# Live **Hacking**: **Breaking** into Your **Web** App

**Brian Vermeer**
**@BrianVerm**

snyk

# DevSecOps

@BrianVerm

snyk

# Brian Vermeer

**Developer Advocate**

**snyk**

@BrianVerm

- ✓ Java Champion
- ✓ Virtual JUG leader
- ✓ NLJUG leader
- ✓ DevSecCon co-leader
- ✓ Foojay Community Manager Security
- ✓ Top 21 Developers Shaping Tech

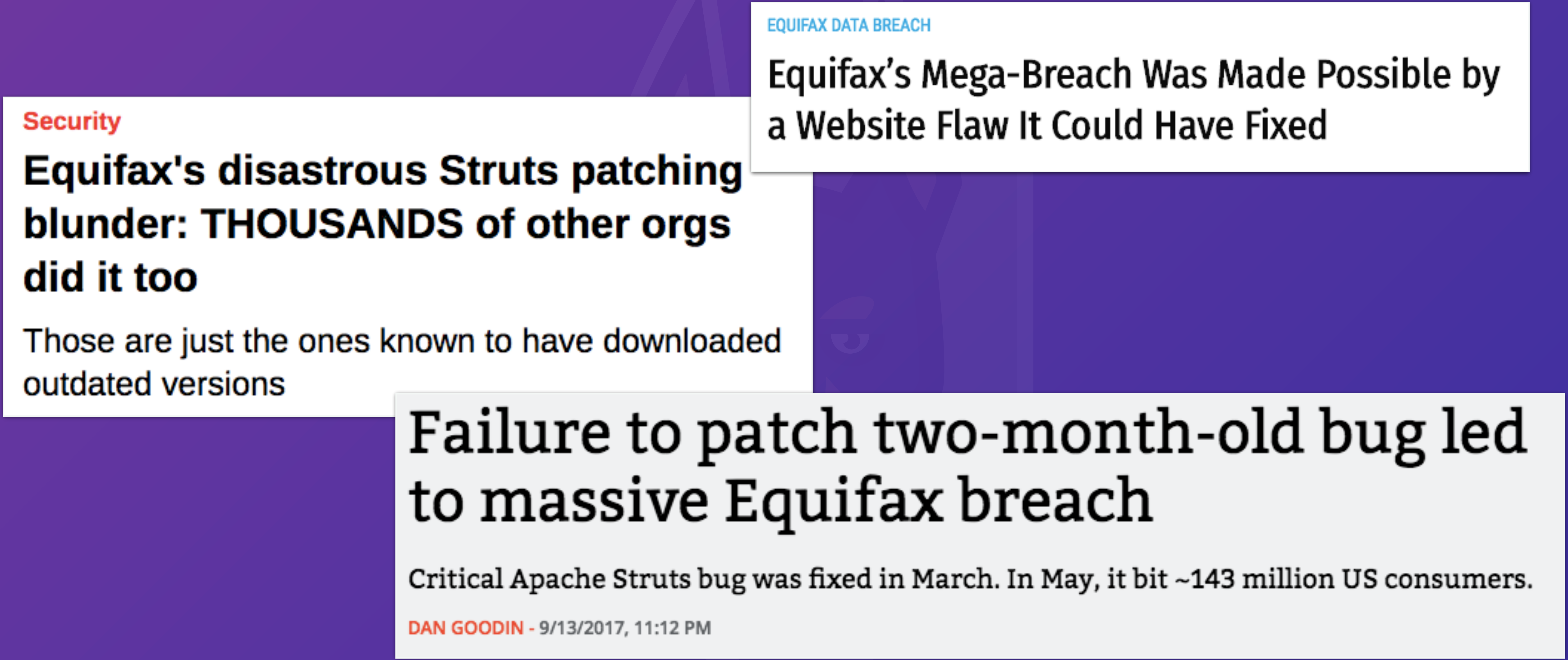
# What are the **Problems?**

1. Software delivery **sped up** with little thought to **security**
2. **Lack of security focus** throughout the app lifecycle
3. **Silo**-ed security expertise
4. **Customer data** could be compromised

@BrianVerm

snyk

# How **bad** is the **Situation?**



EQUIFAX DATA BREACH

## Equifax's Mega-Breach Was Made Possible by a Website Flaw It Could Have Fixed

Security

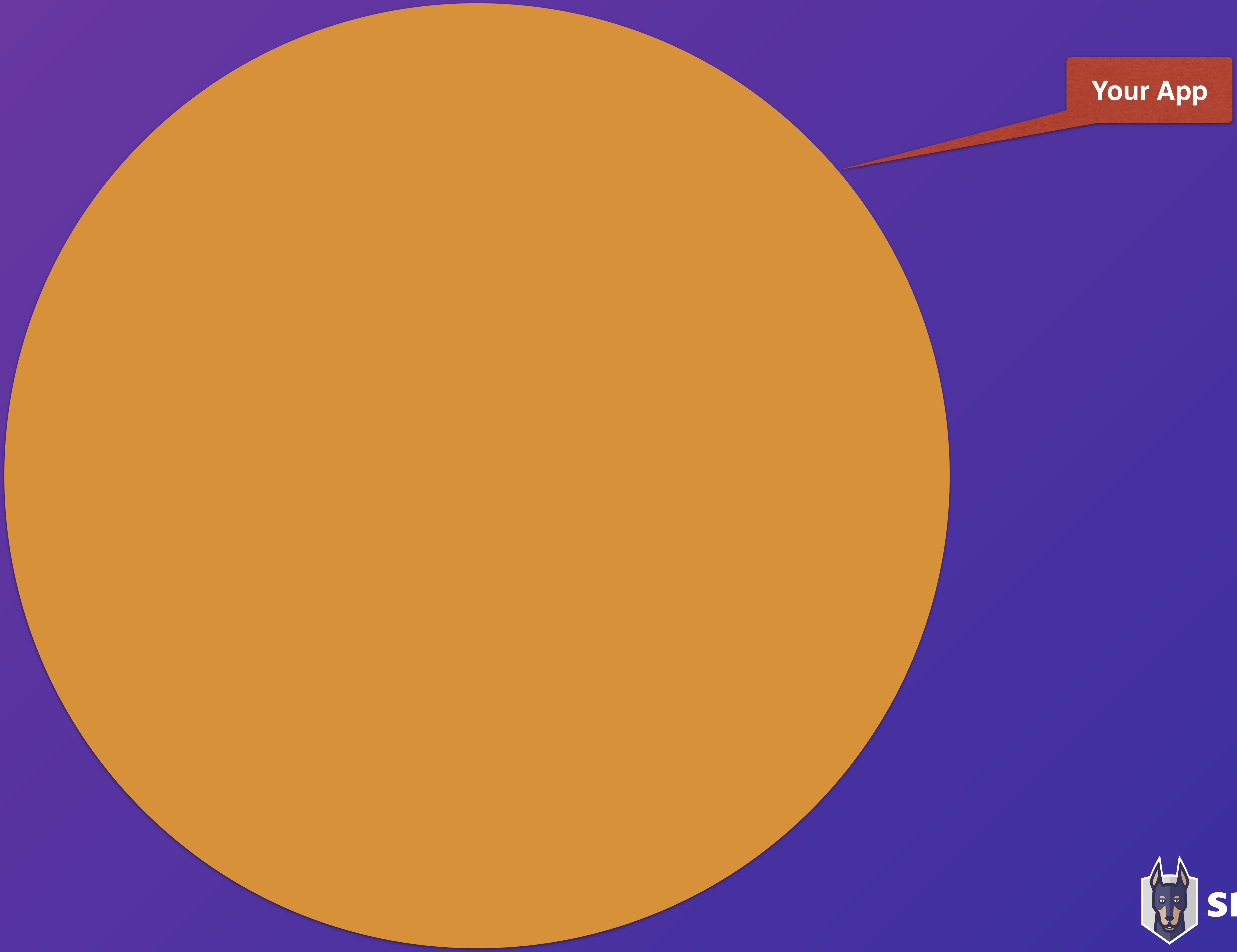
## Equifax's disastrous Struts patching blunder: THOUSANDS of other orgs did it too

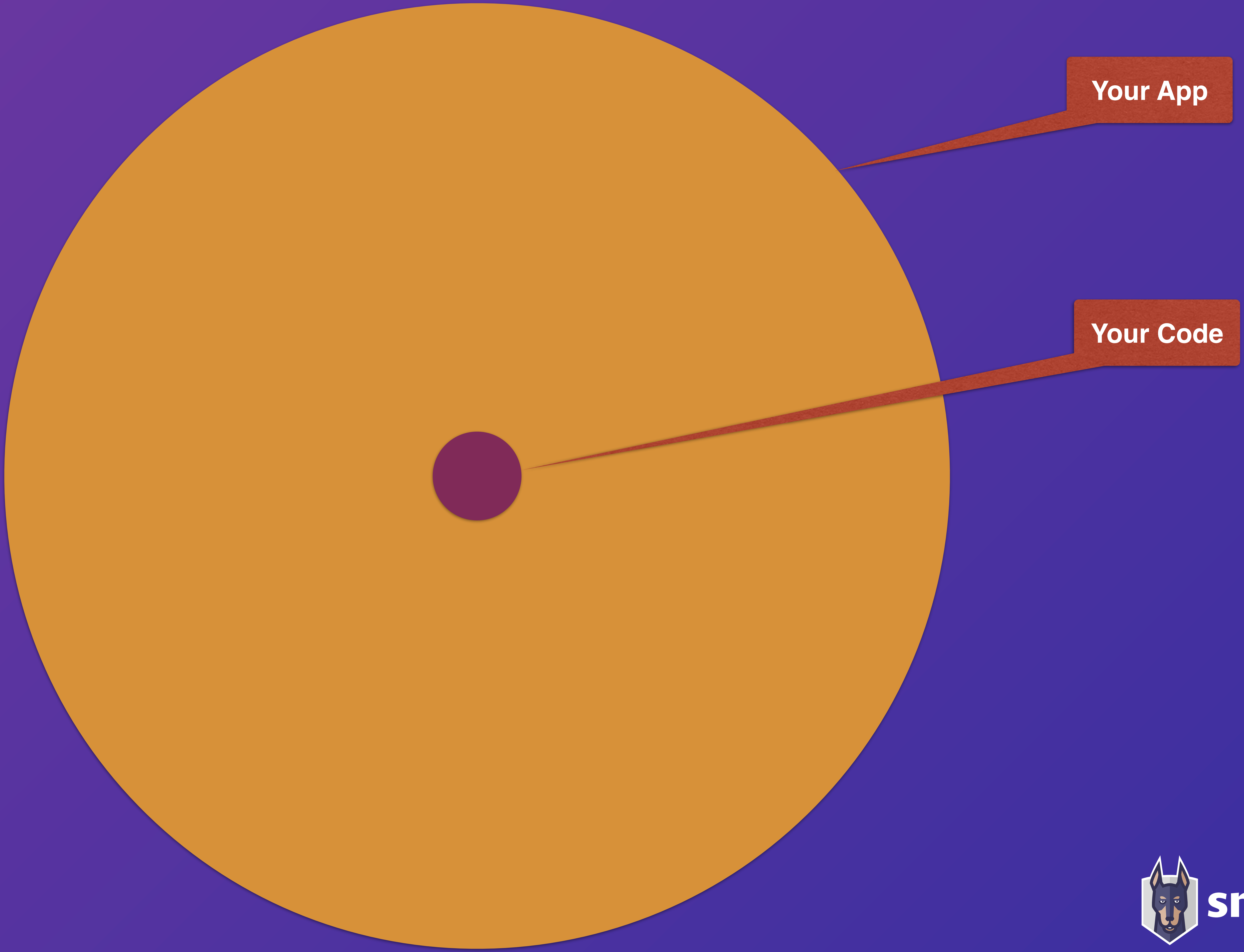
Those are just the ones known to have downloaded outdated versions

## Failure to patch two-month-old bug led to massive Equifax breach

Critical Apache Struts bug was fixed in March. In May, it bit ~143 million US consumers.

DAN GOODIN - 9/13/2017, 11:12 PM


Your App

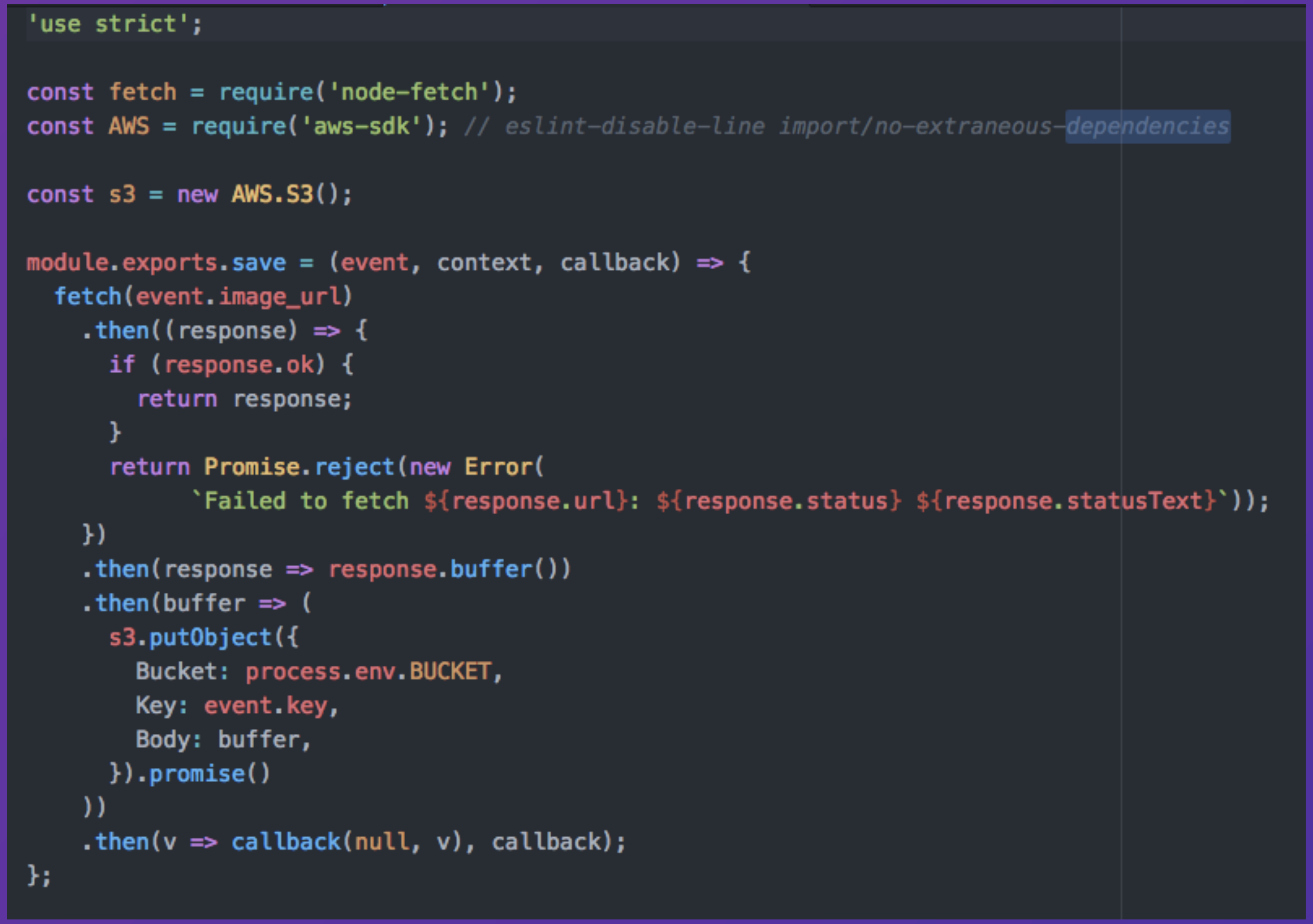


Your App

Your Code

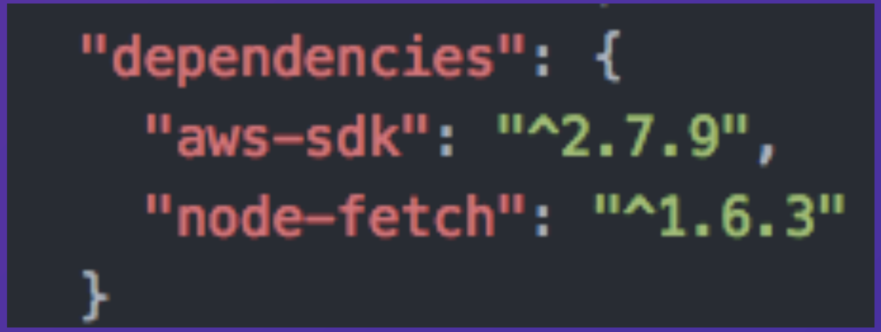
# Serverless Example: Fetch file & store in s3

(Serverless Framework Example)



```
'use strict';

const fetch = require('node-fetch');
const AWS = require('aws-sdk'); // eslint-disable-line import/no-extraneous-dependencies

const s3 = new AWS.S3();

module.exports.save = (event, context, callback) => {
  fetch(event.image_url)
    .then((response) => {
      if (response.ok) {
        return response;
      }
      return Promise.reject(new Error(
            `Failed to fetch ${response.url}: ${response.status} ${response.statusText}`));
    })
    .then(response => response.buffer())
    .then(buffer => (
      s3.putObject({
        Bucket: process.env.BUCKET,
        Key: event.key,
        Body: buffer,
      }).promise()
    ))
    .then(v => callback(null, v), callback);
};
```

19 Lines of Code



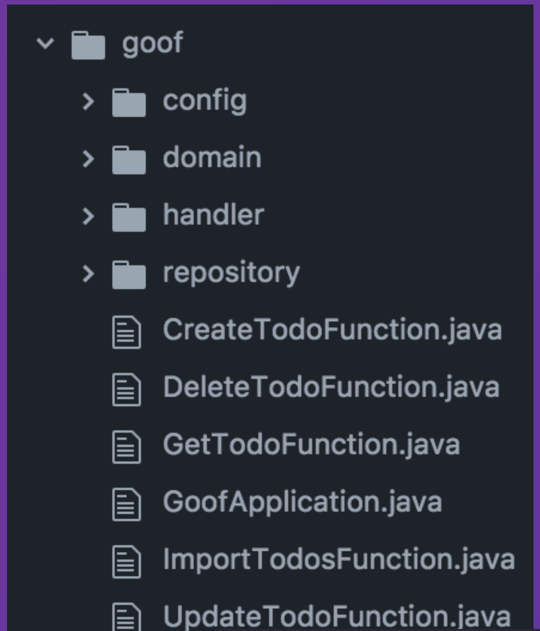
```
"dependencies": {
  "aws-sdk": "^2.7.9",
  "node-fetch": "^1.6.3"
}
```

2 Direct dependencies

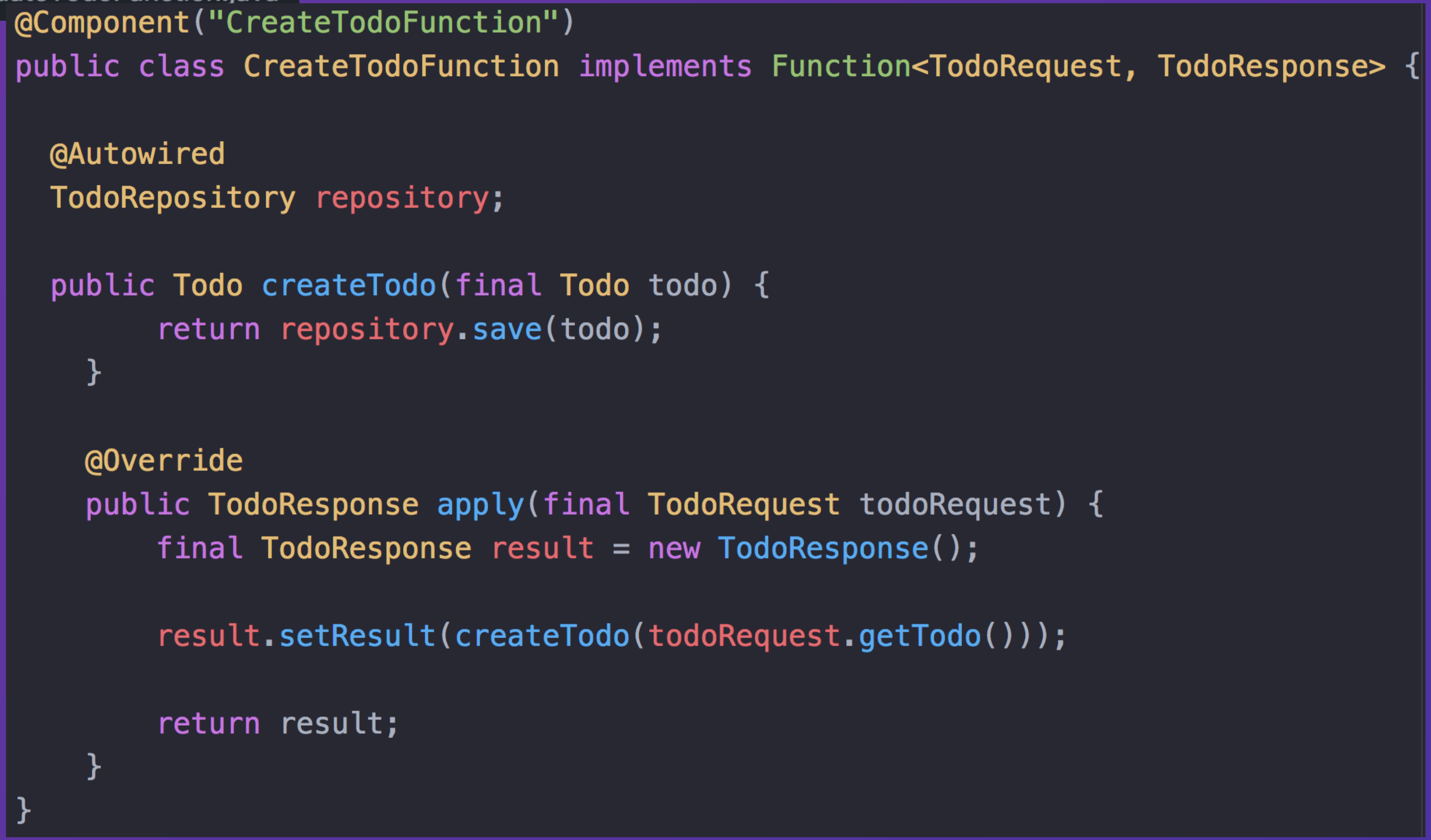
19 dependencies (incl. indirect)

191,155 Lines of Code

# Spring Serverless Example

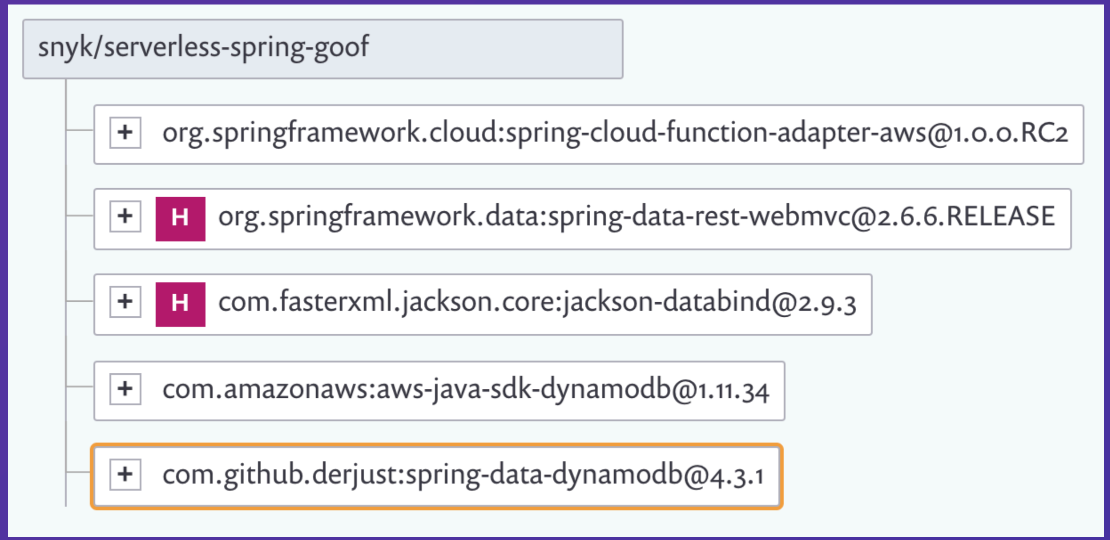


```
goof
  config
  domain
  handler
  repository
  CreateTodoFunction.java
  DeleteTodoFunction.java
  GetTodoFunction.java
  GoofApplication.java
  ImportTodosFunction.java
  UpdateTodoFunction.java
```



```java
@Component("CreateTodoFunction")
public class CreateTodoFunction implements Function<TodoRequest, TodoResponse> {

  @Autowired
  TodoRepository repository;

  public Todo createTodo(final Todo todo) {
        return repository.save(todo);
    }

    @Override
    public TodoResponse apply(final TodoRequest todoRequest) {
        final TodoResponse result = new TodoResponse();

        result.setResult(createTodo(todoRequest.getTodo()));

        return result;
    }
}
```

222 Lines of Code



snyk/serverless-spring-goof

- + org.springframework.cloud:spring-cloud-function-adapter-aws@1.0.0.RC2
- + H org.springframework.data:spring-data-rest-webmvc@2.6.6.RELEASE
- + H com.fasterxml.jackson.core:jackson-databind@2.9.3
- + com.amazonaws:aws-java-sdk-dynamodb@1.11.34
- + com.github.derjust:spring-data-dynamodb@4.3.1

5 Direct dependencies

54 dependencies (incl. indirect)

460,046 Lines of Code

@BrianVerm

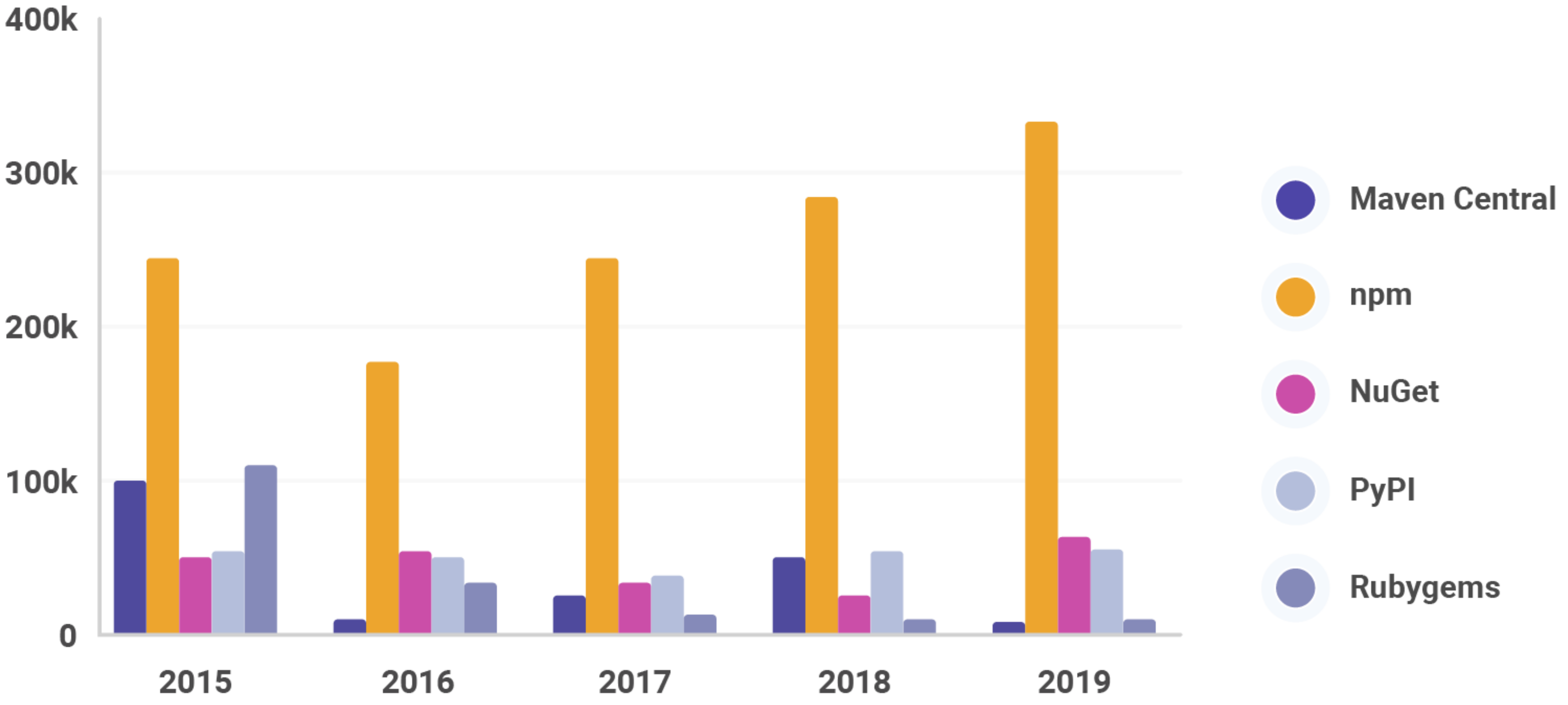
snyk

# Open Source Usage Has Exploded

# Attackers Are Targeting Open Source

One vulnerability, many victims

# New packages created by ecosystem per year

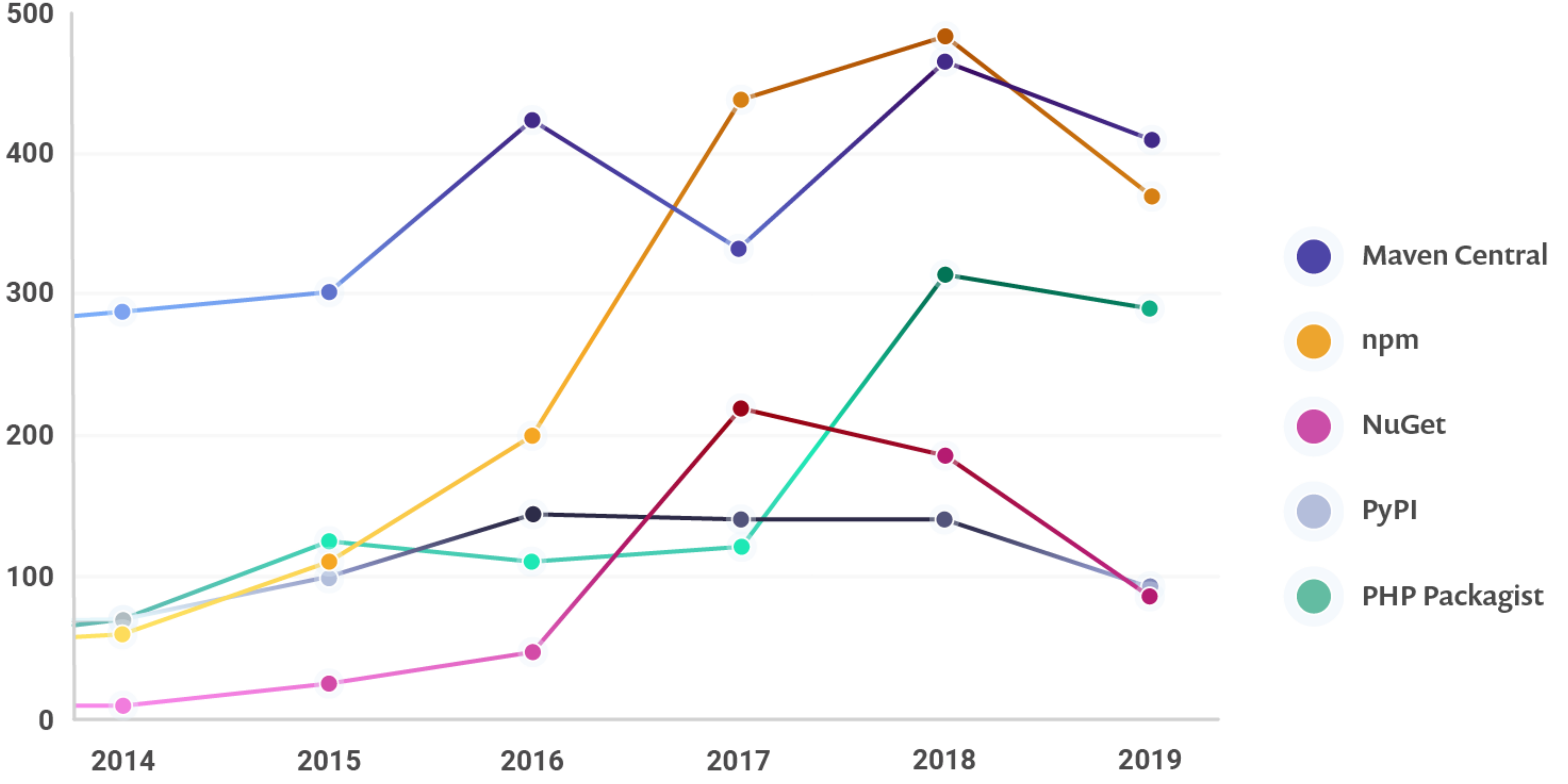


https://info.snyk.io/sooss-report-2020

@BrianVerm

# Vulnerabilities identified in ecosystems since 2014

snyk

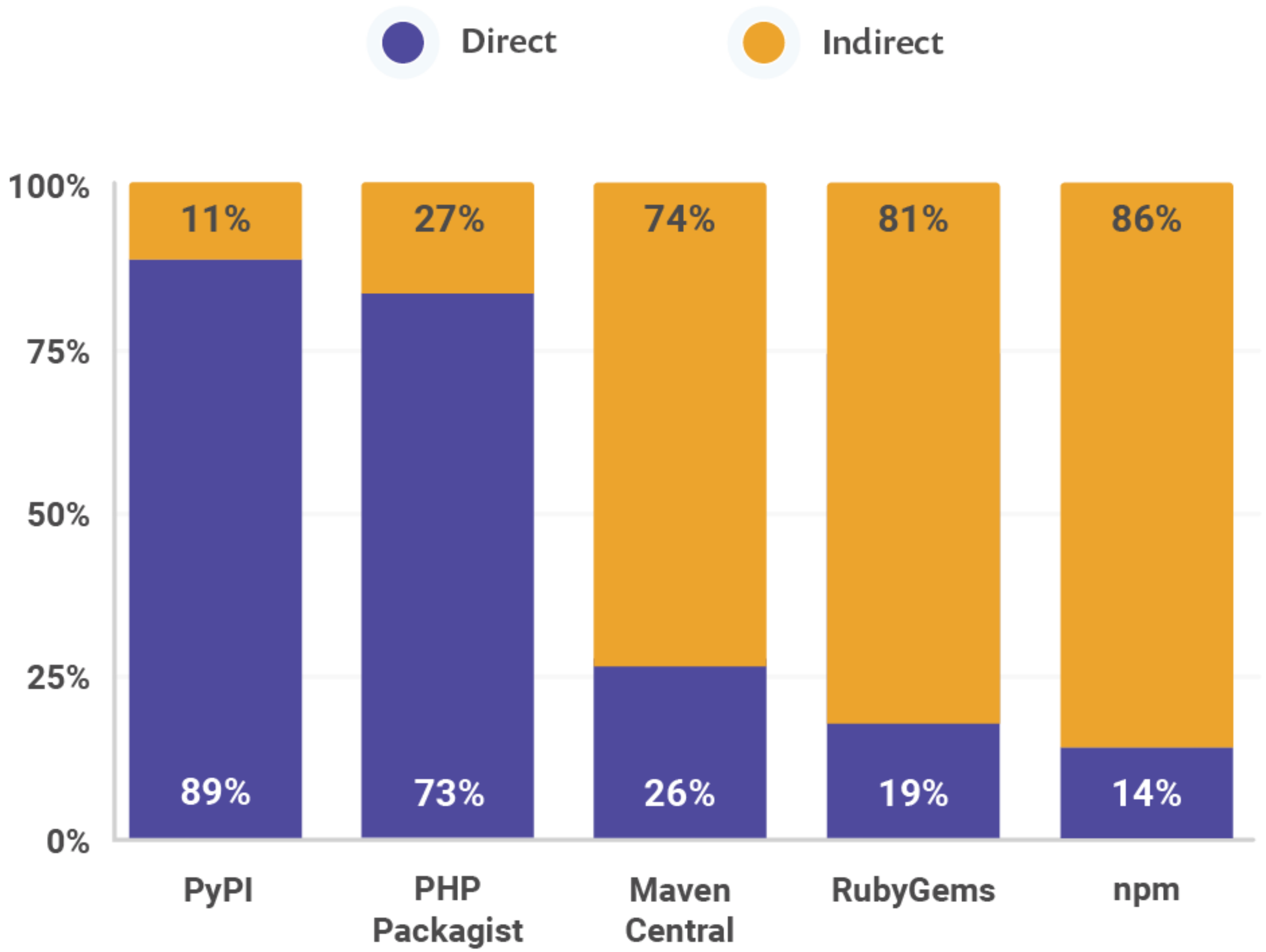


https://info.snyk.io/sooss-report-2020

@BrianVerm

snyk

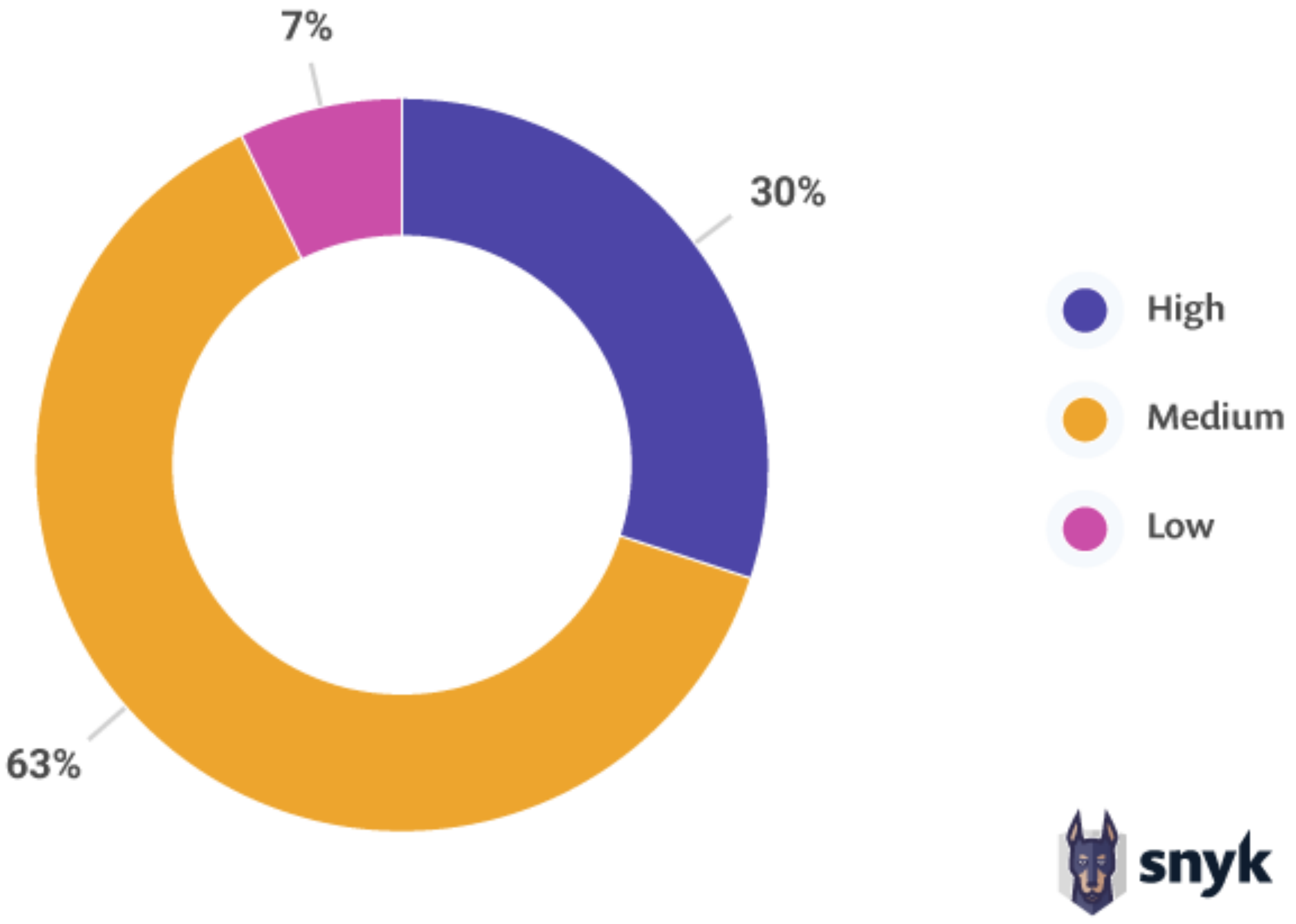
# Vulnerabilities from direct versus indirect dependencies



@BrianVerm

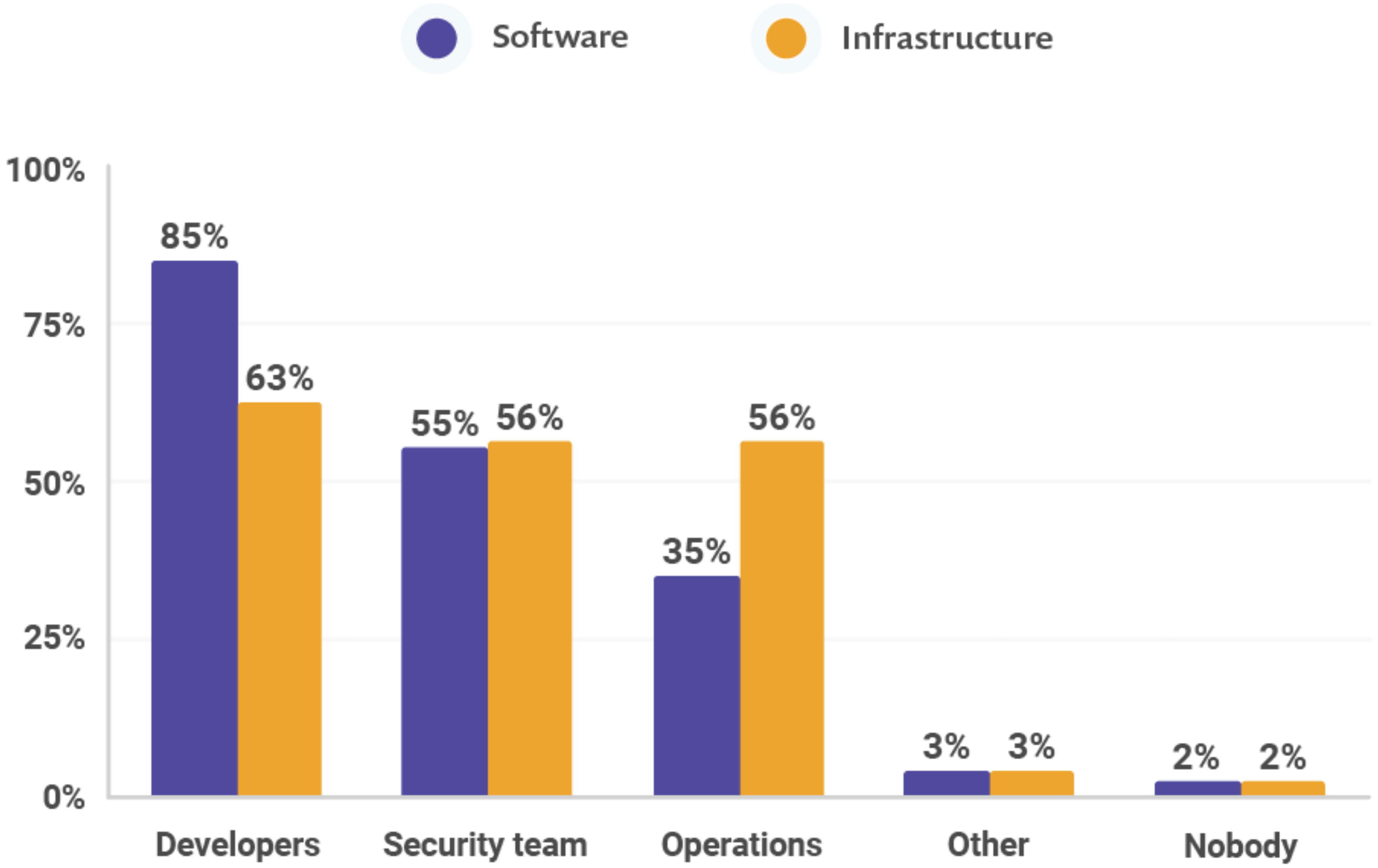
https://info.snyk.io/sooss-report-2020

# OS maintainers are confident in their own security knowledge



@BrianVerm

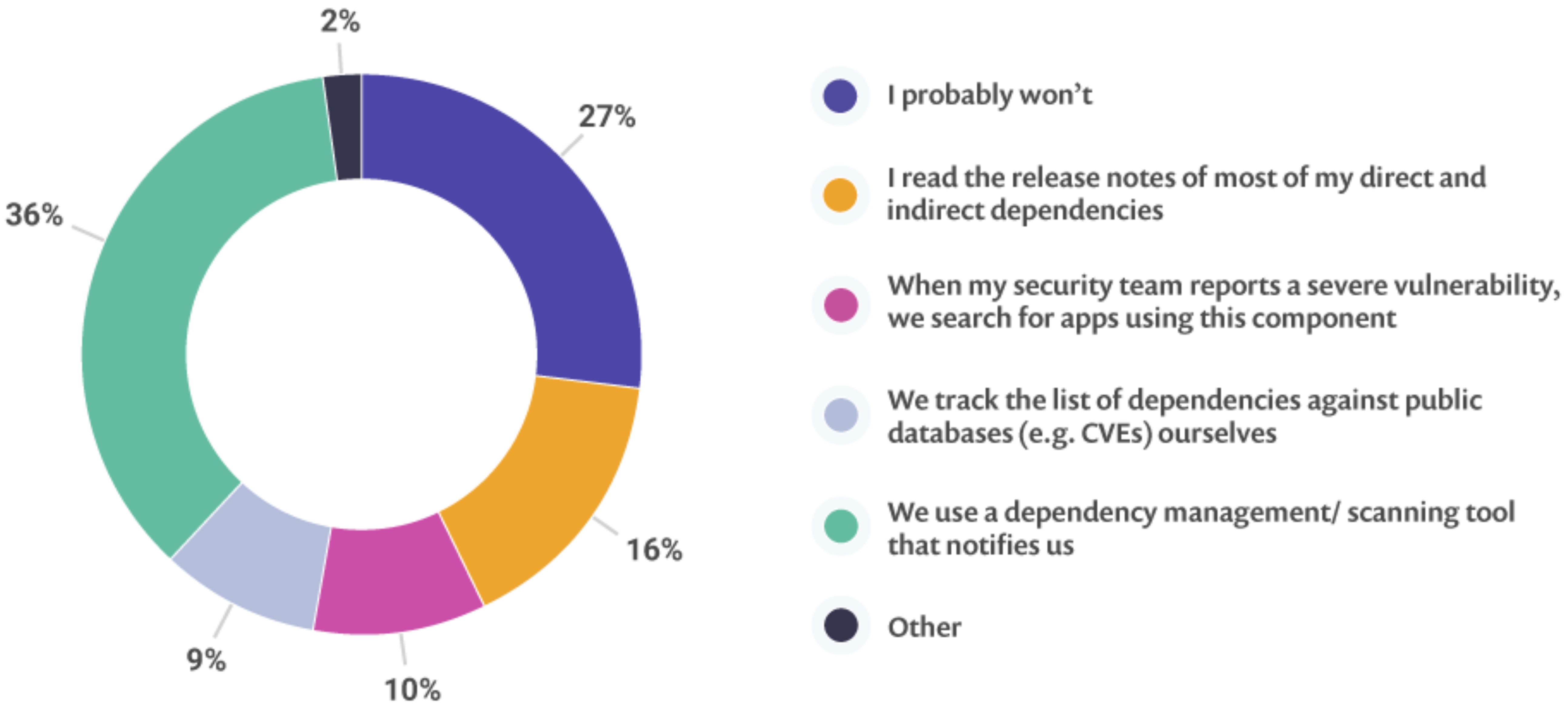
# Who should be responsible for security?



@BrianVerm

https://info.snyk.io/sooss-report-2020

# How do you find about vulnerabilities?



# Vulnerabilities generally remain undiscovered for a long time.

The median time from inclusion to discovery for in application libraries:

@BrianVerm

# Vulnerabilities generally remain undiscovered for a long time.

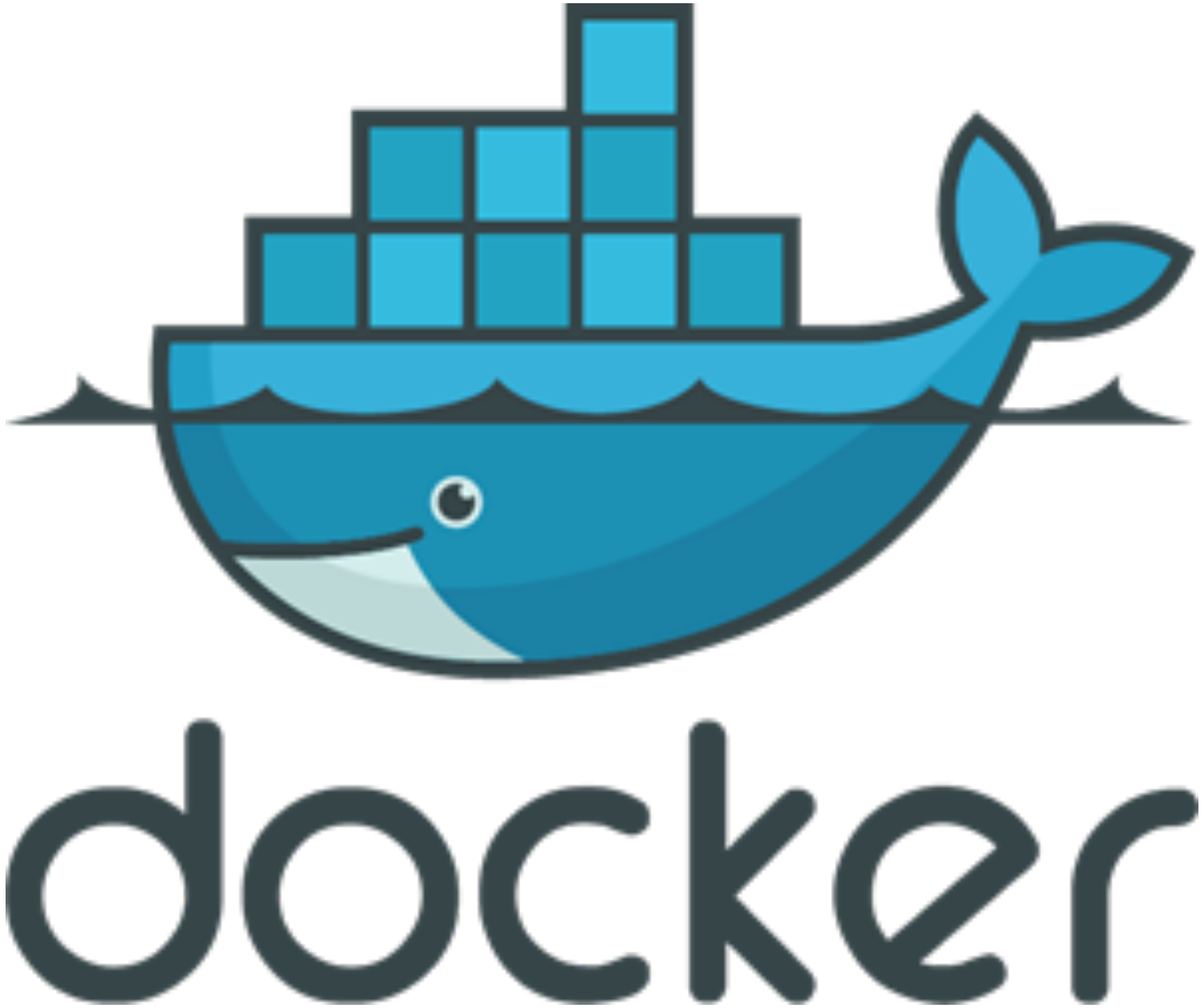
The median time from inclusion to discovery for in application libraries:

**2.5 years**

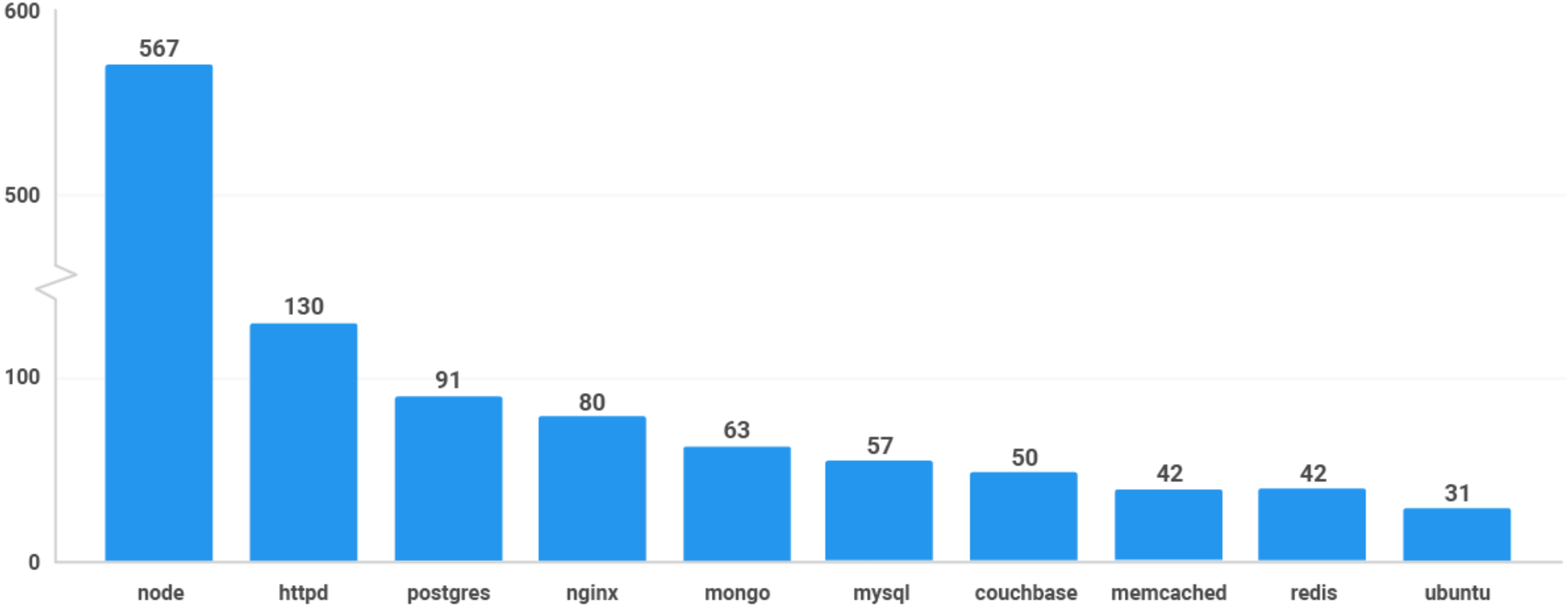
# Let’s HACK!

@BrianVerm

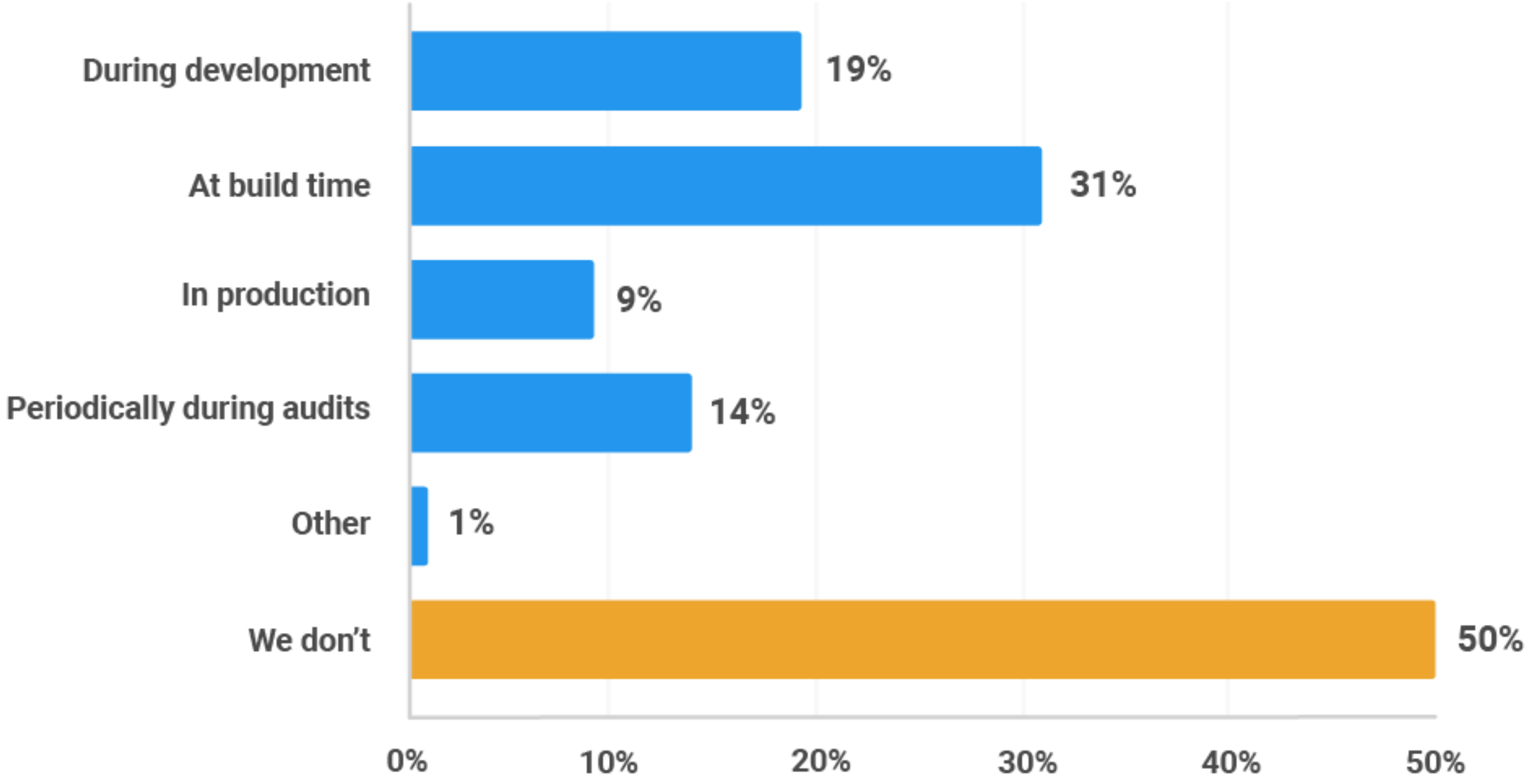
snyk


docker

# Vulnerabilities per Docker image

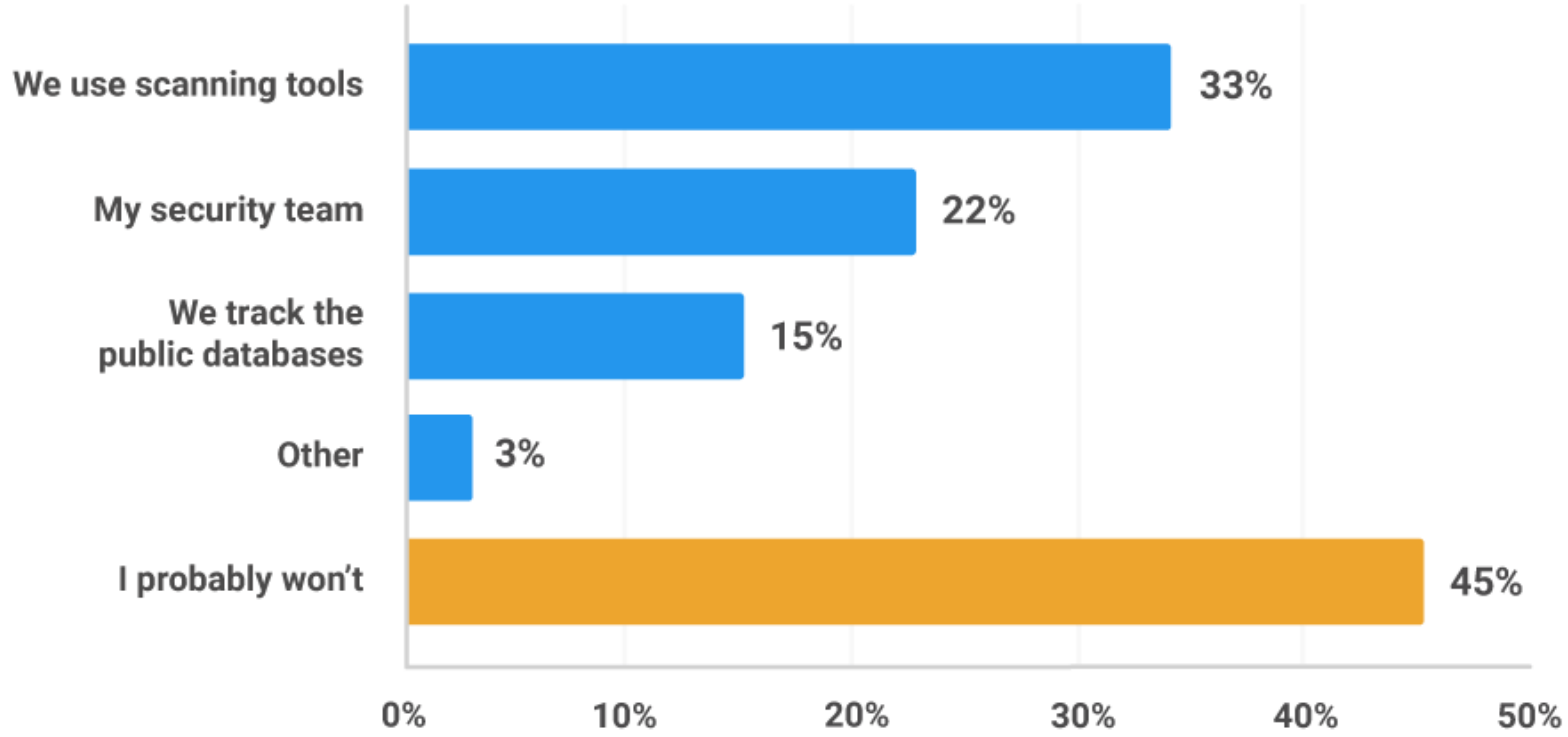


# When do you scan your Docker image for OS vulns?



@BrianV

# How do you find out about new vulnerabilities in your deployed containers?



@BrianV

# Let’s HACK!

@BrianVerm

snyk

# What’s the **Solution?**

**Culture**

**Process**

**Tooling**

# Culture

What do people care about?

- Developers
- Operations
- Security
- Management

# Process

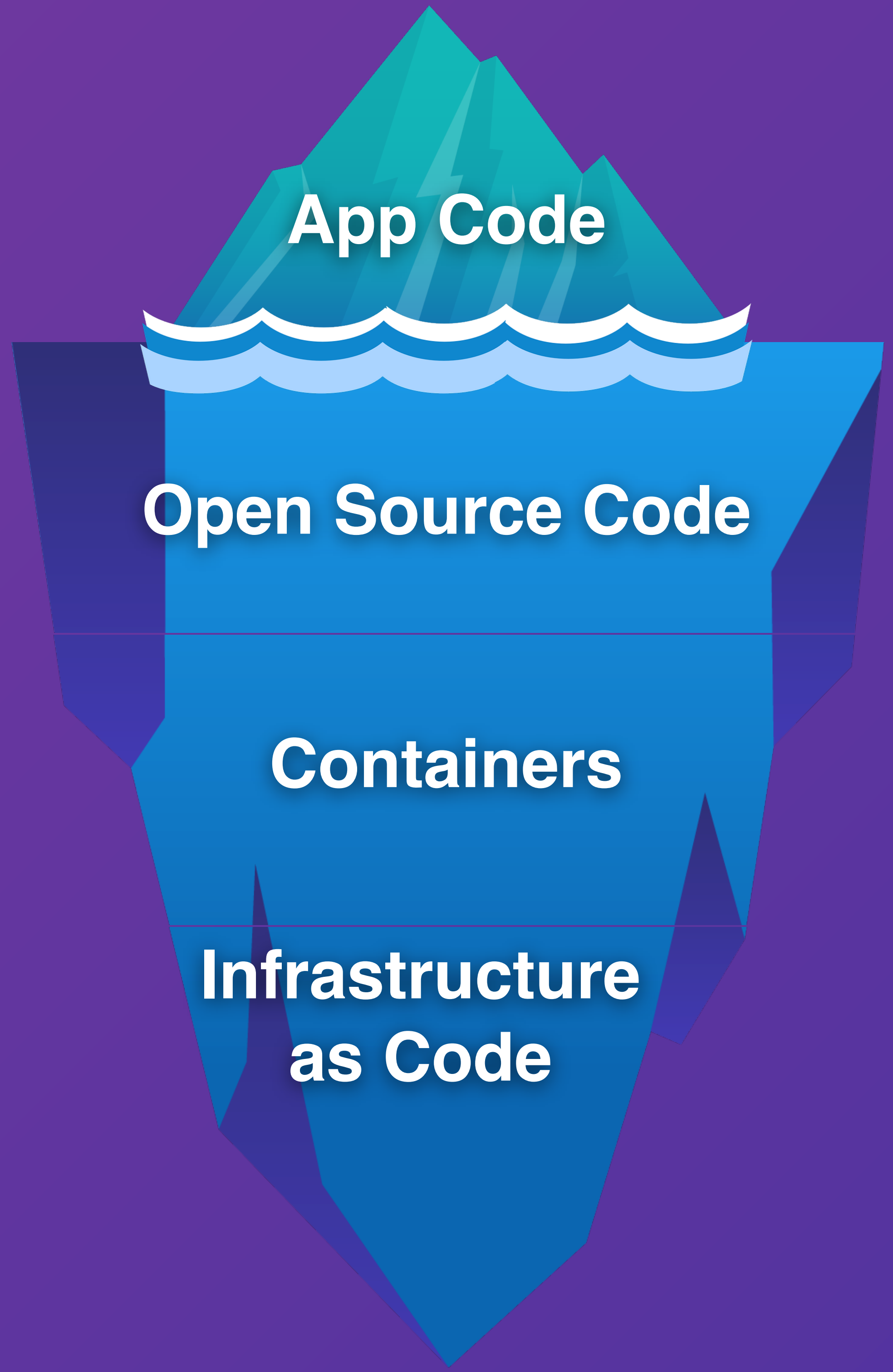
The **best** way to adopt a **new practice** is to **integrate** it into existing processes, **not create more**.

# Tooling

Tooling can **help**

**Automate** away **manual** steps

**Alert** you to issues **when** they happen



Snyk Code

Snyk Open Source

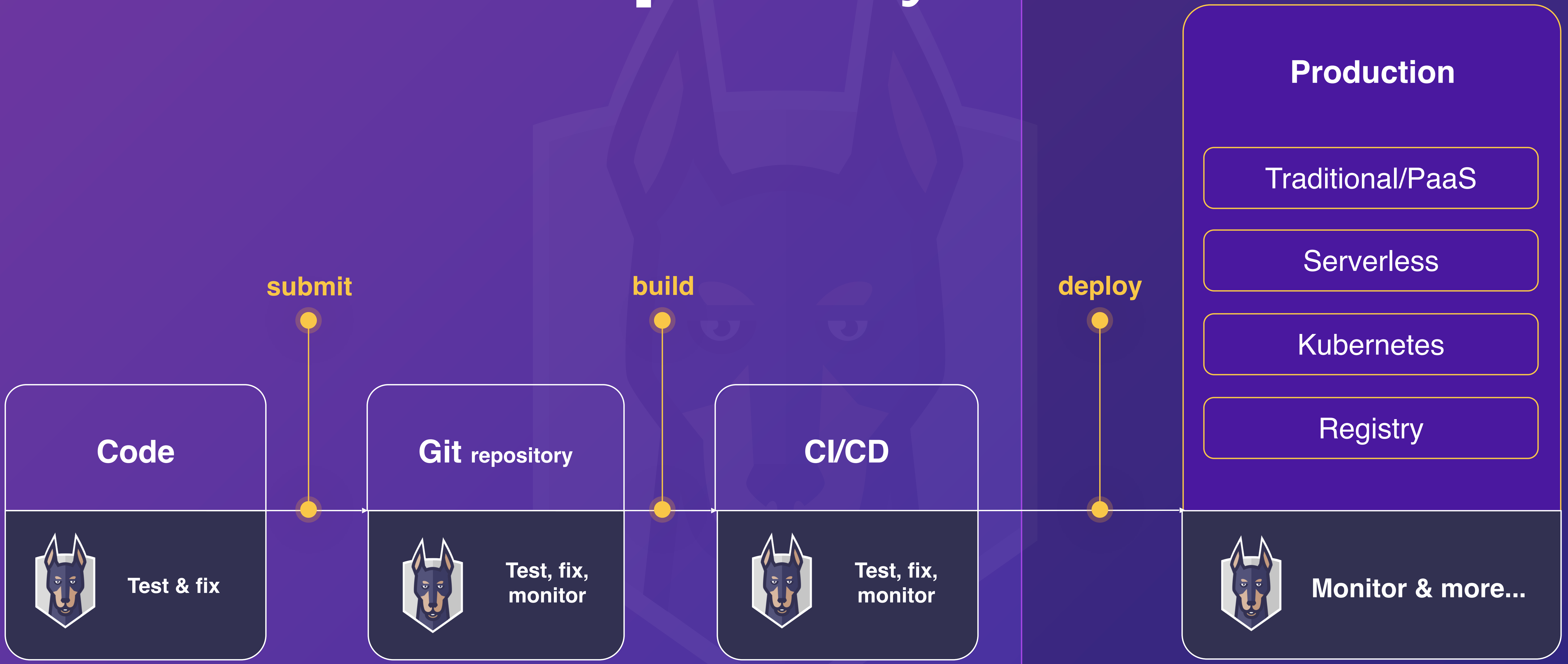
Snyk Container

Snyk IaC

@BrianVerm

snyk

# DevSecOps in your SDLC



Code

Test & fix

submit

Git repository

Test, fix, monitor

build

CI/CD

Test, fix, monitor

deploy

Production

Traditional/PaaS

Serverless

Kubernetes

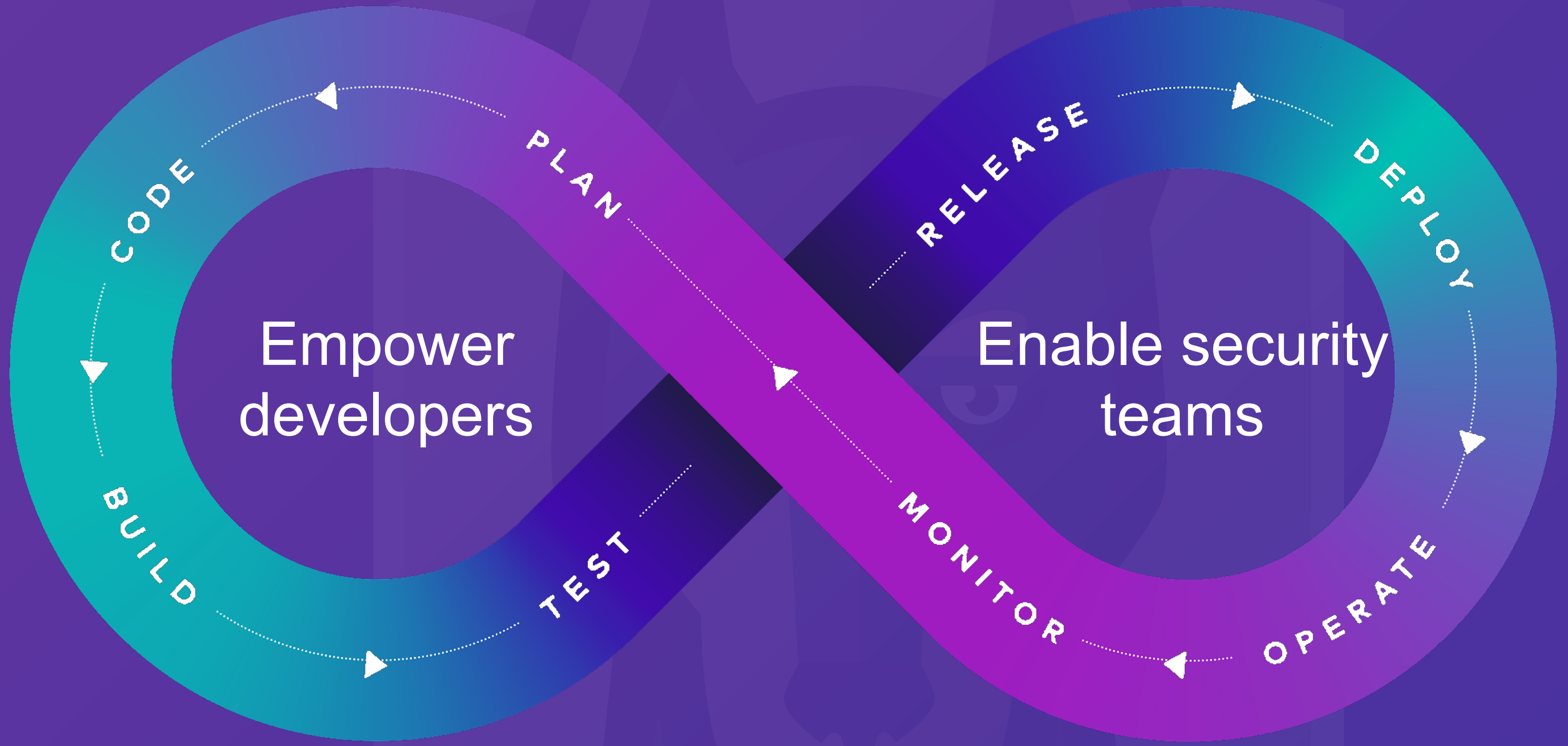
Registry

Monitor & more...

@BrianVerm

snyk

# “Shift left” is not enough



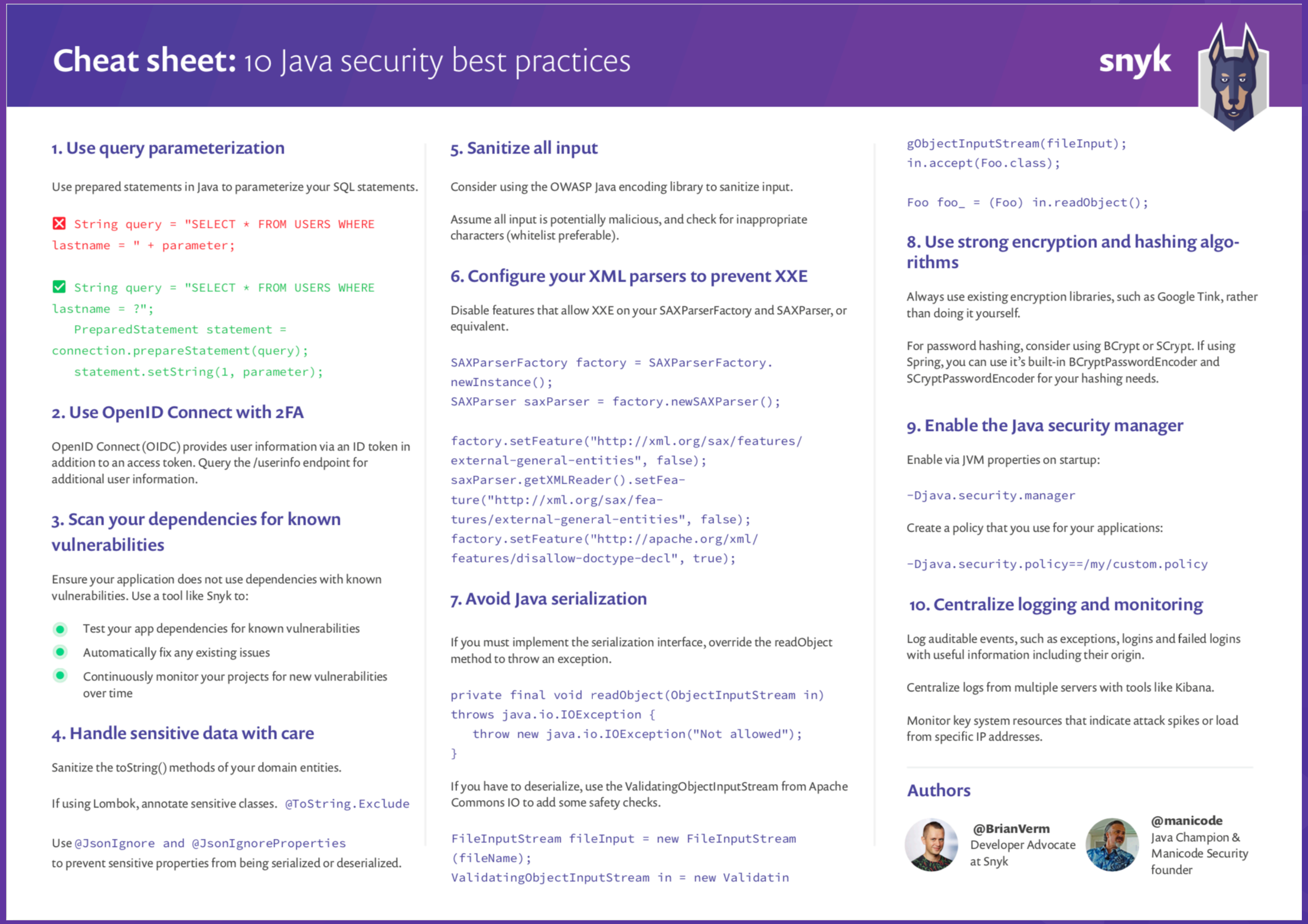
Empowering developers to build applications securely within the entire development process

@BrianVerm

snyk

http://bit.ly/java-security

http://bit.ly/npm-sec



## Cheat sheet: 10 Java security best practices

snyk

### 1. Use query parameterization

Use prepared statements in Java to parameterize your SQL statements.

```
☒ String query = "SELECT * FROM USERS WHERE
lastname = " + parameter;
```

```
☑ String query = "SELECT * FROM USERS WHERE
lastname = ?";
   PreparedStatement statement =
connection.prepareStatement(query);
   statement.setString(1, parameter);
```

### 2. Use OpenID Connect with 2FA

OpenID Connect (OIDC) provides user information via an ID token in addition to an access token. Query the /userinfo endpoint for additional user information.

### 3. Scan your dependencies for known vulnerabilities

Ensure your application does not use dependencies with known vulnerabilities. Use a tool like Snyk to:

- ● Test your app dependencies for known vulnerabilities
- ● Automatically fix any existing issues
- ● Continuously monitor your projects for new vulnerabilities over time

### 4. Handle sensitive data with care

Sanitize the toString() methods of your domain entities.

If using Lombok, annotate sensitive classes. `@ToString.Exclude`

Use `@JsonIgnore and @JsonIgnoreProperties` to prevent sensitive properties from being serialized or deserialized.

### 5. Sanitize all input

Consider using the OWASP Java encoding library to sanitize input.

Assume all input is potentially malicious, and check for inappropriate characters (whitelist preferable).

### 6. Configure your XML parsers to prevent XXE

Disable features that allow XXE on your SAXParserFactory and SAXParser, or equivalent.

```
SAXParserFactory factory = SAXParserFactory.
newInstance();
SAXParser saxParser = factory.newSAXParser();

factory.setFeature("http://xml.org/sax/features/
external-general-entities", false);
saxParser.getXMLReader().setFea-
ture("http://xml.org/sax/fea-
tures/external-general-entities", false);
factory.setFeature("http://apache.org/xml/
features/disallow-doctype-decl", true);
```

### 7. Avoid Java serialization

If you must implement the serialization interface, override the readObject method to throw an exception.

```
private final void readObject(ObjectInputStream in)
throws java.io.IOException {
   throw new java.io.IOException("Not allowed");
}
```

If you have to deserialize, use the ValidatingObjectInputStream from Apache Commons IO to add some safety checks.

```
FileInputStream fileInput = new FileInputStream
(fileName);
ValidatingObjectInputStream in = new ValidatingObjectInputStream(fileInput);
in.accept(Foo.class);

Foo foo_ = (Foo) in.readObject();
```

### 8. Use strong encryption and hashing algorithms

Always use existing encryption libraries, such as Google Tink, rather than doing it yourself.

For password hashing, consider using BCrypt or SCrypt. If using Spring, you can use it's built-in BCryptPasswordEncoder and SCryptPasswordEncoder for your hashing needs.

### 9. Enable the Java security manager

Enable via JVM properties on startup:

```
-Djava.security.manager
```

Create a policy that you use for your applications:

```
-Djava.security.policy==/my/custom.policy
```

### 10. Centralize logging and monitoring

Log auditable events, such as exceptions, logins and failed logins with useful information including their origin.

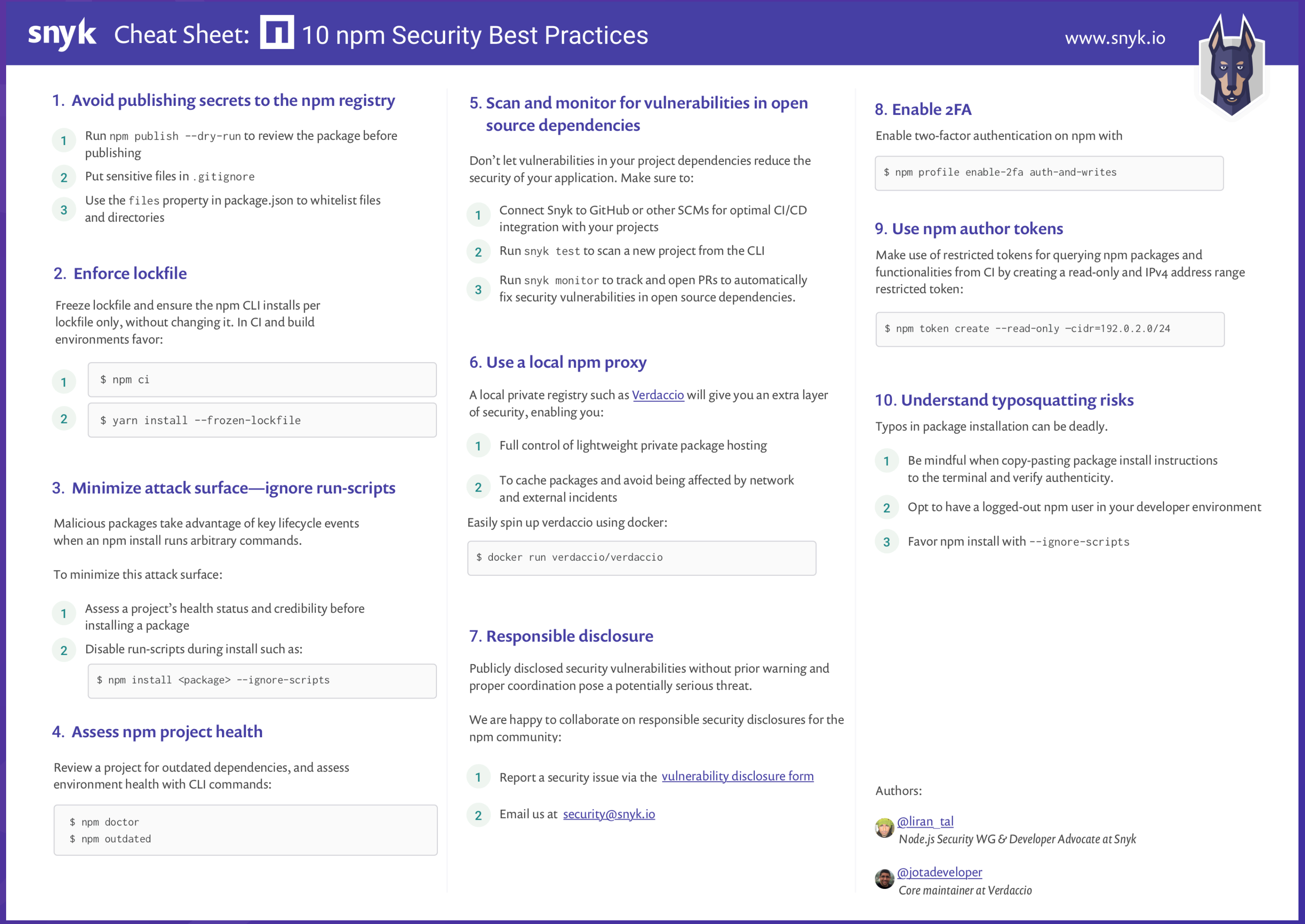
Centralize logs from multiple servers with tools like Kibana.

Monitor key system resources that indicate attack spikes or load from specific IP addresses.

### Authors

**@BrianVerm**
Developer Advocate
at Snyk

**@manicode**
Java Champion &
Manicode Security
founder



## snyk Cheat Sheet: 10 npm Security Best Practices

www.snyk.io

### 1. Avoid publishing secrets to the npm registry

1 Run `npm publish --dry-run` to review the package before publishing

2 Put sensitive files in `.gitignore`

3 Use the `files` property in package.json to whitelist files and directories

### 2. Enforce lockfile

Freeze lockfile and ensure the npm CLI installs per lockfile only, without changing it. In CI and build environments favor:

1
```
$ npm ci
```

2
```
$ yarn install --frozen-lockfile
```

### 3. Minimize attack surface—ignore run-scripts

Malicious packages take advantage of key lifecycle events when an npm install runs arbitrary commands.

To minimize this attack surface:

1 Assess a project's health status and credibility before installing a package

2 Disable run-scripts during install such as:

```
$ npm install <package> --ignore-scripts
```

### 4. Assess npm project health

Review a project for outdated dependencies, and assess environment health with CLI commands:

```
$ npm doctor
$ npm outdated
```

### 5. Scan and monitor for vulnerabilities in open source dependencies

Don't let vulnerabilities in your project dependencies reduce the security of your application. Make sure to:

1 Connect Snyk to GitHub or other SCMs for optimal CI/CD integration with your projects

2 Run `snyk test` to scan a new project from the CLI

3 Run `snyk monitor` to track and open PRs to automatically fix security vulnerabilities in open source dependencies.

### 6. Use a local npm proxy

A local private registry such as Verdaccio will give you an extra layer of security, enabling you:

1 Full control of lightweight private package hosting

2 To cache packages and avoid being affected by network and external incidents

Easily spin up verdaccio using docker:

```
$ docker run verdaccio/verdaccio
```

### 7. Responsible disclosure

Publicly disclosed security vulnerabilities without prior warning and proper coordination pose a potentially serious threat.

We are happy to collaborate on responsible security disclosures for the npm community:

1 Report a security issue via the vulnerability disclosure form

2 Email us at security@snyk.io

### 8. Enable 2FA

Enable two-factor authentication on npm with

```
$ npm profile enable-2fa auth-and-writes
```

### 9. Use npm author tokens

Make use of restricted tokens for querying npm packages and functionalities from CI by creating a read-only and IPv4 address range restricted token:

```
$ npm token create --read-only –cidr=192.0.2.0/24
```

### 10. Understand typosquatting risks

Typos in package installation can be deadly.

1 Be mindful when copy-pasting package install instructions to the terminal and verify authenticity.

2 Opt to have a logged-out npm user in your developer environment

3 Favor npm install with `--ignore-scripts`

Authors:

@liran_tal
*Node.js Security WG & Developer Advocate at Snyk*

@jotadeveloper
*Core maintainer at Verdaccio*

http://snyk.io/blog

@BrianVerm

snyk

# Questions?

**BRIAN VERMEER**

BRIANVERMEER@SNYK.IO

**@BRIANVERM**