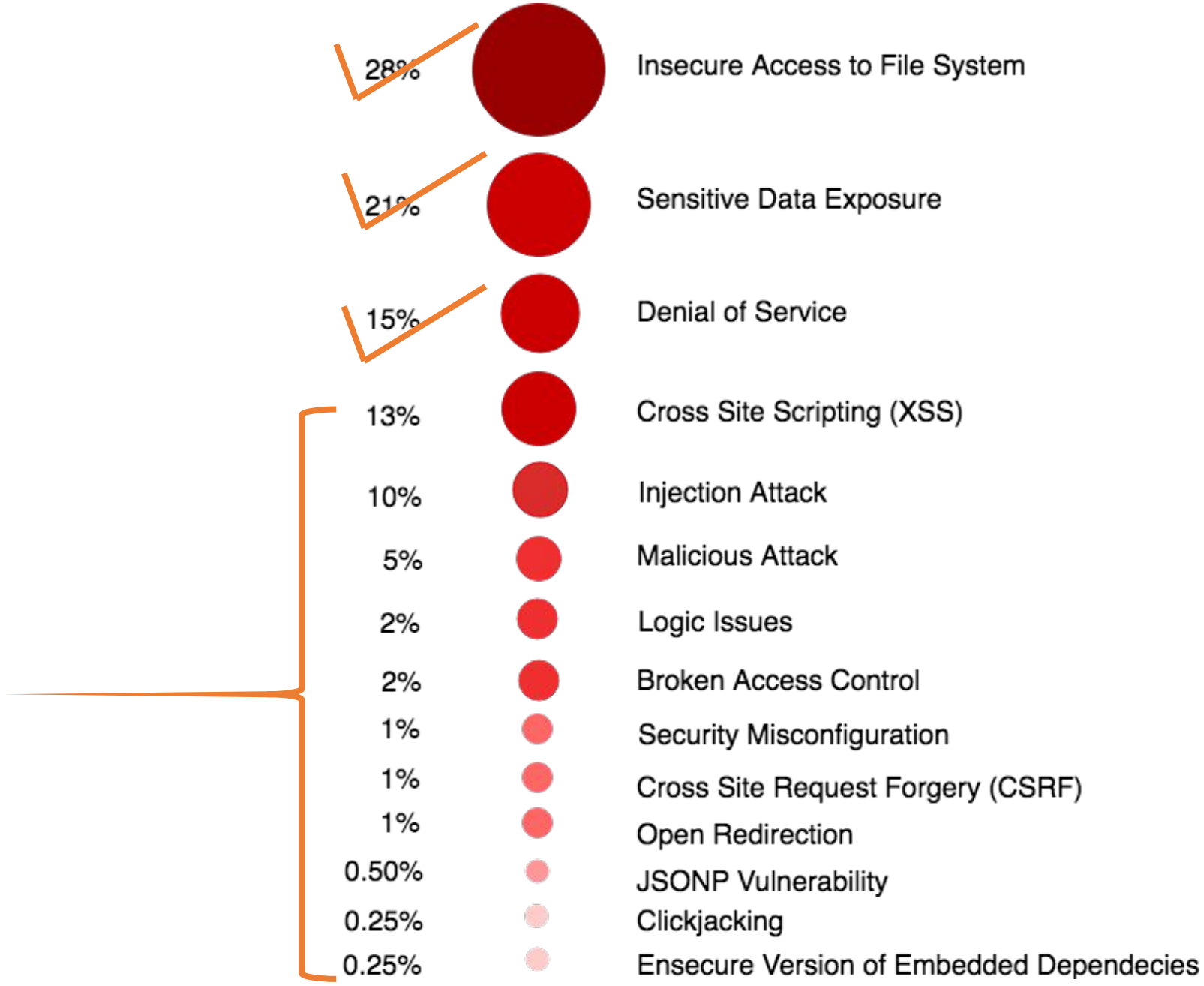
- Sensitive Data Exposure
  - Pattern #1 Leaking Application Secrets
  - Pattern #2 Predictable Secrets (Insecure Randomness)
  - Pattern #3 Predictable Secrets (Non-constant Time Comparison)
  - Pattern #4 Remote Memory Exposure
  - Pattern #5 Insecure Network Usage


28%
Insecure Access to File System
21%
Sensitive Data Exposure
15%
Denial of Service
13%
Cross Site Scripting (XSS)
10%
Injection Attack
5%
Malicious Attack
2%
Logic Issues
2%
Broken Access Control
1%
Security Misconfiguration
1%
Cross Site Request Forgery (CSRF)
1%
Open Redirection
0.50%
JSONP Vulnerability
0.25%
Clickjacking
0.25%
Ensecure Version of Embedded Dependecies



Quick Recap

- Denial of Service
  - Pattern #1 Exhausting System Resources
  - Pattern #2 Blocking Event Loop
  - Pattern #3 Crashing Event Loop By Unhandled Operational Errors

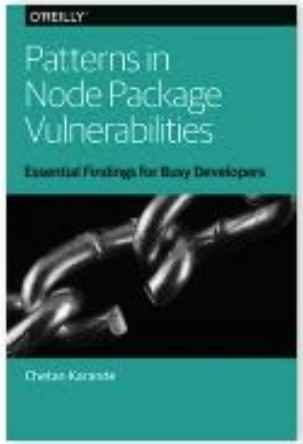

28%
Insecure Access to File System
21%
Sensitive Data Exposure
15%
Denial of Service
13%
Cross Site Scripting (XSS)
10%
Injection Attack
5%
Malicious Attack
2%
Logic Issues
2%
Broken Access Control
1%
Security Misconfiguration
1%
Cross Site Request Forgery (CSRF)
1%
Open Redirection
0.50%
JSONP Vulnerability
0.25%
Clickjacking
0.25%
Ensecure Version of Embedded Dependecies

Patterns in Node Package Vulnerabilities

O'REILLY®

Safari

Search... Enterprise Pricing



Patterns in Node Package Vulnerabilities

by Chetan Karande

Publisher: O'Reilly Media, Inc.

Release Date: June 2018

ISBN: 9781491999981

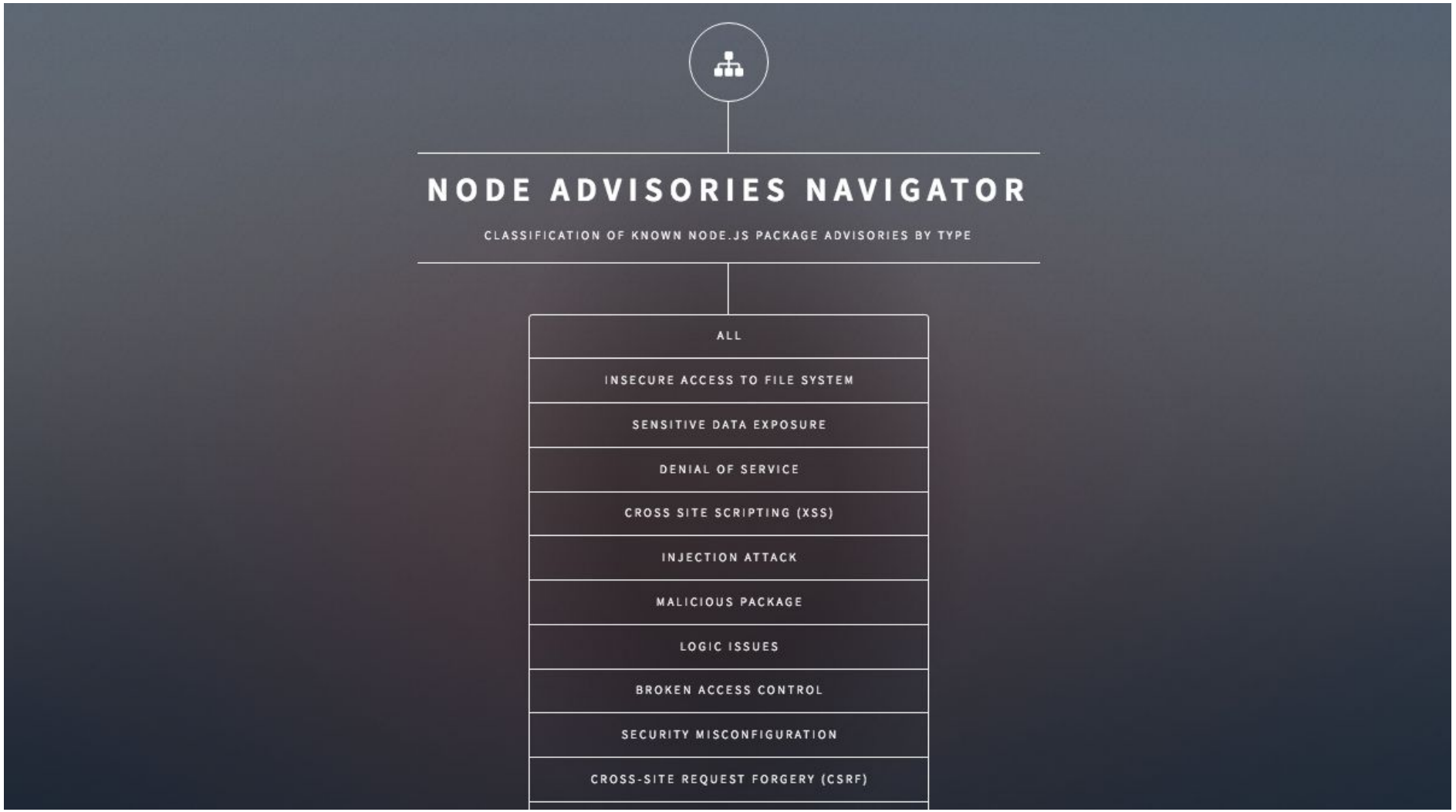
Topics: Node.js

View table of contents

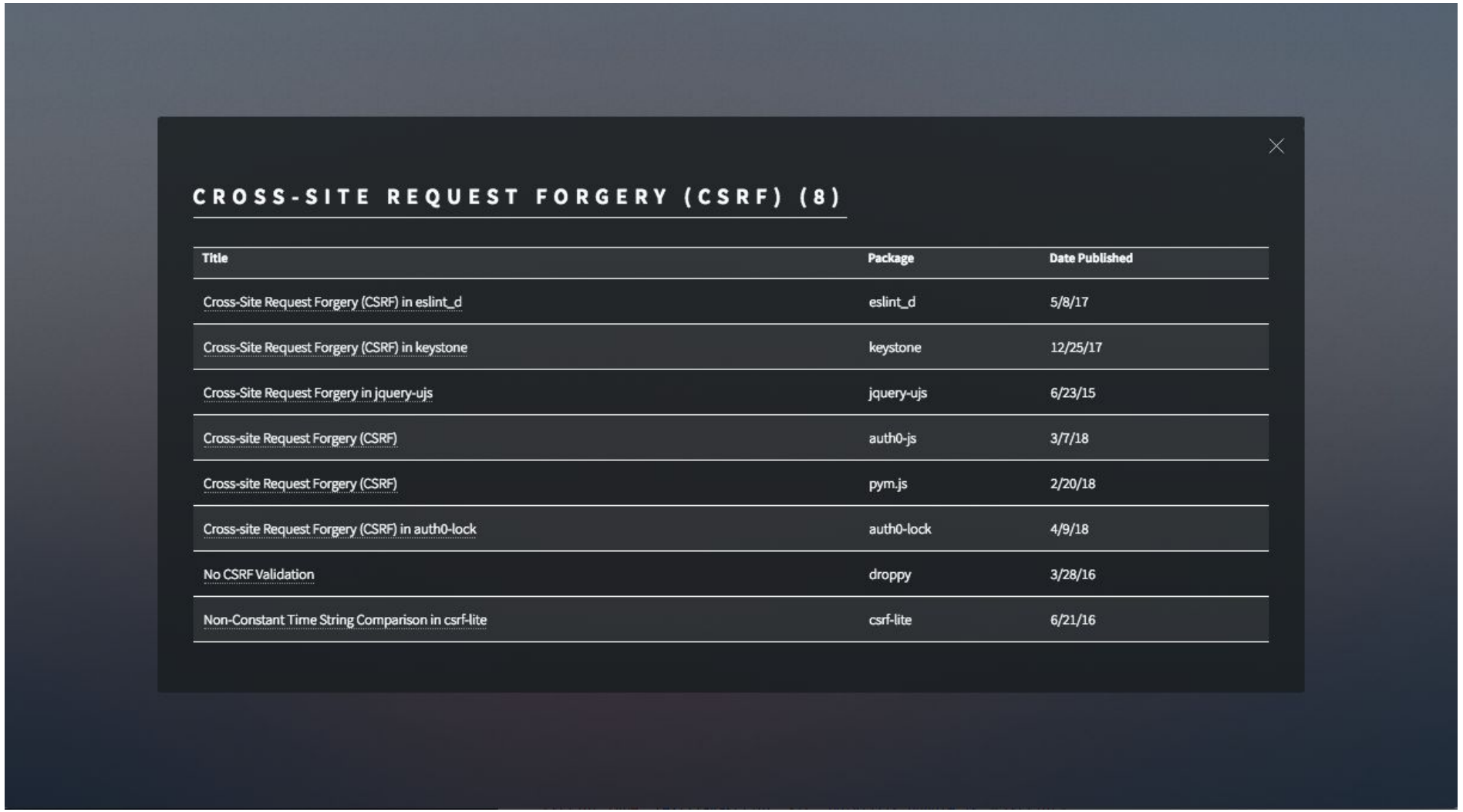
Book Description

With more than 500 new Node.js packages arriving each day, npm is the world's largest reusable package registry and the Node
But as the number of detected vulnerabilities continues to rise significantly, the packages themselves are becoming a liability. Th
opers and penetration testers practical strategies for evaluating and working with today's npm packages.

node.advisories.io



node.advisories.io



CROSS-SITE REQUEST FORGERY (CSRF) (8)

Title	Package	Date Published
Cross-Site Request Forgery (CSRF) in eslint_d	eslint_d	5/8/17
Cross-Site Request Forgery (CSRF) in keystone	keystone	12/25/17
Cross-Site Request Forgery in jquery-ujs	jquery-ujs	6/23/15
Cross-site Request Forgery (CSRF)	auth0-js	3/7/18
Cross-site Request Forgery (CSRF)	pym.js	2/20/18
Cross-site Request Forgery (CSRF) in auth0-lock	auth0-lock	4/9/18
No CSRF Validation	droppy	3/28/16
Non-Constant Time String Comparison in csrf-lite	csrf-lite	6/21/16


OWASP
AppSec Europe
London 2nd-6th July 2018
@karande_c