OWASP Dependency Check

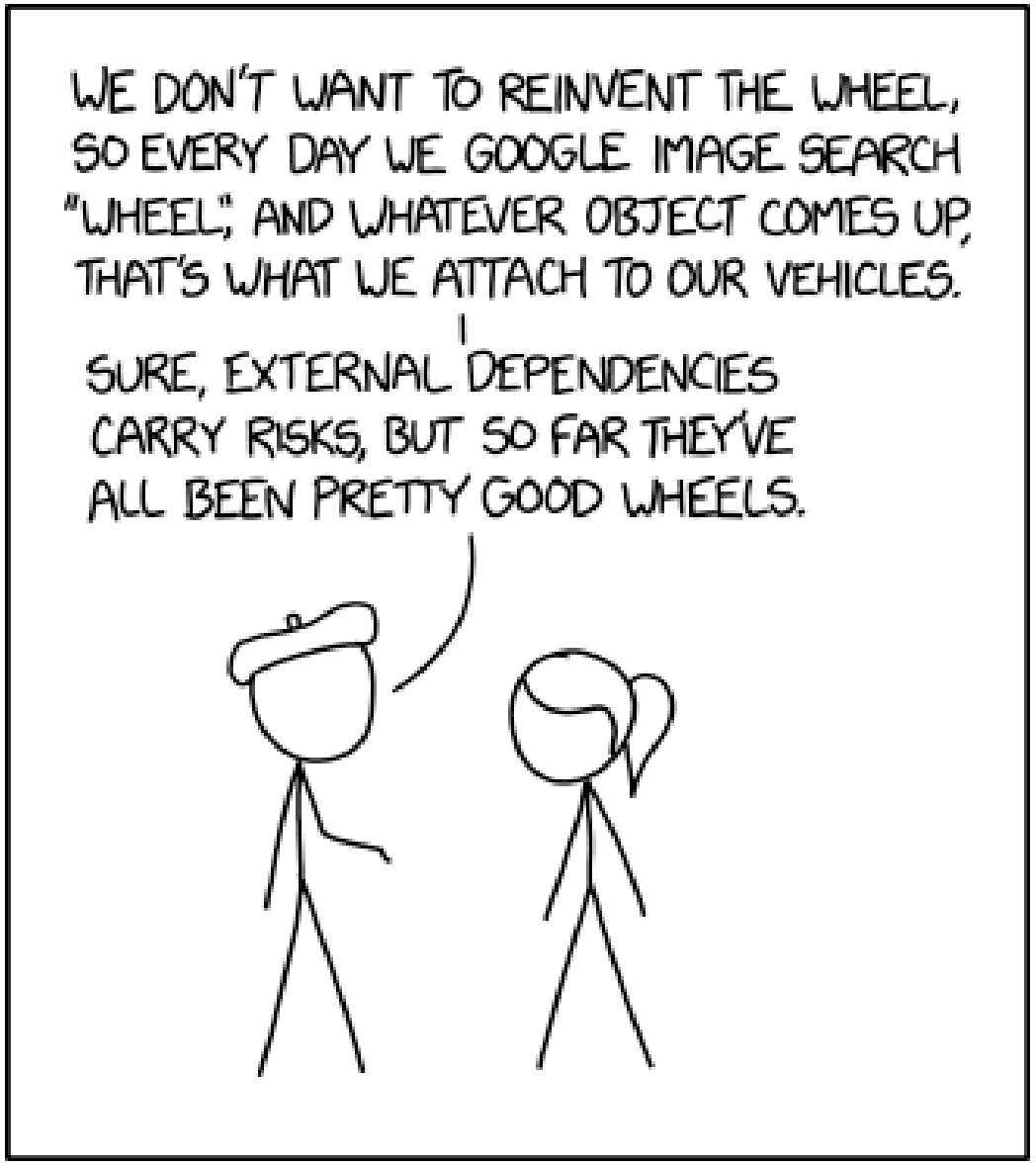
A Primer

The Problem

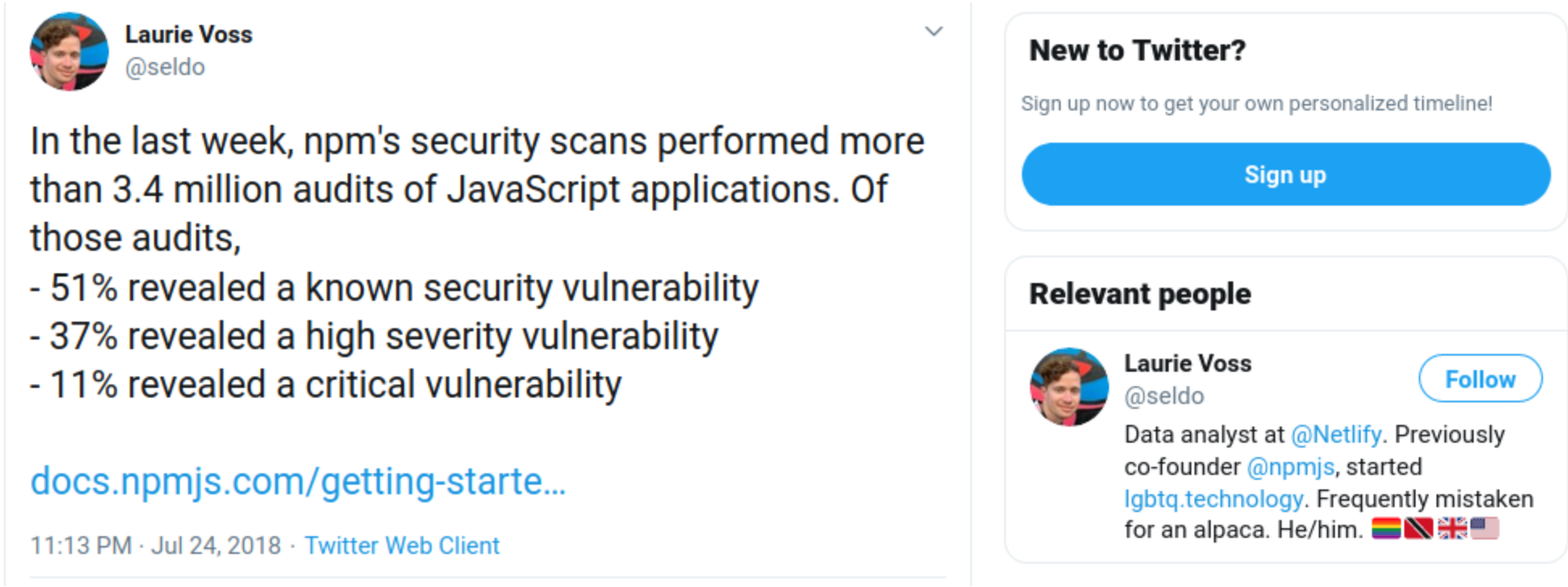
In A Nutshell

1. Modern applications depend on open source,
2. they contain many 3rd party components **and their vulnerabilities**.

"Reinvent The Wheel" by xkcd:
https://xkcd.com/2140/



The Problem



Vulnerabilities on NPM in 2018: https://twitter.com/seldo/status/1021865857813630976

The Problem

This problem has been recognized by the OWASP Top 10 Web Application Security Risks.

OWASP Top 10

> *"**#9 Using Components with Known Vulnerabilities.** Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts."* [owasp.org]

For more Information see:
Using_Components_with_Known_Vulnerabilities

Enter *OWASP Dependency Check*

A tool for mitigating OWASP Top 10 #9.

Checks dependencies for **Known Vulnerabilities**.

Developed by OWASP / Jeremy Long.

Full support for Java and .NET applications.

Experimental support for Python, Ruby, PHP and JavaScript/Node.js applications.

References:

- Project site on owasp.org
- Online documentation on github.io

Azure Pipeline Integration

Hosted Agents

Just add this tasks to your pipeline.

```yaml
- task: dependency-check-build-task@5
  displayName: 'Dependency Check: Run'
  inputs:
    projectName: MyProject        # name of the project
    scanPath: path/to/scanPath    # path of artifacts to scan
    failOnCVSS: 0                 # threshold when to fail build
    format: 'HTML'                # output format
    enableExperimental: false     # use experimental analyzers
    enableRetired: false          # use retired analyzers
    enableVerbose: false          # run in verbose mode
```

Azure Pipeline Integration

On-Premise Agents (1/2)

Dependency-Check requires JRE/JDK to run.

```yaml
- task: JavaToolInstaller@0
  displayName: 'Dependency Check: Install OpenJDK'
  inputs:
    versionSpec: "13"
    jdkArchitectureOption: x64
    jdkSourceOption: LocalDirectory
    jdkFile: "path/to/openjdk-13.0.2_windows-x64_bin.zip"
    jdkDestinationDirectory: "DependencyCheck/Binaries/Externals"
    cleanDestinationDirectory: true

- task: dependency-check-build-task@5
  ...
```

Azure Pipeline Integration

On-Premise Agents (2/2)

.NET Analyzers require .NET Core.

```yaml
- task: UseDotNet@2
  displayName: 'Dependency Check: Install .NET Core sdk'
  inputs:
    packageType: sdk
    version: 2.x
    installationPath: $(Agent.ToolsDirectory)/dotnet

- task: JavaToolInstaller@0
  ...

- task: dependency-check-build-task@5
  ...
```

Common Vulnerabilities and Exposures

Project Site: https://cve.mitre.org/

A system to identify publicly known vulnerabilities and exposures.

- **CVE Number**
  identifies publicly known vulnerabilities and exposures
- **CVE Numbering Authority (CNA)**
  assigns CVE Numbers
  e.g. The MITRE Corporation, Microsoft, Red Hat and others
- **CVE Number Syntax**
  `CVE prefix + Year + Arbitrary Digits`
- **CVE Number Example**
  CVE-2020-11022 (a jQuery XSS vulnerability)

Common Vulnerabilities and Exposures

CVE Numbers are only assigned to flaws which satisfy the following criteria.

A flaw must be:

1. **Independently Fixable**
   The flaw can be fixed independently of any other bugs.
2. **Acknowledged by the affected vendor** or **Documented**
   The flaw is either confirmed by the vendor or has a recorded prove.
3. **Affecting one codebase**
   The flaw may impact many products, e.g white-labeling, but resides in a single codebase.

For further information see:
https://www.redhat.com/en/topics/security/what-is-cve

Common Weakness Enumeration

Project Site: https://cwe.mitre.org/

A category system for software weaknesses and vulnerabilities.

The CWE system is a community project which aims to understand, identify, fix and prevent common security flaws in software and to create automated tools helping with these objectives.

- **CWE Number**
  identifies a category of known weaknesses or a concrete known weakness in software
- **CWE Number Syntax**
  `CWE prefix + Arbitrary Digits`
- **CWE Number Examples**
  - *CWE Category:* CWE-1211 Authentication Errors
  - *CWE Weakness:* CWE-295 Imporper Certificate Validation

The MITRE Corporation

The CVE and CWE systems are maintained and sponsored by The MITRE Corporation:

- Project Site: https://www.mitre.org/
- Non-Profit Organization
- Primary CNA
- Funded by various US Government institutions:
  - Dpt. of Homeland Security
  - Dpt. of Defense
  - Federal Aviation Administration
  - Internal Revenue Service
  - Department of Veterans Affairs.
  - National Institute of Standards and Technology
  - Administrative Office of the United States Courts
  - Centers for Medicare and Medicaid Services

FYI: "MITRE" has no meaning, although it originated around the Massachusetts Institute of Technology (MIT)

Common Vulnerability Scoring System

Project Site: https://www.first.org/cvss/

A system for calculating the serverity of vulnerabilities.

- **CVSS Vector**
  describes *exploitability* and *impact* of a vulnerability
  - **Example**
    `CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:C/C:H/I:L/A:N`
- **CVSS Score**
  describes the approximate serverity of a vulnerability
  floating point value between 0 (good) and 10 (bad)
  - **Example**
    `Base Score 6.9` (medium serverity)

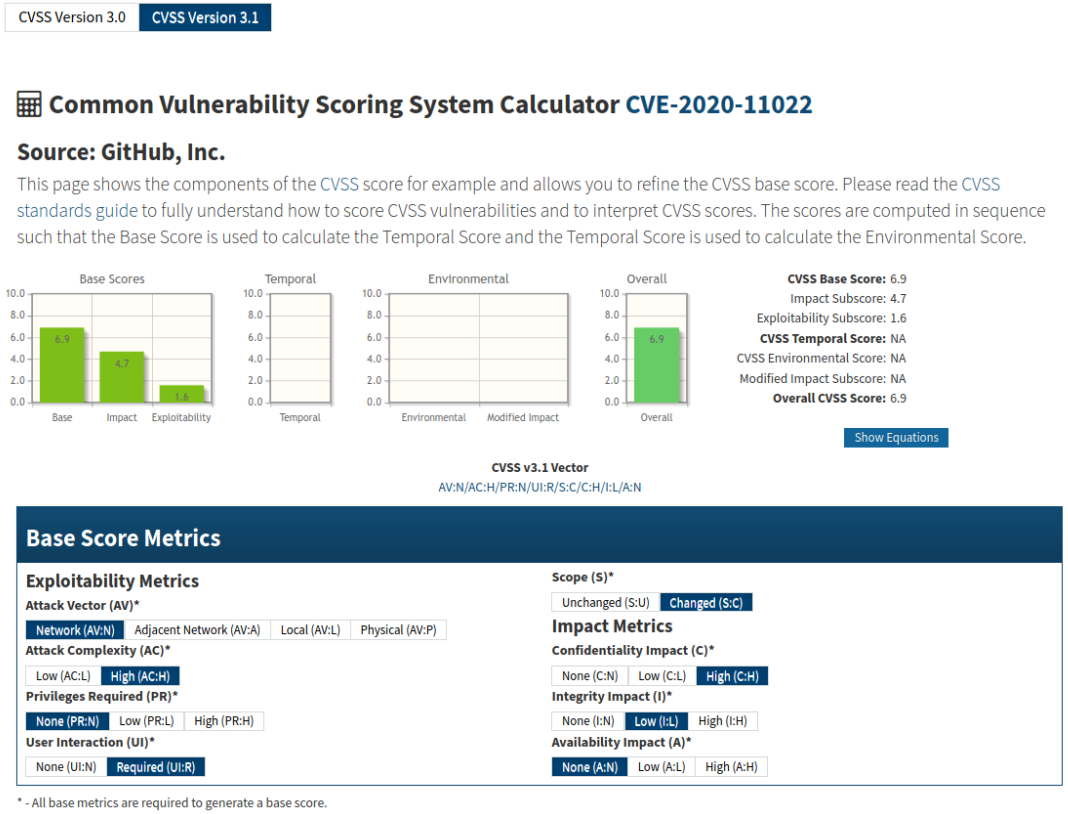
Common Vulnerability Scoring System

CVSS Vector: `CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:C/C:H/I:L/A:N`

CVSS Version: 3.1

Metric	Value	Category
Attack Vector (AV)	Network (N), Adjacent Network (A), Local (L), Physical (P)	Exploitability
Access Complexity (AC)	Low (L), High (H)	Exploitability
Privileges Required (PR)	None (N), Low (L), High (H)	Exploitability
User Interaction (UI)	None (N), Required (R)	Exploitability
(Authorization) Scope (S)	Unchanged (U), Changed (C)	Exploitability
Confidentiality Impact (C)	None (N), Low (L), High (H)	Impact
Integrity Impact (I)	None (N), Low (L), High (H)	Impact
Availability Impact (A)	None (N), Low (L), High (H)	Impact

Common Vulnerability Scoring System



This example of a CVSS calculation for CVE-2020-11022 can be found on NVD.

Common Platform Enumeration

Project Site: https://nvd.nist.gov/products/cpe

A naming system to uniquely identify information technology systems (hardware and software).

Originally developed be MITRE (https://cpe.mitre.org/), now part of the *Security Content Automation Protocol* (SCAP) maintained by the National Institute of Standards and Technology (NIST).

- **CPE Well-Formed Name (WFN)**
  uniquely identifies an information technology system (hardware or software)
- **CPE URI**
  represents a CPE WFN

NOTE: *CPE WFNs are not really important for working with Dependency Check.*

Common Platform Enumeration

- **CPE URI Syntax**
  - *CPE 2.2:*
    `cpe:/{part}:{vendor}:{product}:{version}:{update}:{edition}:{language}`
  - *CPE 2.3:*
    `cpe:2.3:{part}:{vendor}:{product}:{version}:{update}:{edition}:{language}:{sw_edition}:{target_sw}:{target_hw}:{other}`
- **CPE URI Examples**
  - *CPE 2.2:*
    `cpe:/a:jquery:jquery:1.0.1`
  - *CPE 2.3:*
    `cpe:2.3:a:jquery:jquery:1.0.1:*:*:*:*:*:*:*`

THE CPE Naming Specification Version 2.3 can be found here.

Common Platform Enumeration

Important CPE Sections

CPE 2.2:

```
cpe:/{part}:{vendor}:{product}:{version}
cpe:/a:jquery:jquery:1.0.1
```

CPE 2.3:

```
cpe:2.3:{part}:{vendor}:{product}:{version}
cpe:2.3:a:jquery:jquery:1.0.1:*:*:*:*:*:*:*
```

URI Section	Value	Description
part	a	applications (a), operating systems (o), hardware (h)
vendor	jquery	name of the vendor
product	jquery	name of the product
version	1.0.1	version of the product

How It Works

Reading Reports

False Positives

```yaml
- task: dependency-check-build-task@5
  displayName: 'Dependency Check: Run'
  inputs:
    ...
    suppressionPath: 'path/to/DependencyCheck/Supressions.xml' # add a supression file
    ...
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<suppressions xmlns="https://jeremylong.github.io/DependencyCheck/dependency-suppression.1.3.xsd">
    <suppress until="2020-01-01Z">
      <notes><![CDATA[
      file name: some.jar
      ]]></notes>
      <sha1>66734244CE86857018B023A8C56AE0635C56B6A1</sha1>
      <cpe>cpe:/a:apache:struts:2.0.0</cpe>
    </suppress>
</suppressions>
```

For further information see: Suppressing False Positives

False Negatives

```yaml
- task: dependency-check-build-task@5
  displayName: 'Dependency Check: Run'
  inputs:
    ...
    additionalArguments: '--hints "$(Build.SourcesDirectory)/path/to/DependencyCheck/Hints.xml"' # add a hints file
    ...
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<hints xmlns="https://jeremylong.github.io/DependencyCheck/dependency-hint.1.1.xsd">
    <hint>
        <given>
            <fileName contains="my-thelib-.*\.jar" regex="true" caseSensitive="true"/>
        </given>
        <add>
            <evidence type="product" source="hint analyzer" name="product" value="thelib" confidence="HIGH"/>
        </add>
  </hint>
</hints>
```

For further information see: [Resolving False Negatives]

Thanks!