

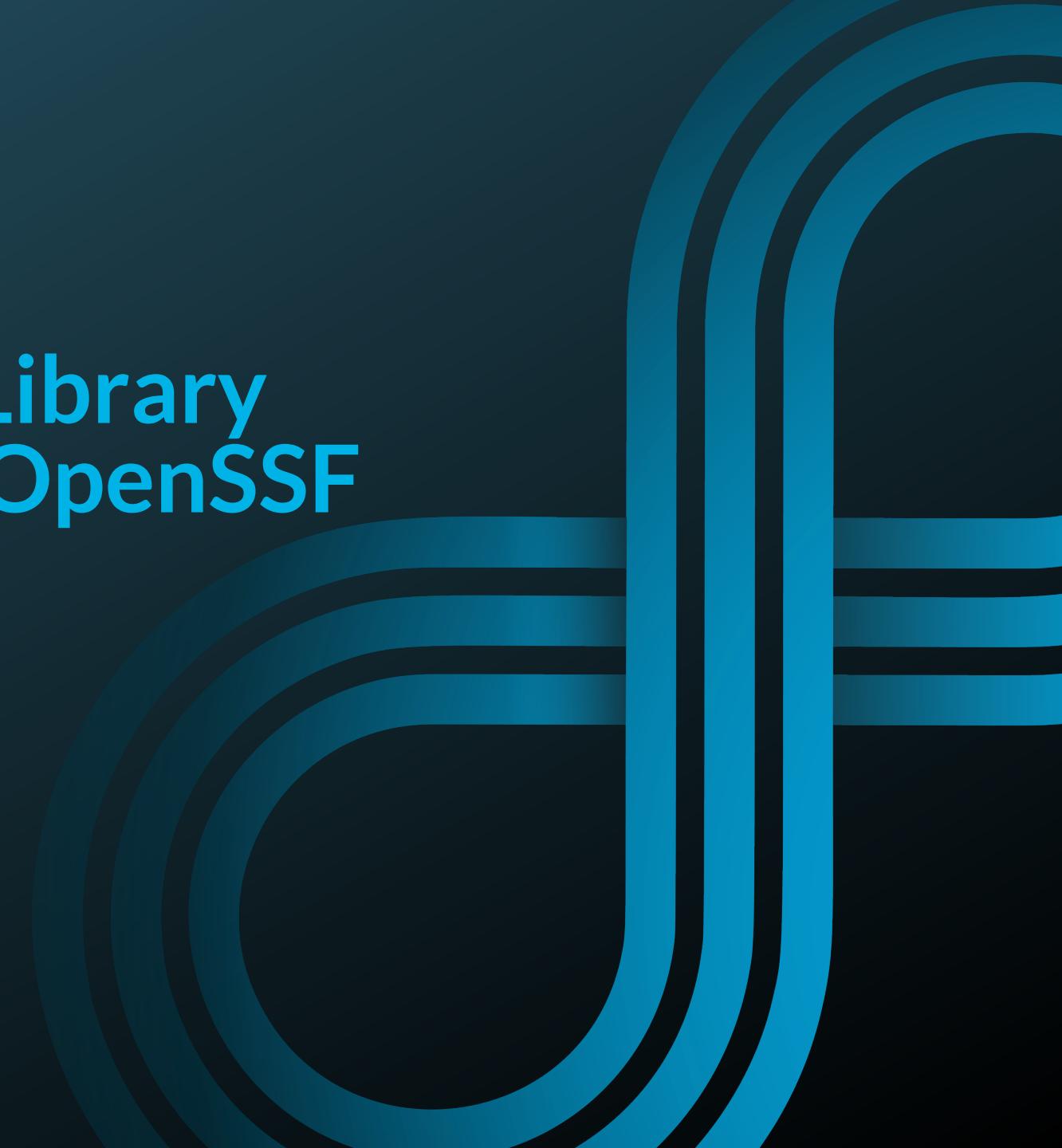


Reviewing 3rd Party Library Security Easily Using OpenSSF Scorecard

Niels Tanis
Sr. Principal Security Researcher



WeAreDevelopers
World Congress



Who am I?

- Niels Tanis
- Sr. Principal Security Researcher
 - Background .NET Development, Pentesting/ethical hacking, and software security consultancy
 - Research on static analysis for .NET apps
 - Enjoying Rust!
- Microsoft MVP – Developer Technologies

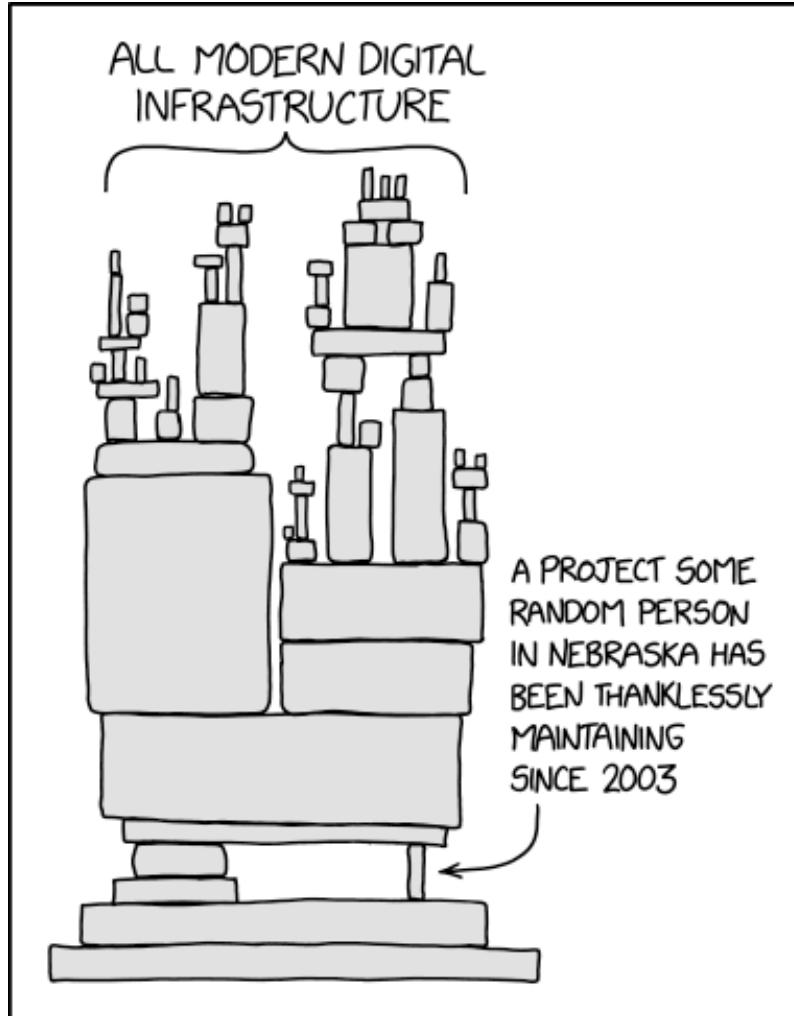
VERACODE



 @nielstanis@infosec.exchange

Modern Application Architecture

XKCD 2347



W>

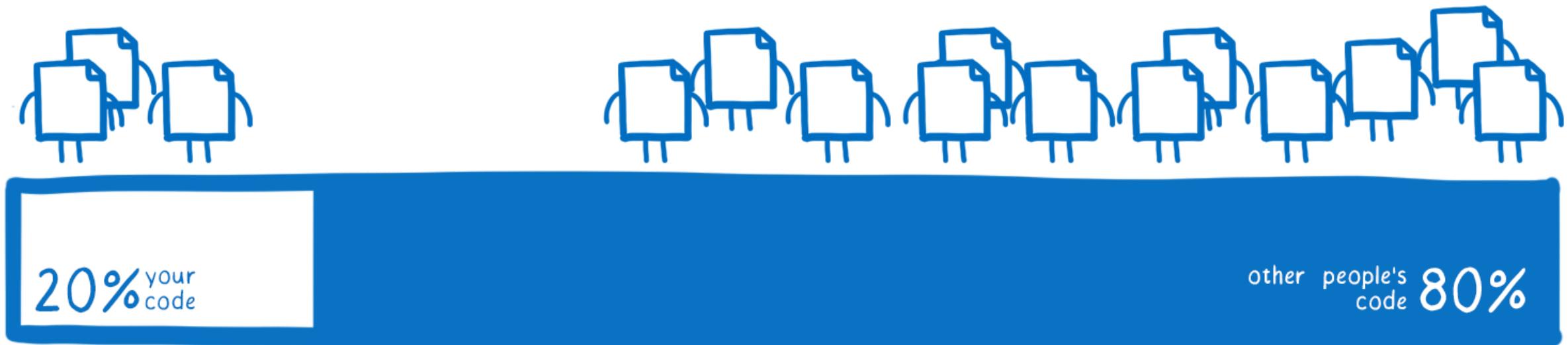
@nielstanis@infosec.exchange

Agenda

- Risks in 3rd Party Packages
- OpenSFF Scorecard
- Measure, New & Improved
- Conclusion - Q&A

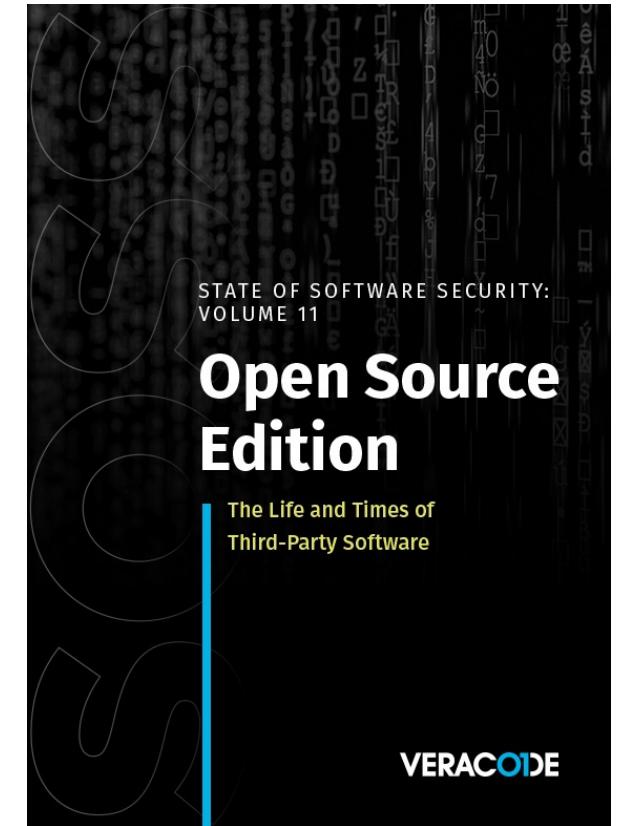


Average codebase composition



State of Software Security v11

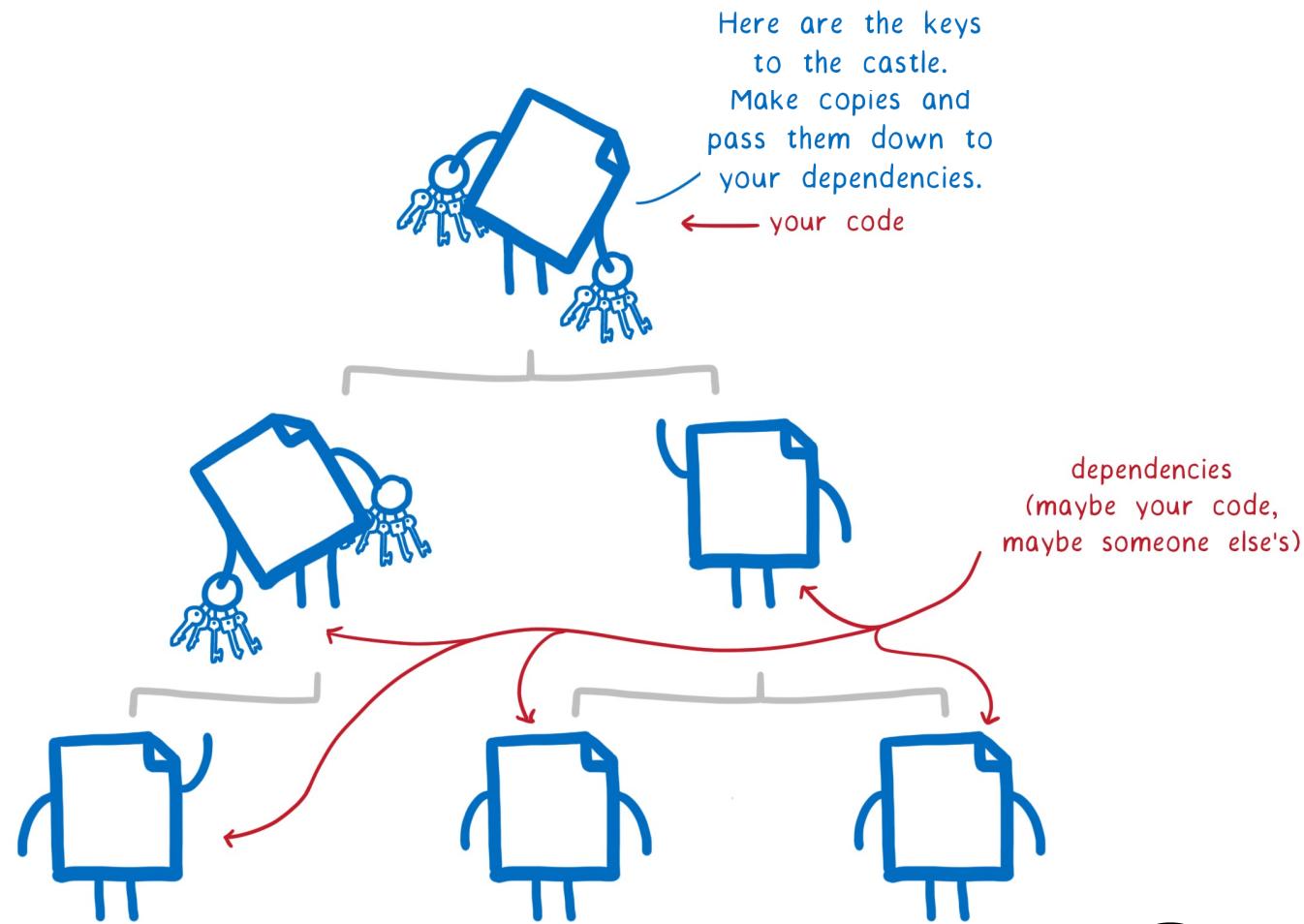
*”Despite this dynamic landscape,
79 percent of the time, developers
never update third-party libraries after
including them in a codebase.”*



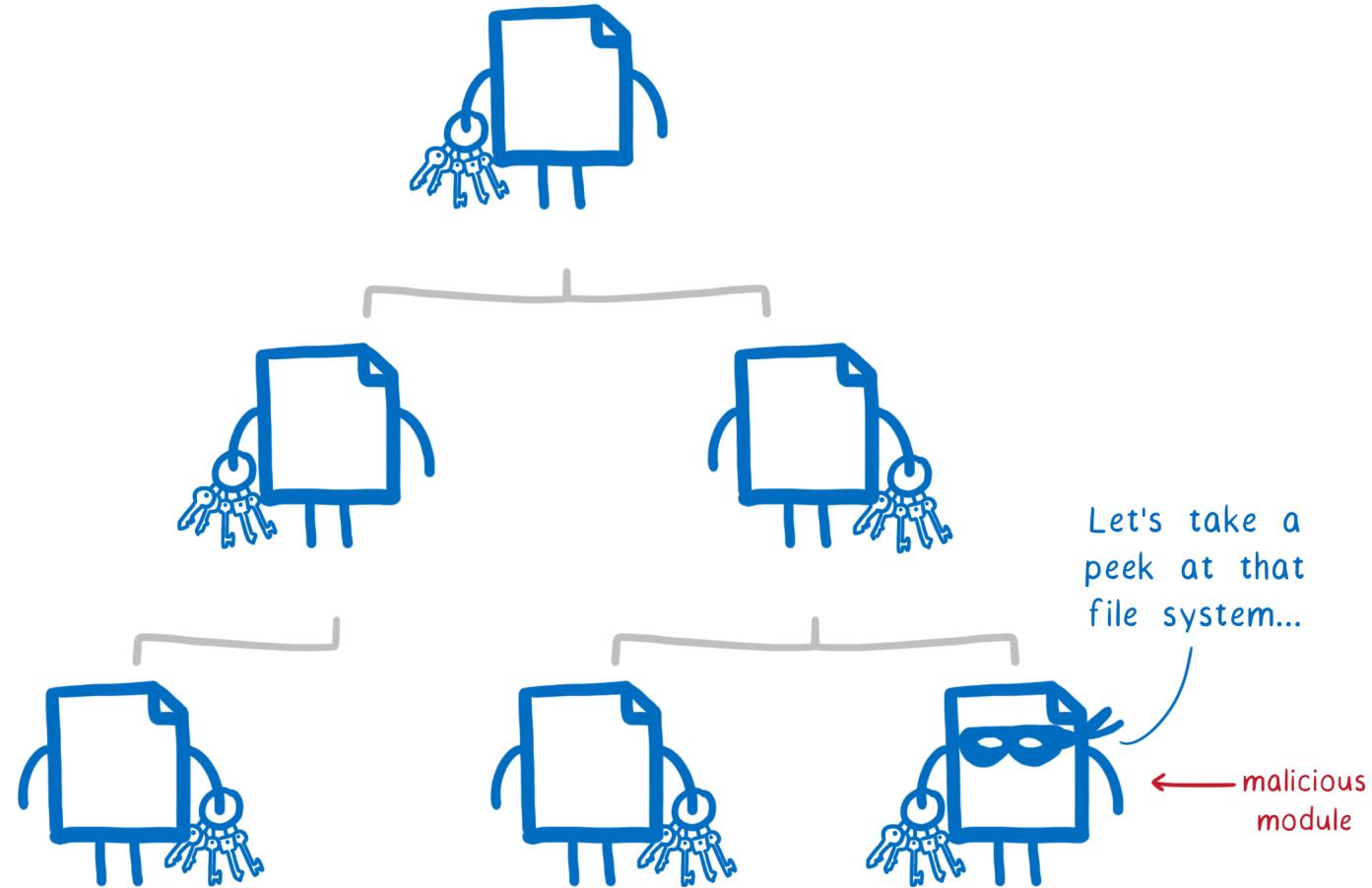
State of Log4j - 2 years later

- Analysed our data August-November 2023
 - Total set of almost 39K unique applications scanned
- 2.8% run version vulnerable to Log4Shell
- 3.8% run version patched but vulnerable to other CVE
- 32% rely on a version that's end-of-life and have no support for any patches.

Average codebase composition



Malicious Library



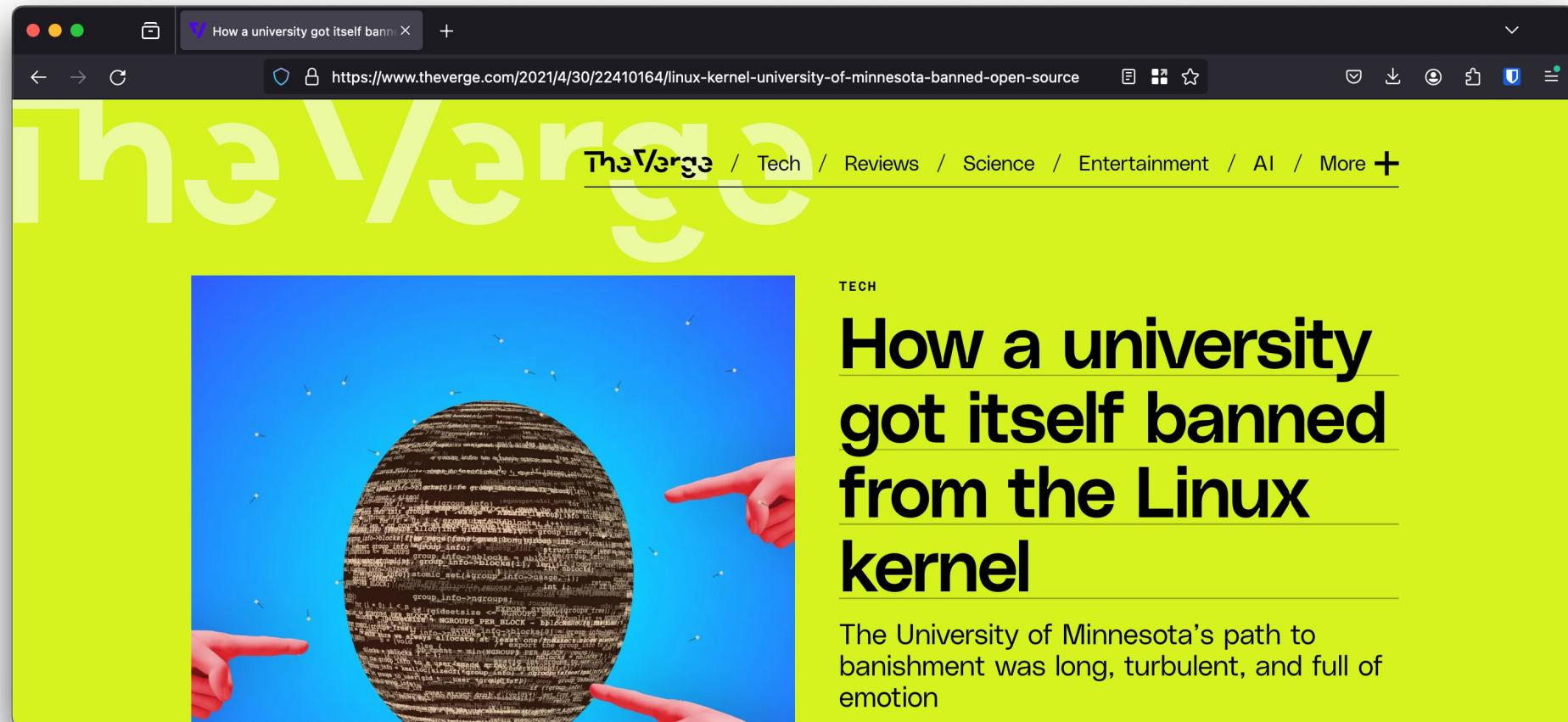
Malicious Package

The screenshot shows a web browser window displaying a news article. The title of the article is "48 Malicious npm Packages Found Deploying Reverse Shells on Developer Systems". The article was published on Nov 03, 2023, by the Newsroom. It is categorized under Software Security / Malware. The main image is a graphic of a shield made of binary code, containing a stylized figure. Below the image, there is a summary text: "A new set of 48 malicious npm packages have been discovered in the npm repository with capabilities to deploy a reverse shell on compromised systems."



 @nielstanis@infosec.exchange

Hypocrite Commits



W>

@nielstanis@infosec.exchange

XZ Backdoor

A screenshot of a web browser window showing an Ars Technica article. The title of the article is "Backdoor found in widely used Linux utility targets encrypted SSH connections". The article discusses a supply chain attack where malicious code was planted in xz Utils. The browser interface includes a dark mode header with the Ars Technica logo, a navigation bar with back, forward, and search icons, and a URL bar showing the article's link.

SUPPLY CHAIN ATTACK —

Backdoor found in widely used Linux utility targets encrypted SSH connections

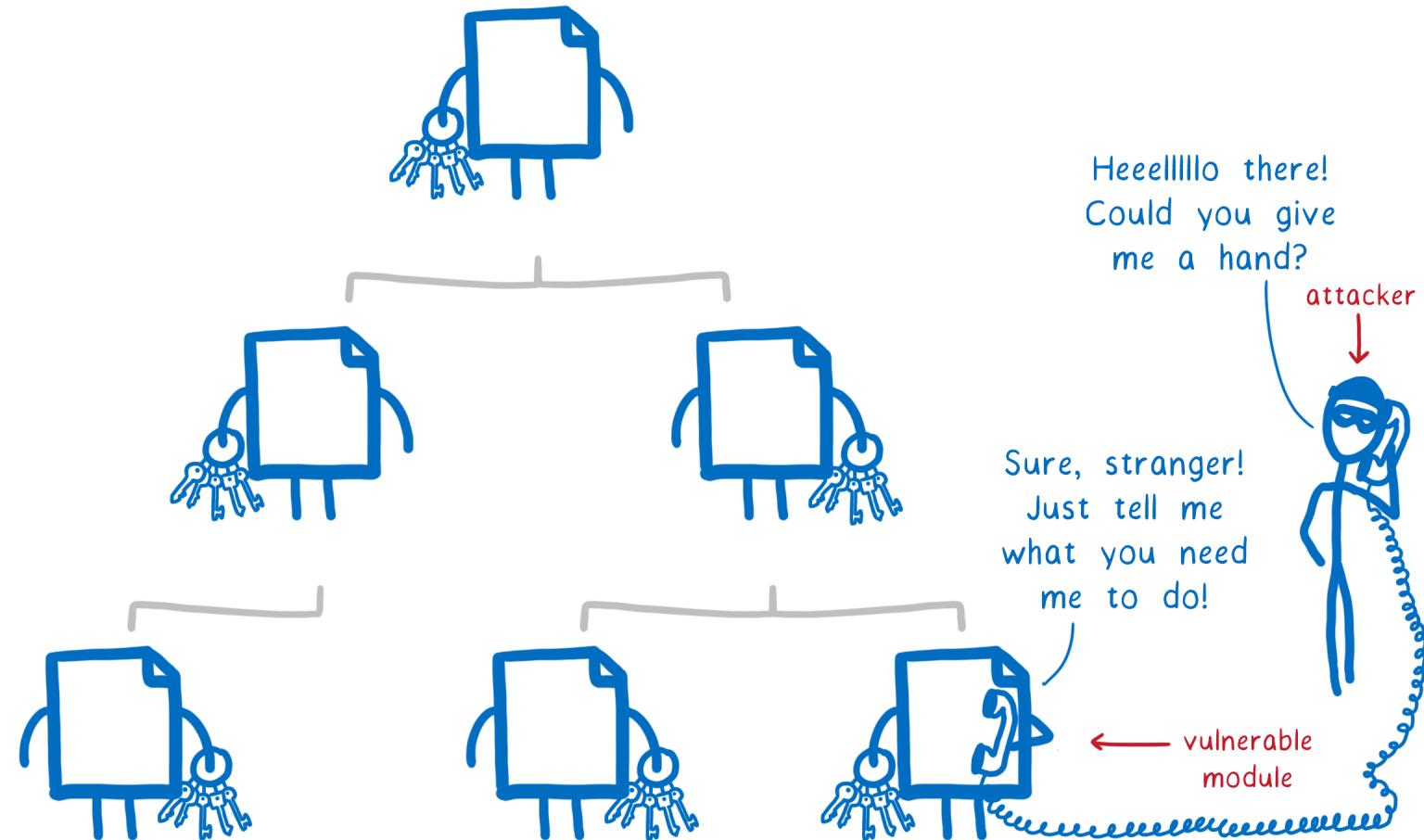
Malicious code planted in xz Utils has been circulating for more than a month.

DAN GOODIN - 3/29/2024, 7:50 PM

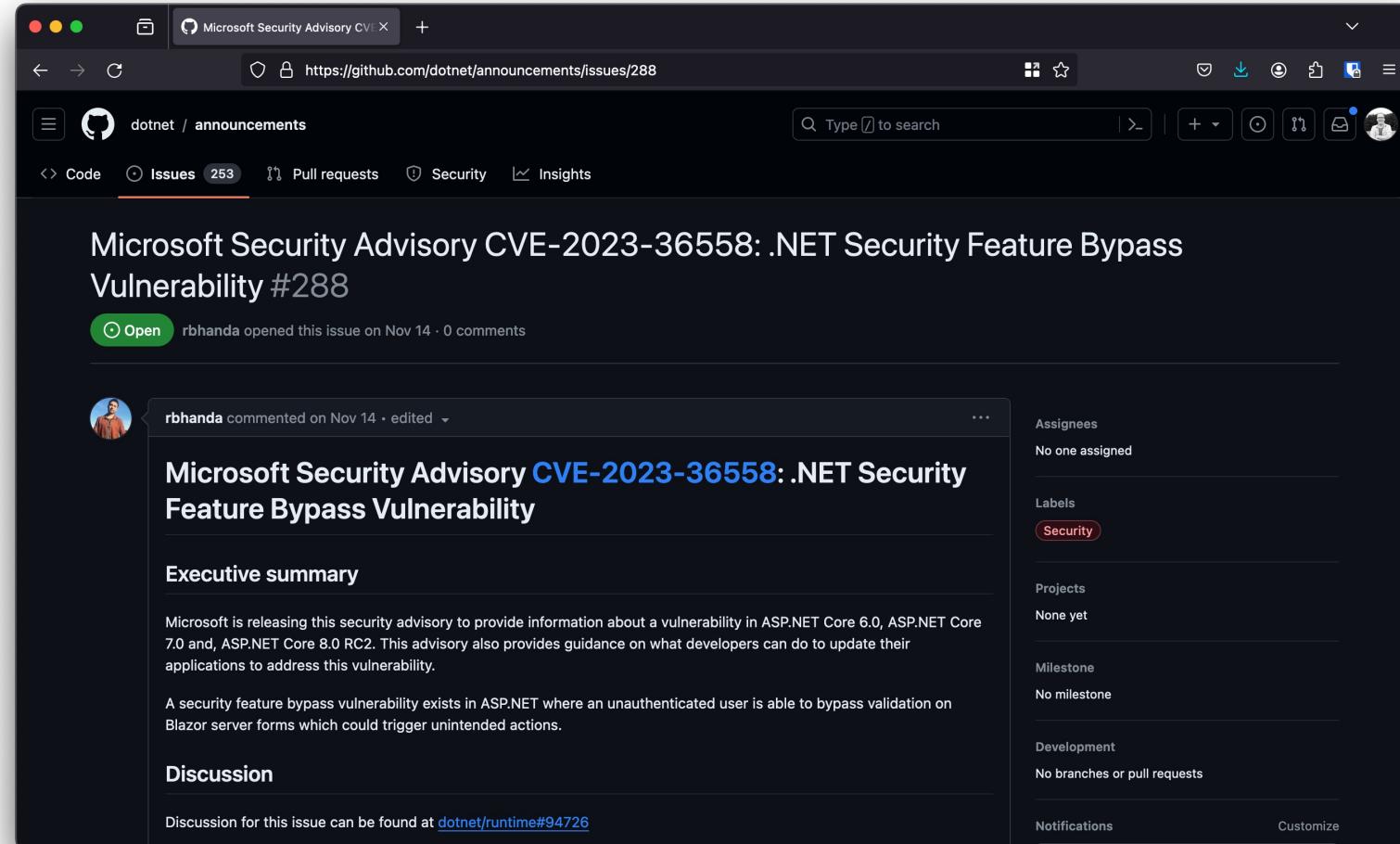


@nielstanis@infosec.exchange

Vulnerable Package



Vulnerabilities in Libraries



@nielstanis@infosec.exchange

DotNet CLI

```
nelson@ghost-m2:~/research/consoleapp$ dotnet list package --vulnerable --include-transitive

The following sources were used:
https://f.feedz.io/fennec/docgenerator/nuget/index.json
https://api.nuget.org/v3/index.json

Project `consoleapp` has the following vulnerable packages
[net8.0]:
Transitive Package      Resolved    Severity   Advisory URL
> Newtonsoft.Json        9.0.1       High       https://github.com/advisories/GHSA-5crp-9r3c-p9vr

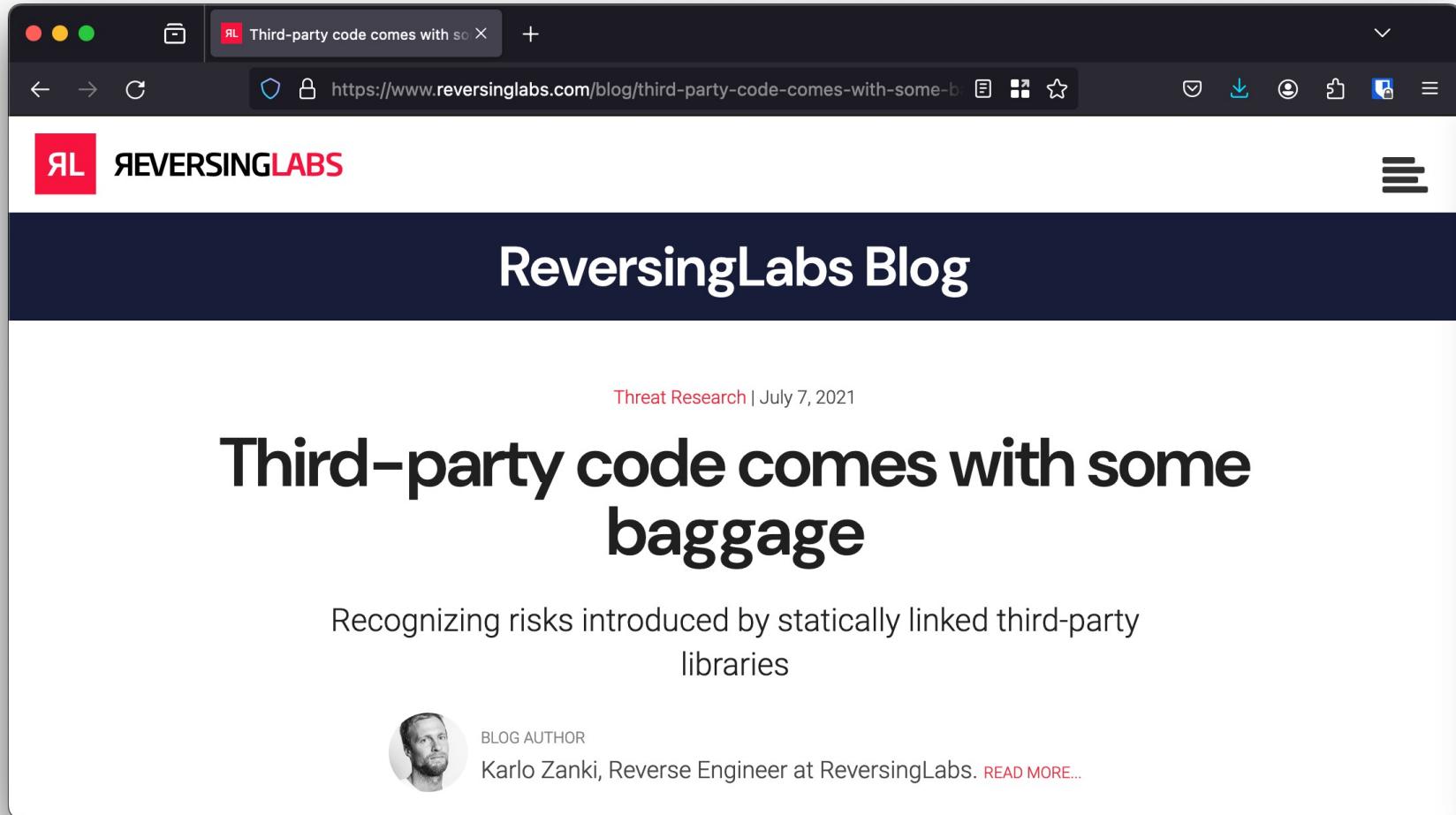
nelson@ghost-m2:~/research/consoleapp$
```



NPM Audit

==== npm audit security report ===	
# Run <code>npm install chokidar@2.0.3</code> to resolve 1 vulnerability	
SEMVER WARNING: Recommended action is a potentially breaking change	
Low	Prototype Pollution
Package	deep-extend
Dependency of	chokidar
Path	chokidar > fsevents > node-pre-gyp > rc > deep-extend
More info	https://nodesecurity.io/advisories/612

Do you know what's inside?



A screenshot of a web browser window showing a blog post from ReversingLabs. The browser has a dark mode interface. The tab bar shows a single tab titled "Third-party code comes with some baggage". The address bar displays the URL <https://www.reversinglabs.com/blog/third-party-code-comes-with-some-baggage>. The main content area features the ReversingLabs logo (a red square with "RL") and the text "REVERSINGLABS". Below this is a dark header bar with the text "ReversingLabs Blog". The main article title is "Third-party code comes with some baggage", with a subtitle "Recognizing risks introduced by statically linked third-party libraries". At the bottom, there is a "BLOG AUTHOR" section featuring a circular profile picture of a man, the name "Karlo Zanki", the title "Reverse Engineer at ReversingLabs", and a "READ MORE..." link.

Third-party code comes with some baggage

Threat Research | July 7, 2021

Third-party code comes with some baggage

Recognizing risks introduced by statically linked third-party libraries

BLOG AUTHOR
Karlo Zanki, Reverse Engineer at ReversingLabs. [READ MORE...](#)



@nielstanis@infosec.exchange

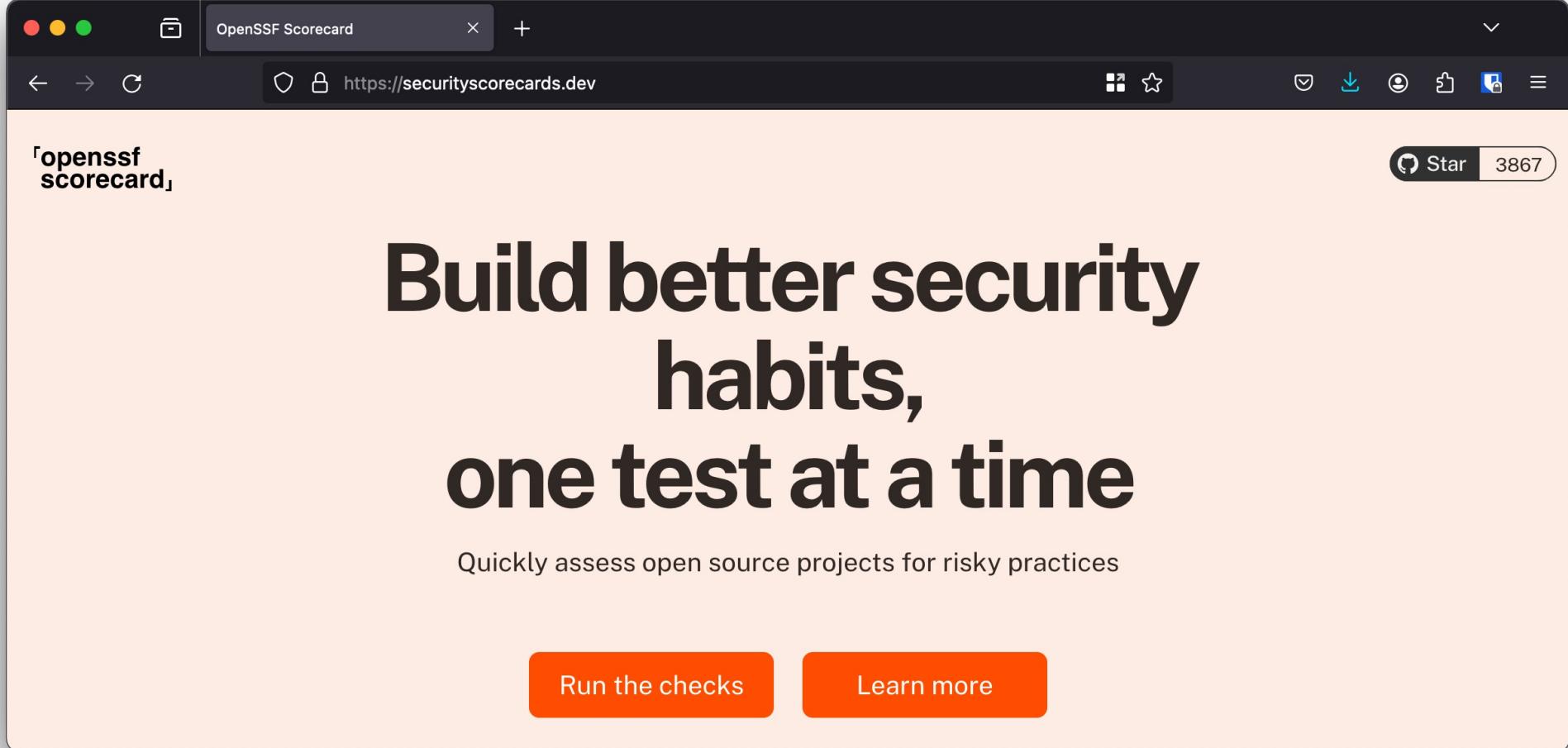
Nutrition Label for Software?



W>

@nielstanis@infosec.exchange

OpenSSF Scorecards

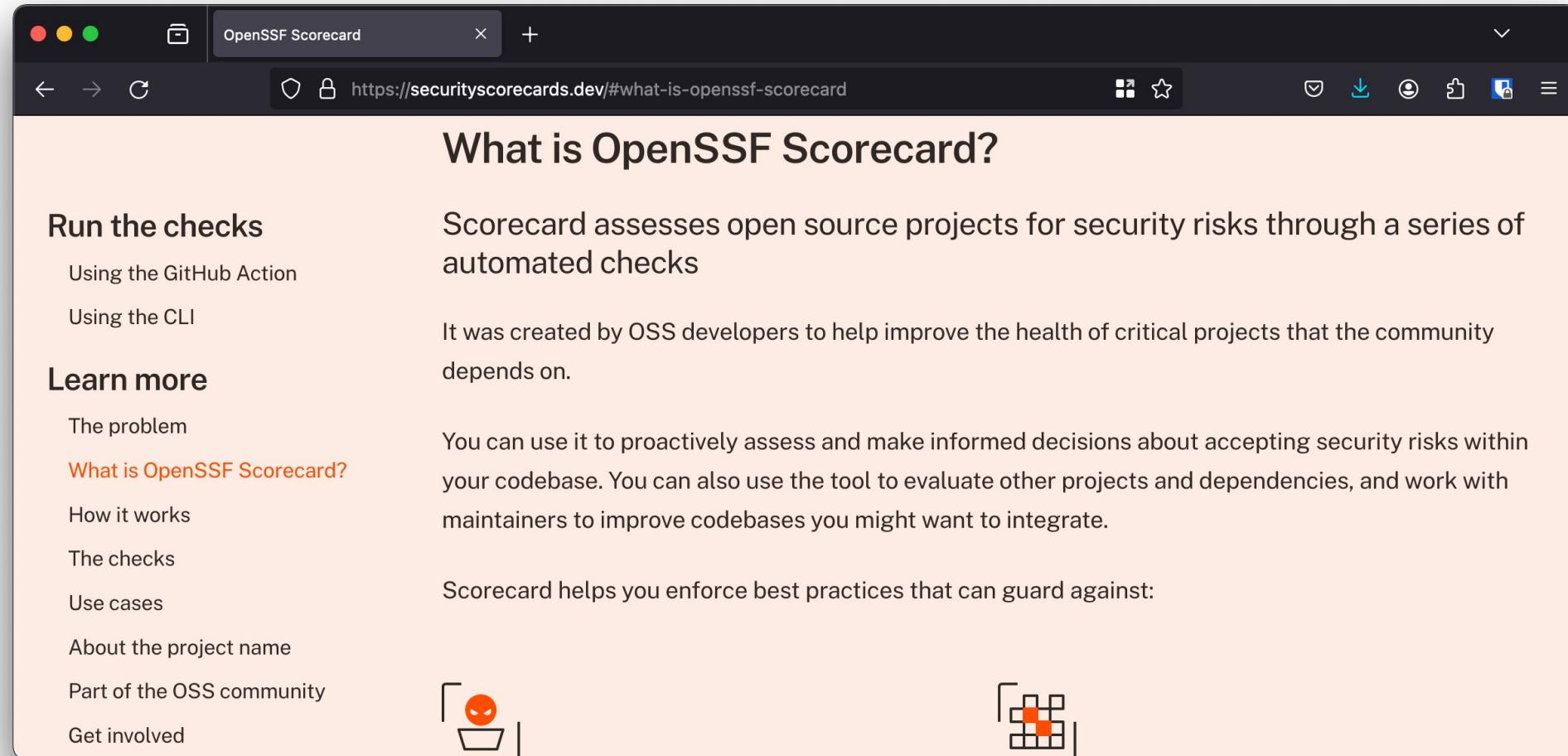


A screenshot of a web browser displaying the OpenSSF Scorecards homepage. The browser window has a dark header bar with the title "OpenSSF Scorecard". The address bar shows the URL "https://securityscorecards.dev". The main content area features a large, bold, black text block that reads: "Build better security habits, one test at a time". Below this text, a smaller subtitle says "Quickly assess open source projects for risky practices". At the bottom of the page are two orange buttons: "Run the checks" and "Learn more". In the top right corner of the browser window, there is a GitHub star icon with the number "3867". The background of the slide features a large, stylized blue "W>" logo on the right side.



 @nielstanis@infosec.exchange

OpenSSF Security Scorecards



The screenshot shows a web browser window titled "OpenSSF Scorecard". The URL in the address bar is <https://securityscorecards.dev/#what-is-openssf-scorecard>. The main content area has a light orange background and features the title "What is OpenSSF Scorecard?" in large bold text. To the left, there's a sidebar with sections like "Run the checks" and "Learn more". The "Run the checks" section includes links for "Using the GitHub Action" and "Using the CLI". The "Learn more" section includes links for "The problem", "What is OpenSSF Scorecard?", "How it works", "The checks", "Use cases", "About the project name", "Part of the OSS community", and "Get involved". The right side of the page contains descriptive text and icons. One icon is a small orange face with a red eye inside a white square frame, and another is a 4x4 grid of squares with one red square in the middle.

What is OpenSSF Scorecard?

Run the checks

Using the GitHub Action
Using the CLI

Learn more

The problem
[What is OpenSSF Scorecard?](#)
How it works
The checks
Use cases
About the project name
Part of the OSS community
Get involved

Scorecard assesses open source projects for security risks through a series of automated checks

It was created by OSS developers to help improve the health of critical projects that the community depends on.

You can use it to proactively assess and make informed decisions about accepting security risks within your codebase. You can also use the tool to evaluate other projects and dependencies, and work with maintainers to improve codebases you might want to integrate.

Scorecard helps you enforce best practices that can guard against:



OpenSSF Security Scorecards

The screenshot shows a web browser window titled "OpenSSF Scorecard" displaying the URL <https://securityscorecards.dev/#how-it-works>. The page content is as follows:

How it works

Run the checks

- Using the GitHub Action
- Using the CLI

Learn more

- The problem
- What is OpenSSF Scorecard?
- How it works**
- The checks
- Use cases
- About the project name
- Part of the OSS community
- Get involved

Scorecard checks for vulnerabilities affecting different parts of the software supply chain including **source code, build, dependencies, testing, and project maintenance**.

Each automated check returns a **score out of 10** and a **risk level**. The risk level **adds a weighting** to the score, and this weighting is compiled into a single, **aggregate score**. This score helps give a sense of the overall security posture of a project.

Alongside the scores, the tool provides remediation prompts to help you **fix problems** and strengthen your development practices.

Risk Level	Score
CRITICAL RISK	10
HIGH RISK	7.5
MEDIUM RISK	5

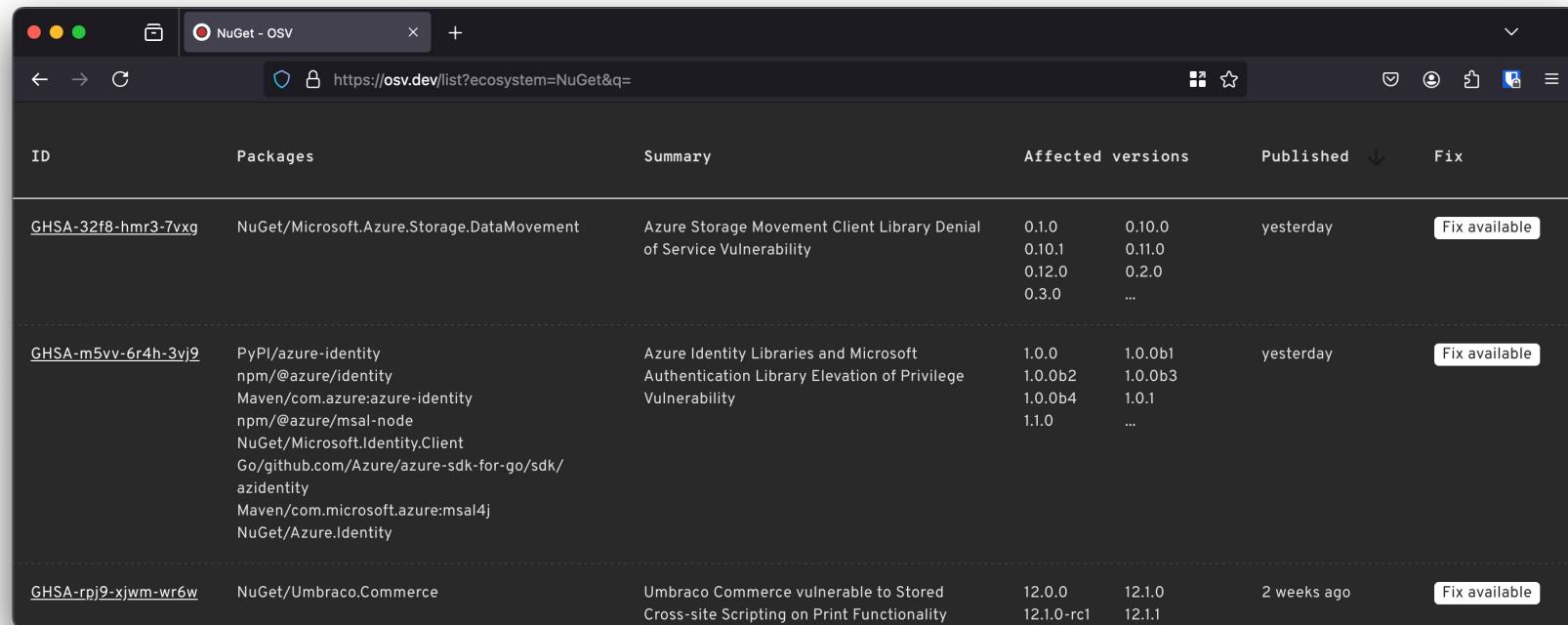


OpenSSF Security Scorecards

The screenshot shows a web browser window titled "OpenSSF Scorecard" at the URL <https://securityscorecards.dev/#the-checks>. The page content includes a sidebar with links for "Run the checks" (GitHub Action, CLI) and "Learn more" (The problem, What is OpenSSF Scorecard?, How it works, The checks, Use cases, About the project name, Part of the OSS community, Get involved). To the right of the sidebar is a large graphic featuring six overlapping circles arranged in a hexagonal pattern, all contained within a larger orange circle. The circles are labeled: BUILD RISK ASSESSMENT, SOURCE RISK ASSESSMENT, CONTINUOUS TESTING, MAINTENANCE, CODE VULNERABILITIES, and HOLISTIC SECURITY PRACTICES (which is highlighted in red).

Code Vulnerabilities (High)

- Does the project have unfixed vulnerabilities?
Uses the OSV service.



The screenshot shows a dark-themed web browser window titled "NuGet - OSV". The URL in the address bar is <https://osv.dev/list?ecosystem=NuGet&q=>. The page displays a table of vulnerabilities:

ID	Packages	Summary	Affected versions	Published	Fix	
GHSA-32f8-hmr3-7vxg	NuGet/Microsoft.Azure.Storage.DataMovement	Azure Storage Movement Client Library Denial of Service Vulnerability	0.1.0 0.10.1 0.12.0 0.3.0	0.10.0 0.11.0 0.2.0 ...	yesterday	Fix available
GHSA-m5vv-6r4h-3vj9	PyPI/azure-identity npm/@azure/identity Maven/com.azure:azure-identity npm/@azure/msal-node NuGet/Microsoft.Identity.Client Go/github.com/Azure/azure-sdk-for-go/sdk/azidentity Maven/com.microsoft.azure:msal4j NuGet/Azure.Identity	Azure Identity Libraries and Microsoft Authentication Library Elevation of Privilege Vulnerability	1.0.0 1.0.0b2 1.0.0b4 1.1.0	1.0.0b1 1.0.0b3 1.0.1 ...	yesterday	Fix available
GHSA-rpj9-xjwm-wr6w	NuGet/Umbraco.Commerce	Umbraco Commerce vulnerable to Stored Cross-site Scripting on Print Functionality	12.0.0 12.1.0-rc1	12.1.0	2 weeks ago	Fix available



Maintenance Dependency-Update-Tool (**High**)

- Does the project use a dependency update tool?
For example Dependabot or Renovate bot?
- Out-of-date dependencies make a project vulnerable
to known flaws and prone to attacks.

Maintenance Security Policy (**Medium**)

- Does project have published security policy?
- E.g. a file named **SECURITY.md** (case-insensitive) in a few well-known directories.
- A security policy can give users information about what constitutes a vulnerability and how to report one securely so that information about a bug is not publicly visible.

Maintenance License (**Low**)

- Does project have license published?
- A license can give users information about how the source code may or may not be used.
- The lack of a license will impede any kind of security review or audit and creates a legal risk for potential users.

Maintenance CII Best Practices (**Low**)

- OpenSSF Best Practices Badge Program
- Way for Open Source Software projects to show that they follow best practices.
- Projects can voluntarily self-certify, at no cost, by using this web application to explain how they follow each best practice.



openssf best practices **passing**

Continuous testing

CI Tests (**Low**)

- Does the project run tests before pull requests are merged?
- The check works by looking for a set of CI-system names in GitHub CheckRuns and Statuses among the recent commits (~30).

Continuous testing

Fuzzing (Medium)

- This check tries to determine if the project uses fuzzing by checking:
 - Added to [OSS-Fuzz](#) project.
 - If [ClusterFuzzLite](#) is deployed in the repository;
- Does it make sense to do fuzzing on managed languages like Java and/or .NET?

Continuous testing

Static Code Analysis (**Medium**)

- This check tries to determine if the project uses Static Application Security Testing (SAST), also known as static code analysis. It is currently limited to repositories hosted on GitHub.
 - CodeQL
 - SonarCloud
- Definitely room for improvement!

Source Risk Assessment

Binary Artifacts (**High**)

- This check determines whether the project has generated executable (binary) artifacts in the source repository.
- Binary artifacts cannot be reviewed, allowing possible obsolete or maliciously subverted executables.
- There is need for **reproducible** builds!



Source Risk Assessment Branch Protection (**High**)

- This check determines whether a project's default and release branches are protected with GitHub's branch protection or repository rules settings.
 - Requiring code review
 - Prevent force push, in case of public branch all is lost!

Source Risk Assessment

Dangerous Workflow (**Critical**)

- This check determines whether the project's GitHub Action workflows has dangerous code patterns.
 - Untrusted Code Checkout with certain triggers
 - Script Injection with Untrusted Context Variables
- <https://securitylab.github.com/research/github-actions-preventing-pwn-requests/>

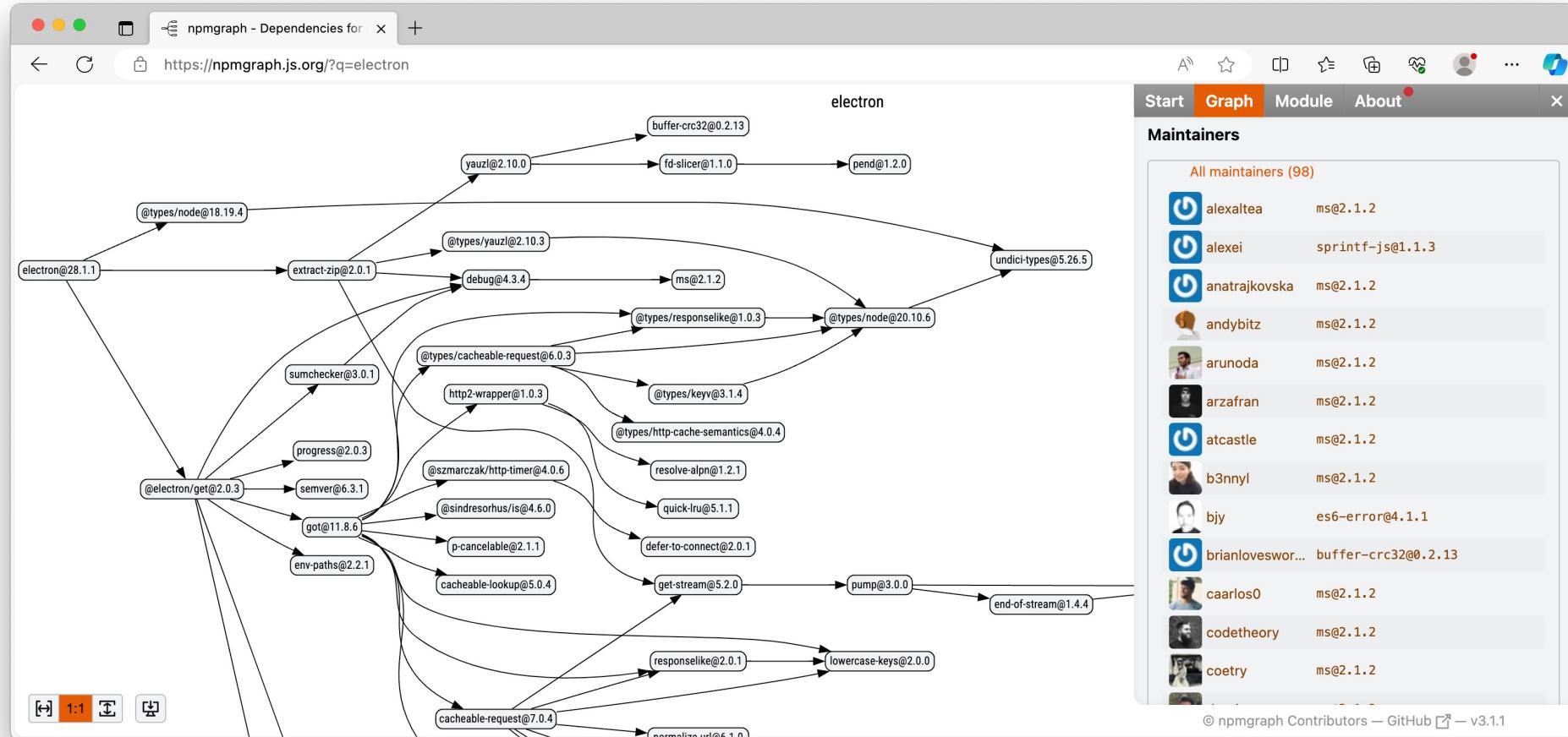
Source Risk Assessment Code Review (**Low**)

- This check determines whether the project requires human code review before pull requests are merged.
- The check determines whether the most recent changes (over the last ~30 commits) have an approval on GitHub and merger!=committer (implicit review)

Source Risk Assessment Contributors (Low)

- This check tries to determine if the project has recent contributors from multiple organizations (e.g., companies).
- Relying on single contributor is a risk for sure!
- But is a large list of contributors good?

Source Risk Assessment Contributors (Low)



@nielstanis@infosec.exchange

Build Risk Assessment

Pinned Dependencies (**High**)

- Does the project pin dependencies used during its build and release process.
- For .NET → `RestorePackagesWithLockFile` in MSBuild results in `packages.lock.json` file containing versioned dependency tree with hashes
- If Workflow is present what about the Actions used?

Build Risk Assessment Token Permission (High)

- This check determines whether the project's automated workflows tokens follow the principle of least privilege.
- This is important because attackers may use a compromised token with write access to, for example, push malicious code into the project.

Build Risk Assesement Packaging (Medium)

- This check tries to determine if the project is published as a package.
- Packages give users of a project an easy way to download, install, update, and uninstall the software by a package manager.

Build Risk Assessment Signed Releases (**High**)

- This check tries to determine if the project cryptographically signs release artifacts.
 - Signed release packages
 - Signed build provenance

Demo OpenSSF Scorecard

Running checks



@nielstanis@infosec.exchange

Measure?



OpenSSF Annual Report 2023

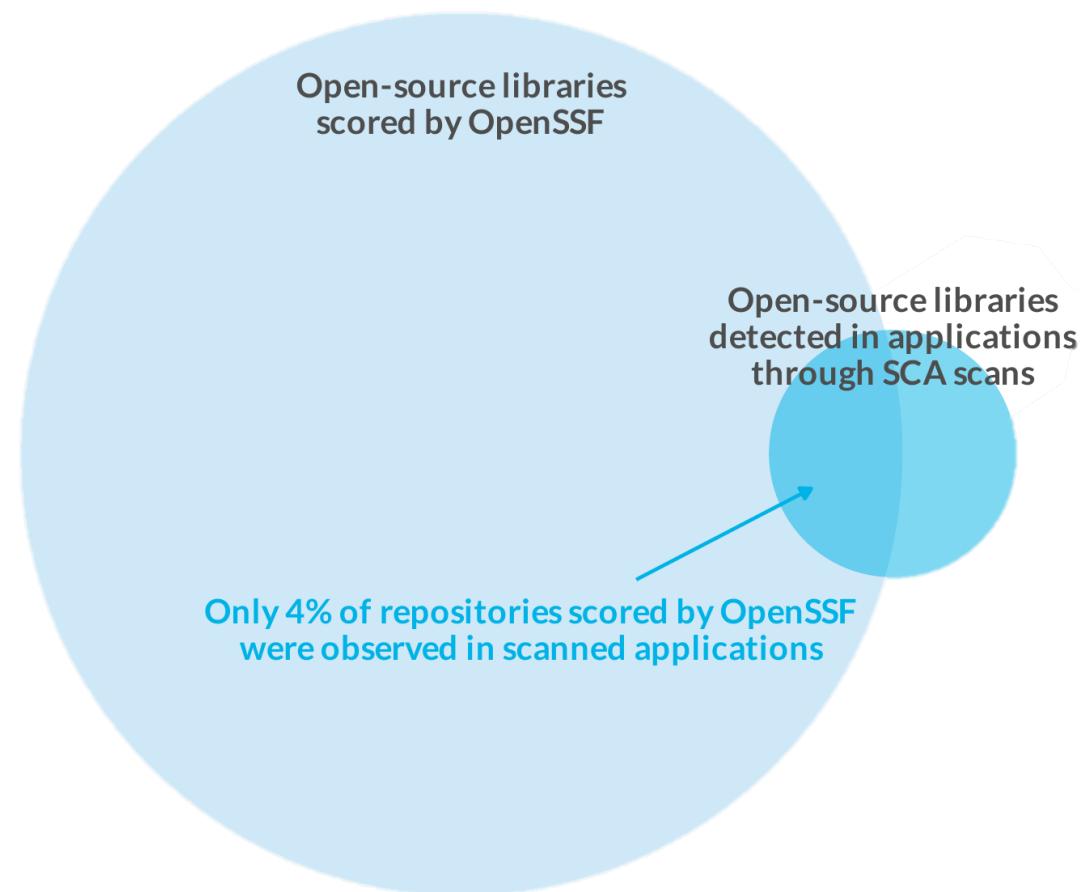
OpenSSF Scorecard project
has **3,776 stars** on GitHub,
and runs a **weekly automated
assessment scan** against
software security criteria
of over **1M OSS projects**



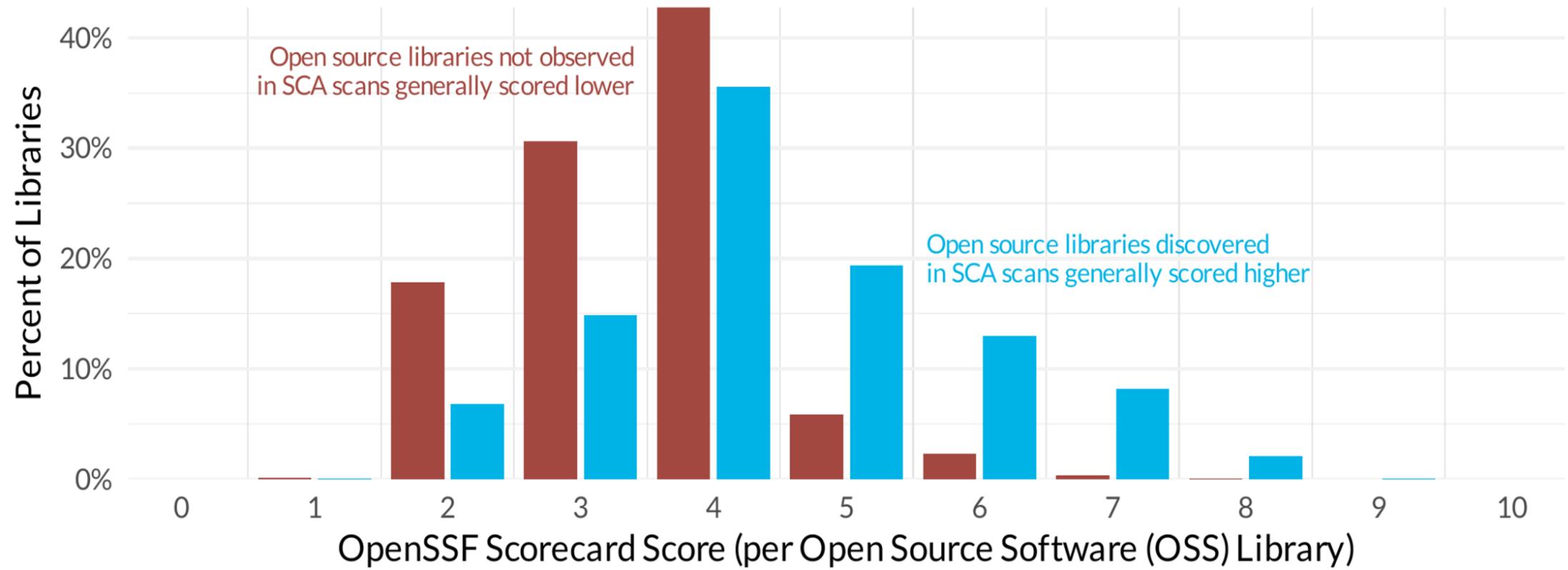
SOSS & OpenSSF Scorecard



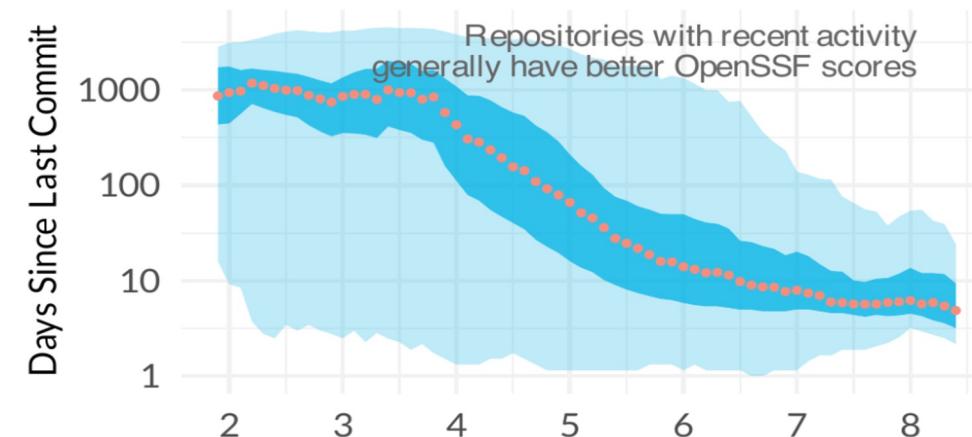
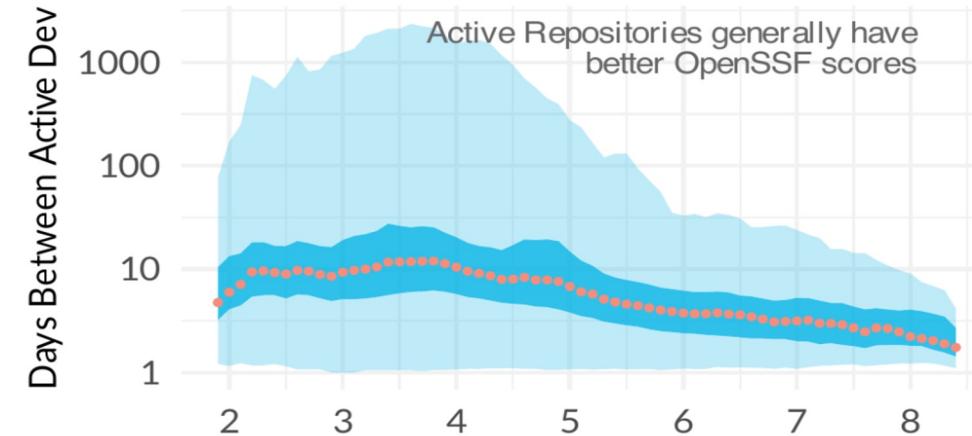
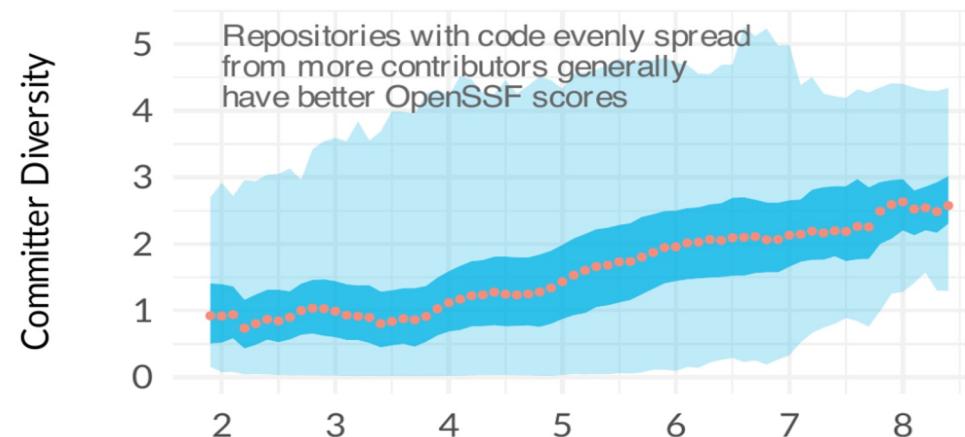
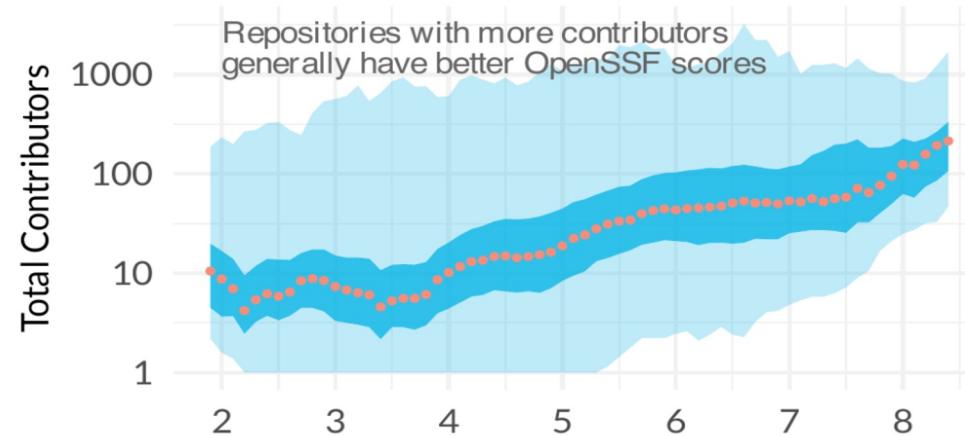
SOSS & OpenSSF Scorecard



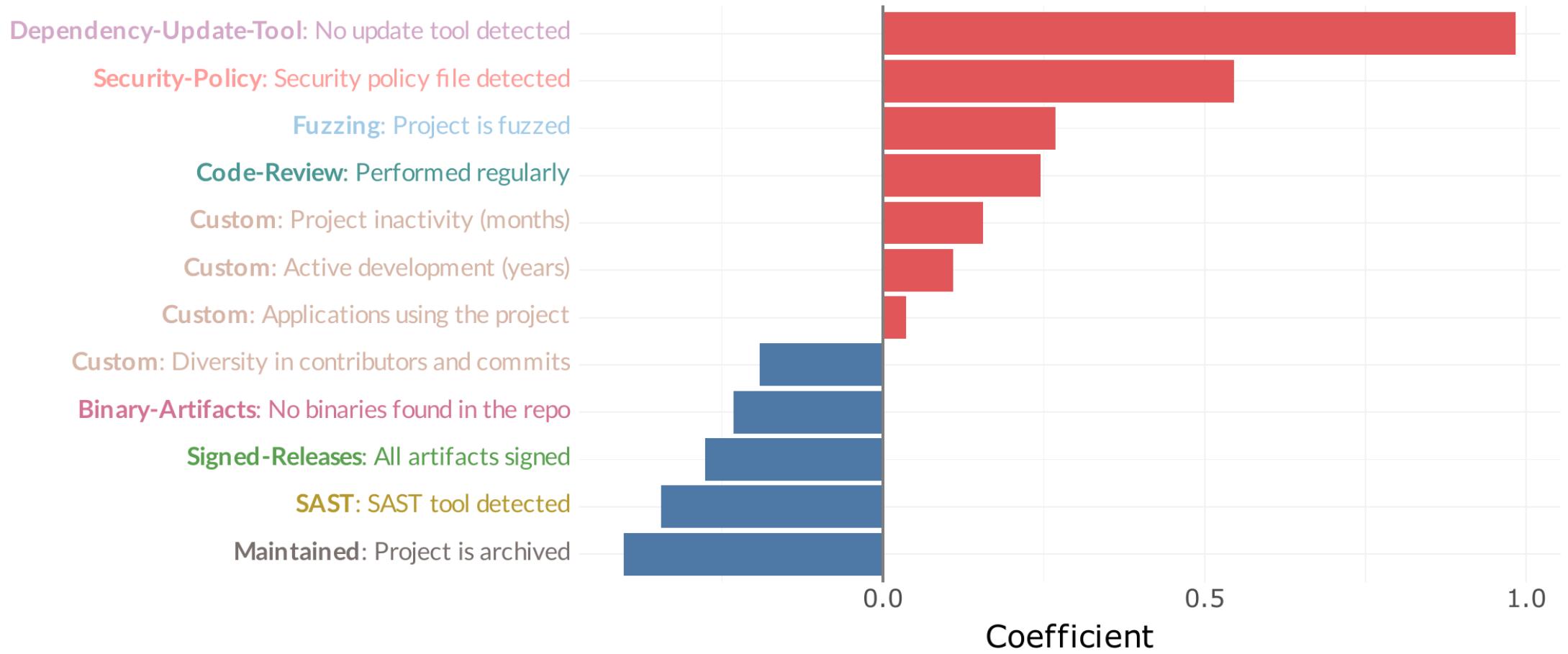
Correlation between SOSS



Github commits vs OpenSSF



What really contributes to OSS Security?



What can we improve?



Fuzzing



- Fuzzing, or fuzz testing
 - Automated software testing method that uses a wide range of *invalid* and unexpected data as input to find flaws
- Definitely good for finding C/C++ memory issues
- Can it be of any value with managed languages like .NET and/or Java?

Fuzzing .NET – Jil JSON Serializer



```
public static void Main(string[] args)
{
    SharpFuzz.Fuzzer.OutOfProcess.Run(stream => {
        try
        {
            using (var reader = new System.IO.StreamReader(stream))
                JSON.DeserializeDynamic(reader);
        }
        catch (DeserializationException) { }
    });
}
```



Fuzzomatic: Using AI to Fuzz Rust

New & Improved!

The screenshot shows a web browser window displaying a blog post titled "Introducing Fuzzomatic: Using AI to Automatically Fuzz Rust Projects from Scratch". The post is dated December 7, 2023. The content includes sections on how it works, the AI models used, and a code example of a Rust fuzz target. At the bottom right of the post, there are social sharing icons for Comment, Reblog, Subscribe, and more.

How does it work?

Fuzzomatic relies on libFuzzer and cargo-fuzz as a backend. It also uses a variety of approaches that combine AI and deterministic techniques to achieve its goal.

We used the OpenAI API to generate and fix fuzz targets in our approaches. We mostly used the gpt-3.5-turbo and gpt-3.5-turbo-16k models. The latter is used as a fallback when our prompts are longer than what the former supports.

Fuzz targets and coverage-guided fuzzing

The output of the first step is a source code file: a fuzz target. A libFuzzer fuzz target in Rust looks like this:

```
1  #![no_main]
2
3  extern crate libfuzzer_sys;
4
5  use mylib_under_test::MyModule;
6  use libfuzzer_sys::fuzz_target;
7
8  fuzz_target!(data: &[u8] {
9      // fuzzed code goes here
10     if let Ok(input) = std::str::from_utf8(data) {
11         MyModule::target_function(input);
12     }
13});
```

This fuzz target needs to be compiled into an executable. As you can see, this program depends on libFuzzer and also depends on the library under test, here "mylib_under_test". The "fuzz_target!" macro makes it easy for us to just write what needs to be called, provided that we receive a byte slice, the "data" variable in the above example. Here we convert these bytes to a UTF-8 string and call our target function and pass that string as an argument. LibFuzzer takes care of calling our fuzz target repeatedly with random bytes. It measures the code coverage to assess whether the random input helps cover more code. We say it's coverage-guided fuzzing.



@nielstanis@infosec.exchange

Static Code Analysis (SAST)



```
public byte[] CreateHash(string password)
{
    var b = Encoding.UTF8.GetBytes(password);
    return SHA1.HashData(b);
}
```



Static Code Analysis (SAST)



```
public class CustomerController : Controller
{
    public IActionResult GenerateCustomerReport(string customerID)
    {
        var data = Reporting.GenerateCustomerReportOverview(customerID)
        return View(data);
    }
    public static class Reporting
    {
        public static byte[] GenerateCustomerReportOverview(string ID)
        {
            return System.IO.File.ReadAllBytes("./data/{ID}.pdf");
        }
    }
}
```



Package Reproducibility



- A build is **reproducible** if given the same source code, build environment and build instructions, any party can recreate bit-by-bit identical copies of all specified artifacts.

Reproducible-Builds.org



The screenshot shows a web browser displaying the 'Definitions' page from the Reproducible-Builds.org documentation. The page title is 'Definitions' and the URL is <https://reproducible-builds.org/docs/definition/>. The left sidebar contains links for Home, News, Documentation (with 'Definitions' selected), Tools, Who is involved?, Talks, Events, Continuous tests, and Contribute. Sponsors listed include the Open Technology Fund and Sovereign TechFund. The main content area has two sections: 'When is a build reproducible?' and 'Explanations'. The 'When is a build reproducible?' section defines reproducibility as creating bit-by-bit identical copies of artifacts given the same source code, environment, and instructions. It also describes the relevant attributes of the build environment and artifacts. The 'Explanations' section provides details on source code, build environment, artifacts, and verification. A sidebar on the right lists related topics like Introduction, Definitions, History, Buy-in, Making plans, Academic publications, Achieve deterministic builds, commandments of reproducible builds, Variations in the build environment, SOURCE_DATE_EPOCH, Deterministic build systems, Volatile inputs can disappear, Stable order for inputs, Value initialization, Version information, and Timestamps.

Definitions

When is a build reproducible?

A build is **reproducible** if given the same source code, build environment and build instructions, any party can recreate bit-by-bit identical copies of all specified artifacts.

The relevant attributes of the build environment, the build instructions and the source code as well as the expected reproducible artifacts are defined by the authors or distributors. The artifacts of a build are the parts of the build results that are the desired primary output.

Explanations

Source code is usually a checkout from version control at a specific revision or a source code archive.

Relevant attributes of the build environment would usually include dependencies and their versions, build configuration flags and environment variables as far as they are used by the build system (eg. the locale). It is preferable to reduce this set of attributes.

Artifacts would include executables, distribution packages or filesystem images. They would not usually include build logs or similar ancillary outputs.

The reproducibility of artifacts is **verified** by bit-by-bit comparison. This is usually performed using

[Documentation home](#)

[Introduction](#)
[Definitions](#)
[History](#)
[Buy-in](#)
[Making plans](#)
[Academic publications](#)

[Achieve deterministic builds](#)
[commandments of reproducible builds](#)
[Variations in the build environment](#)
[SOURCE_DATE_EPOCH](#)
[Deterministic build systems](#)
[Volatile inputs can disappear](#)
[Stable order for inputs](#)
[Value initialization](#)
[Version information](#)
[Timestamps](#)



Application Inspector



The screenshot shows the Microsoft Application Inspector interface running in a Microsoft Edge browser window. The title bar reads "Microsoft Application Ir" and the address bar shows "file:///C:/Demo/Appinspector/publish/output.html#". The main content area has a header "Application Features" with a sub-instruction about viewing major characteristics by feature group. Below this is a "Feature Groups" section containing ten categories with corresponding icons: Select Features, General Features, Development, Active Content, Data Storage, Sensitive Data, Cloud Services, OS Integration, OS System Changes, and Other. To the right is an "Associated Rules" section with a heading "Name (click to view source)" and a list of four items: Authentication: Microsoft (Identity), Authentication: General, and Authentication: (Oauth).



Application Inspector



— Select Features	Feature	Confidence	Details
	 Authentication		View
	 Authorization		View
	 Cryptography		View
	 Object Deserialization		N/A
	 AV Media Parsing		N/A
	 Dynamic Command Execution		N/A



 @nielstanis@infosec.exchange

Community Review



The screenshot shows a web browser window displaying the [Cargo Vet](https://mozilla.github.io/cargo-vet/) documentation. The page has a dark theme with light-colored text. On the left, there's a sidebar with a table of contents:

- 1. Introduction
 - 1.1. Motivation
 - 1.2. How it Works
- 2. Tutorial
 - 2.1. Installation
 - 2.2. Setup
 - 2.3. Audit Criteria
 - 2.4. Importing Audits
 - 2.5. Recording Audits
 - 2.6. Performing Audits
 - 2.7. Trusting Publishers
 - 2.8. Specifying Policies
 - 2.9. Multiple Repositories
 - 2.10. Configuring CI
 - 2.11. Curating Your Audit Set
- 3. Reference
 - 3.1. Configuration
 - 3.2. Audit Entries
 - 3.3. Wildcard Audit Entries
 - 3.4. Trusted Entries

The main content area is titled "Cargo Vet". It starts with a brief introduction:

The `cargo vet` subcommand is a tool to help projects ensure that third-party Rust dependencies have been audited by a trusted entity. It strives to be lightweight and easy to integrate.

When run, `cargo vet` matches all of a project's third-party dependencies against a set of audits performed by the project authors or entities they trust. If there are any gaps, the tool provides mechanical assistance in performing and documenting the audit.

The primary reason that people do not ordinarily audit open-source dependencies is that it is too much work. There are a few key ways that `cargo vet` aims to reduce developer effort to a manageable level:

- **Sharing:** Public crates are often used by many projects. These projects can share their findings with each other to avoid duplicating work.
- **Relative Audits:** Different versions of the same crate are often quite similar to each other. Developers can inspect the difference between two versions, and record that if the first version was vetted, the second can be considered vetted as well.
- **Deferred Audits:** It is not always practical to achieve full coverage. Dependencies can be added to a list of exceptions which can be ratcheted down over time. This makes it trivial to introduce `cargo vet` to a new project and guard against future vulnerabilities while vetting the



Conclusion

- Scorecard helps security reviewing a used Package
- Better understand what's inside, how it's build/maintained and what are the risks!
- Scorecard should not be a goal on it's own!

Conclusion

- Room for improvements
 - Reproducibility Tools (diff, insights)
 - Reporting
 - Trust Graph
 - Contribute back to OpenSSF Scorecard



Questions?



W>

 @nielstanis@infosec.exchange

Links

- <https://github.com/nielstanis/wearedevs2024/>
- ntanis at Veracode.com
- @nielstanis@infosec.exchange
- <https://www.fennec.dev> & <https://blog.fennec.dev>

