



Reviewing NuGet Packages security easily using OpenSSF Scorecard

Niels Tanis
Sr. Principal Security Researcher

NDC { Security }

Who am I?

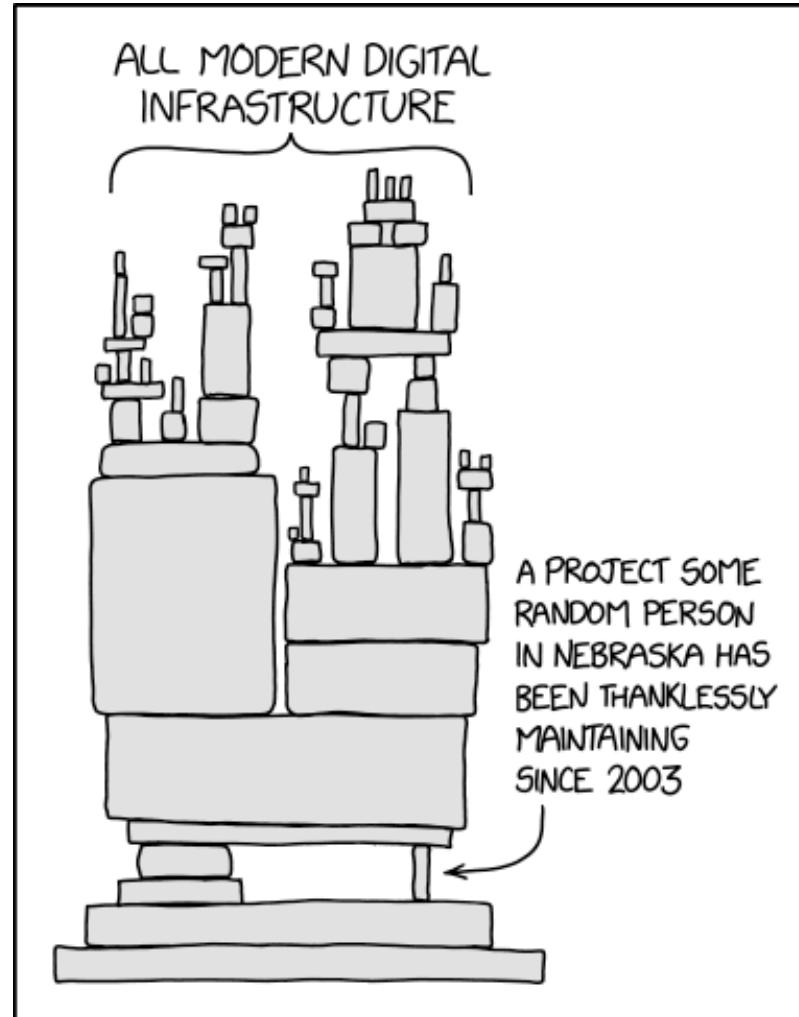
- Niels Tanis
- Sr. Principal Security Researcher
 - Background .NET Development, Pentesting/ethical hacking, and software security consultancy
 - Research on static analysis for .NET apps
 - Enjoying Rust!
- Microsoft MVP – Developer Technologies

VERACODE



Modern Application Architecture

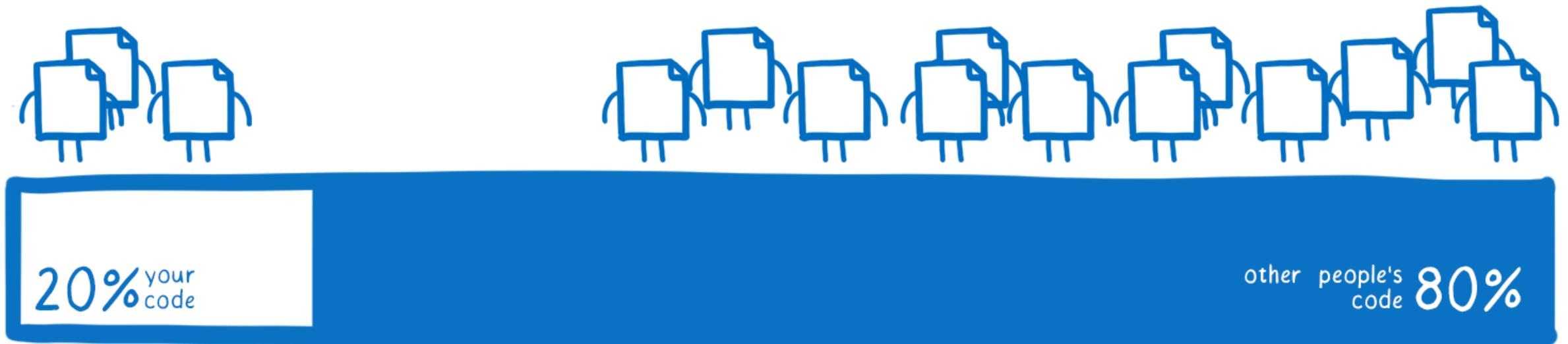
XKCD 2347



Agenda

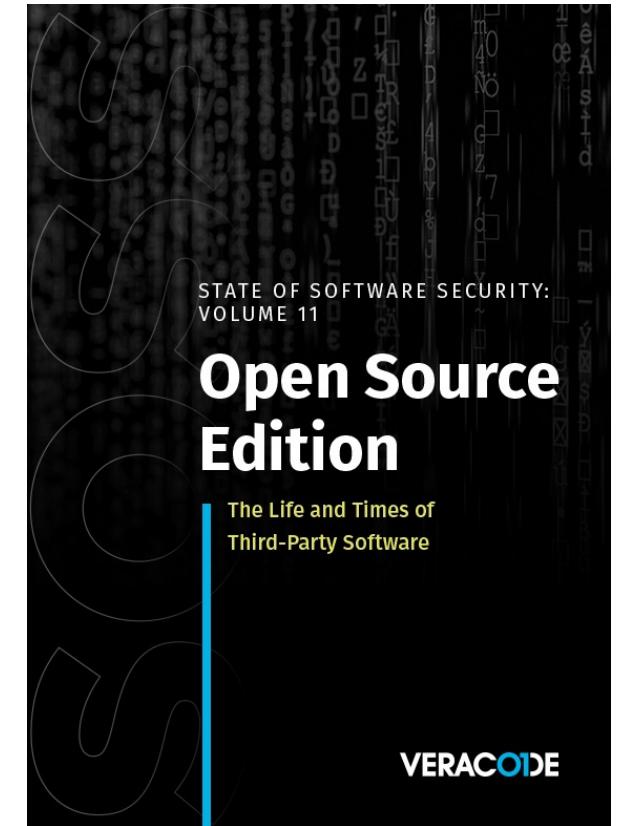
- Risks in 3rd NuGet Package
- OpenSFF Scorecard
- New & Improved
- Conclusion - Q&A

Average codebase composition



State of Software Security v11

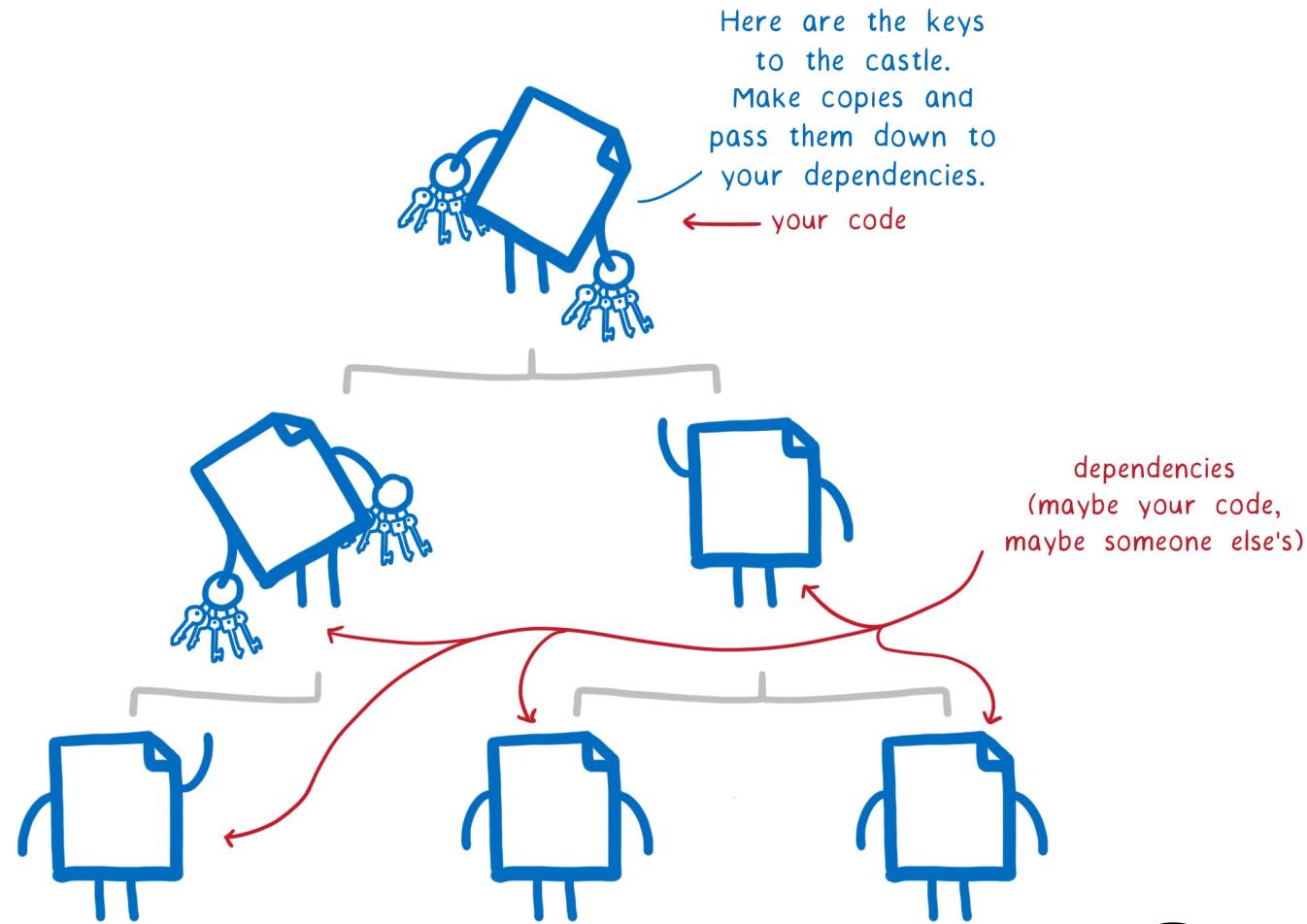
*”Despite this dynamic landscape,
79 percent of the time, developers
never update third-party libraries after
including them in a codebase.”*



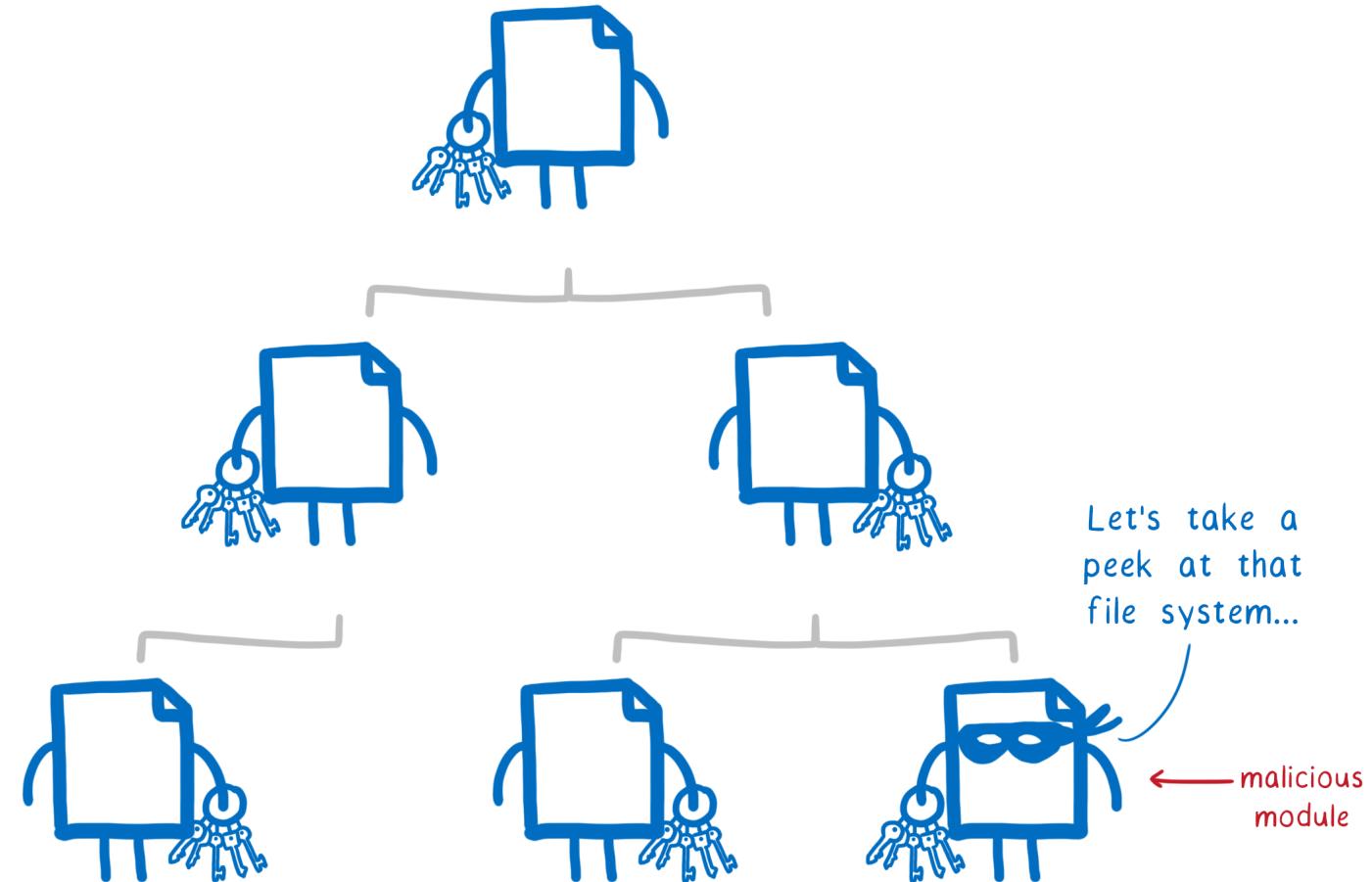
State of Log4j - 2 years later

- 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.

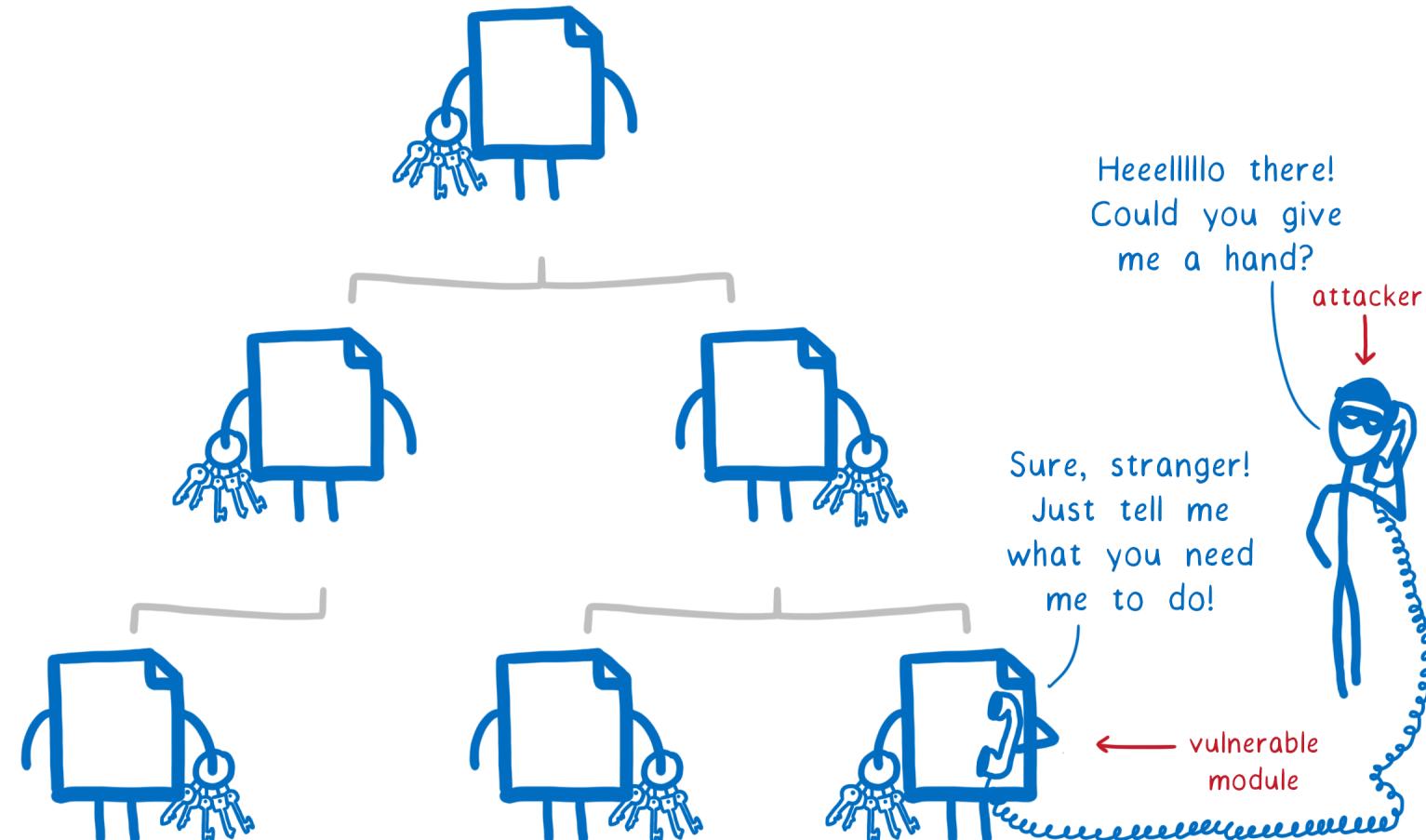
Average codebase composition



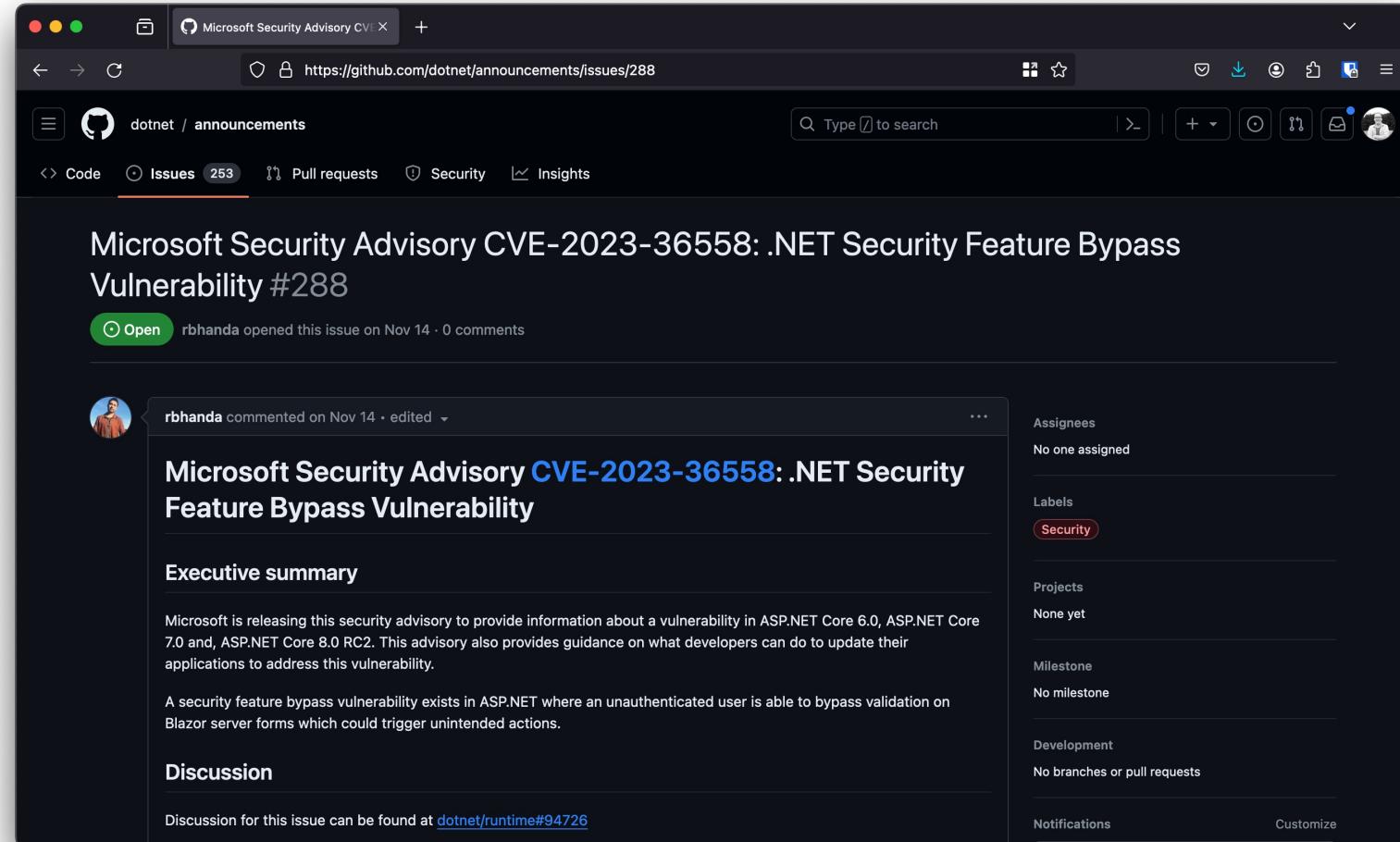
Malicious Assembly



Vulnerable Assembly



Vulnerabilities in Libraries



DotNet CLI

```
nelson@ghost-m2 ~$ dotnet list package
Project 'consoleapp' has the following package references
[net8.0]:
Top-level Package      Requested    Resolved
> docgenerator        1.0.0        1.0.0

nelson@ghost-m2 ~$ dotnet list package --vulnerable

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

The given project `consoleapp` has no vulnerable packages given the current sources.
```

DotNet CLI

```
nelson@ghost-m2 ~$ dotnet list package --include-transitive
Project 'consoleapp' has the following package references
[net8.0]:
Top-level Package      Requested   Resolved
> docgenerator        1.0.0       1.0.0

Transitive Package                                     Resolved
> iText7                                         7.2.2
> iText7.common                                     7.2.2
> Microsoft.CSharp                                4.0.1
> Microsoft.DotNet.PlatformAbstractions          1.1.0
> Microsoft.Extensions.DependencyInjection           5.0.0
> Microsoft.Extensions.DependencyInjection.Abstractions 5.0.0
> Microsoft.Extensions.DependencyModel            1.1.0
> Microsoft.Extensions.Logging                     5.0.0
> Microsoft.Extensions.Logging.Abstractions         5.0.0
> Microsoft.Extensions.Options                   5.0.0
> Microsoft.Extensions.Primitives                5.0.0
```

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$
```

Malicious Package

The screenshot shows a web browser window with a dark theme. The title bar reads "Hackers target .NET developers X". The address bar shows the URL "https://www.bleepingcomputer.com/news/security/hackers-target-net-developers-with-malicious-". The main content area displays a news article titled "Hackers target .NET developers with malicious NuGet packages" by Sergiu Gatlan, published on March 20, 2023, at 03:22 PM, with 0 comments. The article discusses threat actors targeting .NET developers with cryptocurrency stealers delivered through the NuGet repository and impersonating multiple legitimate packages via typosquatting. It mentions three packages downloaded over 150,000 times and quotes JFrog security researchers Natan Nehorai and Brian Moussalli. The text also notes the use of typosquatting in package names.

Hackers target .NET developers with malicious NuGet packages

By [Sergiu Gatlan](#) March 20, 2023 03:22 PM 0

Threat actors are targeting and infecting .NET developers with cryptocurrency stealers delivered through the NuGet repository and impersonating multiple legitimate packages via typosquatting.

Three of them have been downloaded over 150,000 times within a month, according to JFrog security researchers Natan Nehorai and Brian Moussalli, who spotted this ongoing campaign.

While the massive number of downloads could point to a large number of .NET developers who had their systems compromised, it could also be explained by the attackers' efforts to legitimize their malicious NuGet packages.

"The top three packages were downloaded an incredible amount of times – this could be an indicator that the attack was highly successful, infecting a large amount of machines," the [JFrog security researchers said](#).

"However, this is not a fully reliable indicator of the attack's success since the attackers could have automatically inflated the download count (with bots) to make the packages seem more legitimate."

The threat actors also used typosquatting when creating their NuGet repository profiles to impersonate

Malicious Package

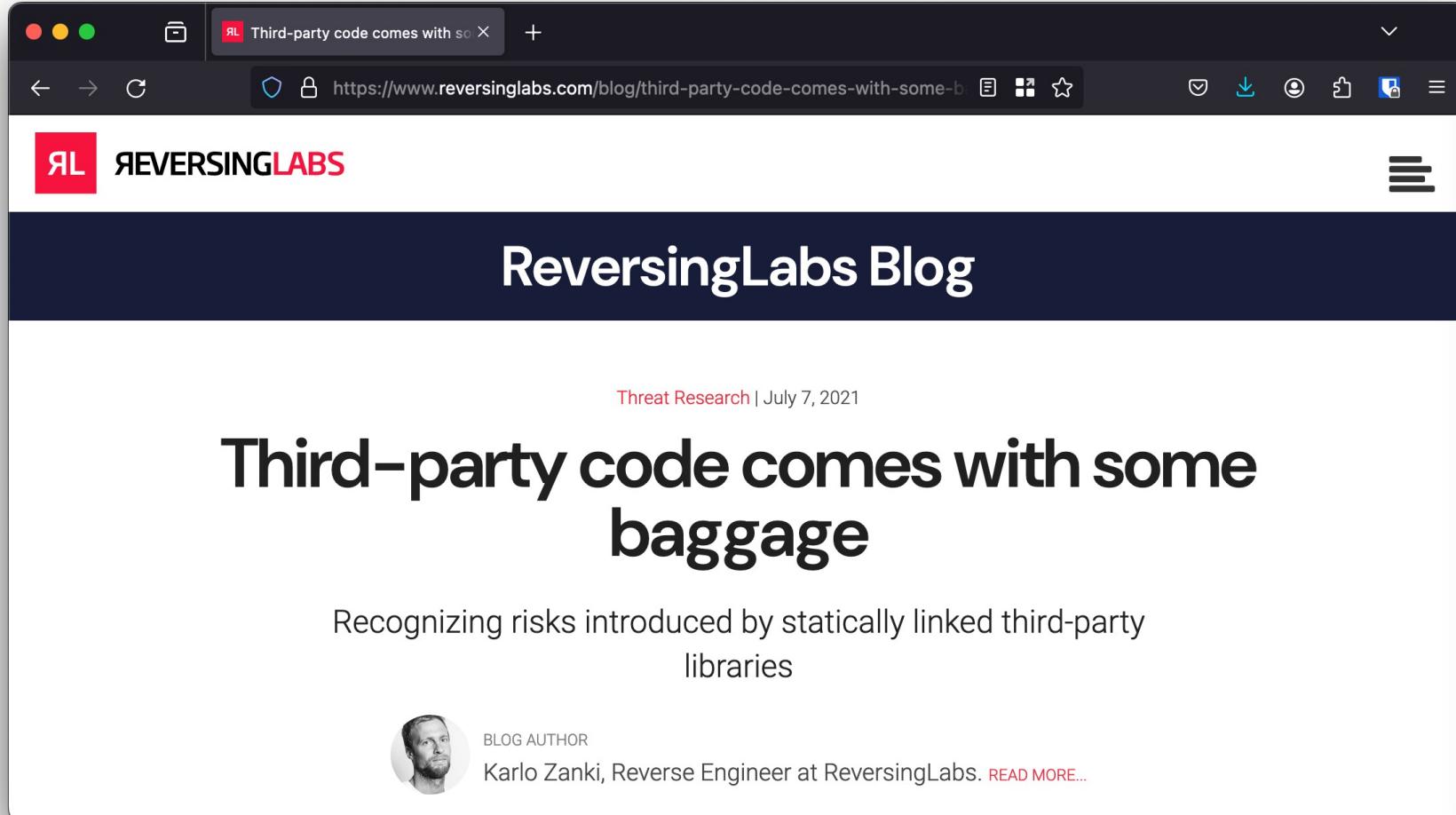
A screenshot of a web browser window showing a blog post from ReversingLabs. The title of the post is "IAmReboot: Malicious NuGet packages exploit loophole in MSBuild integrations". The post discusses threats in npm, PyPI, and RubyGEMS, stating that NuGet is also exposed to malicious activities. The browser's address bar shows the URL: <https://www.reversinglabs.com/blog/iamreboot-malicious-nuget-packages>.

Threat Research | October 31, 2023

IAmReboot: Malicious NuGet packages exploit loophole in MSBuild integrations

ReversingLabs has highlighted threats in npm, PyPI and RubyGEMS in recent years. This finding shows NuGet is equally exposed to malicious activities by threat actors.

Do you know what's inside?

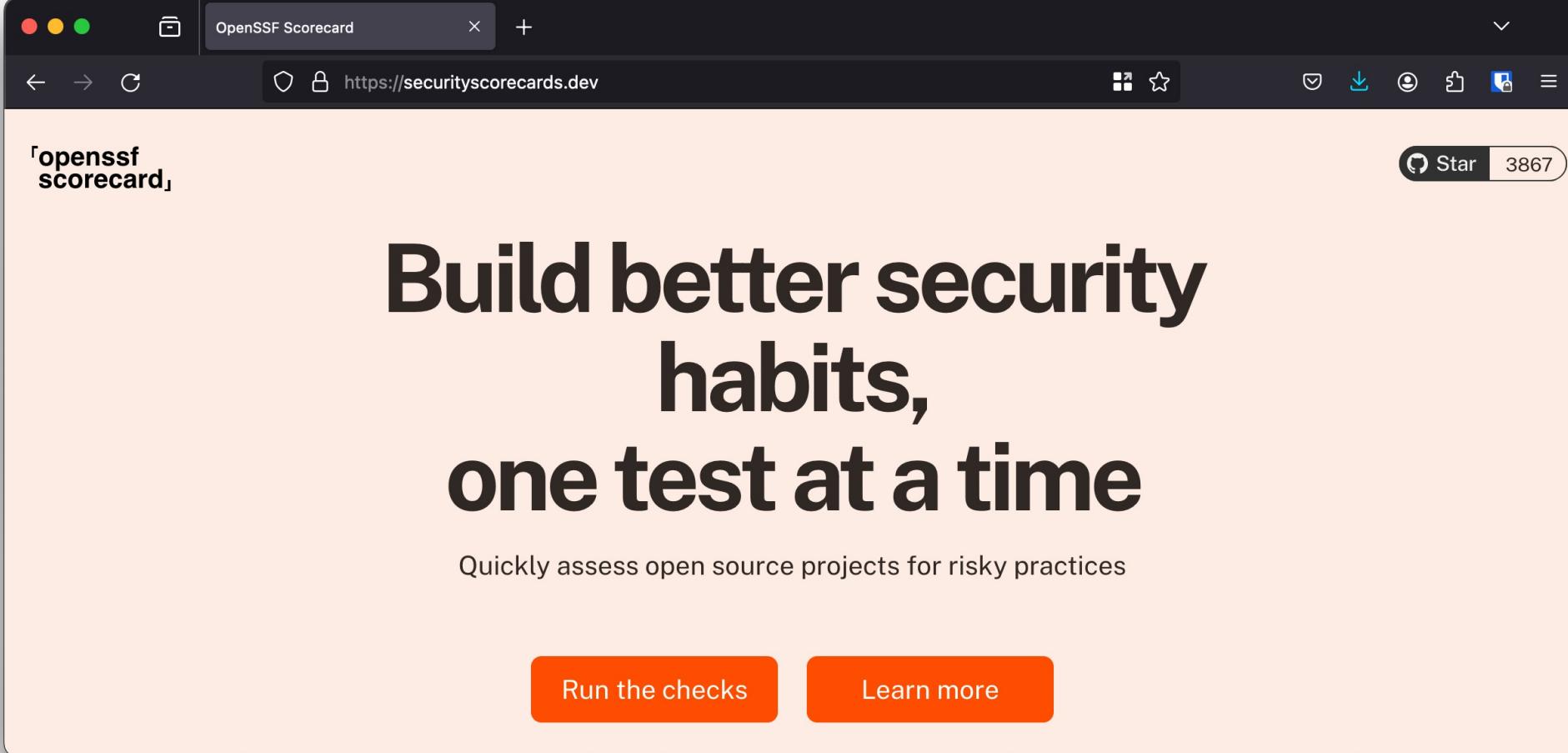


A screenshot of a web browser window showing a blog post from ReversingLabs. The browser has a dark mode interface. The tab bar shows a single tab titled "Third-party code comes with some baggage". The address bar displays the URL <https://www.reversinglabs.com/blog/third-party-code-comes-with-some-baggage>. 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". A circular profile picture of a man (Karlo Zanki) is shown next to the author's name, "BLOG AUTHOR Karlo Zanki, Reverse Engineer at ReversingLabs. [READ MORE...](#)". The overall theme is cybersecurity and threat research.

Nutrition Label for Software?

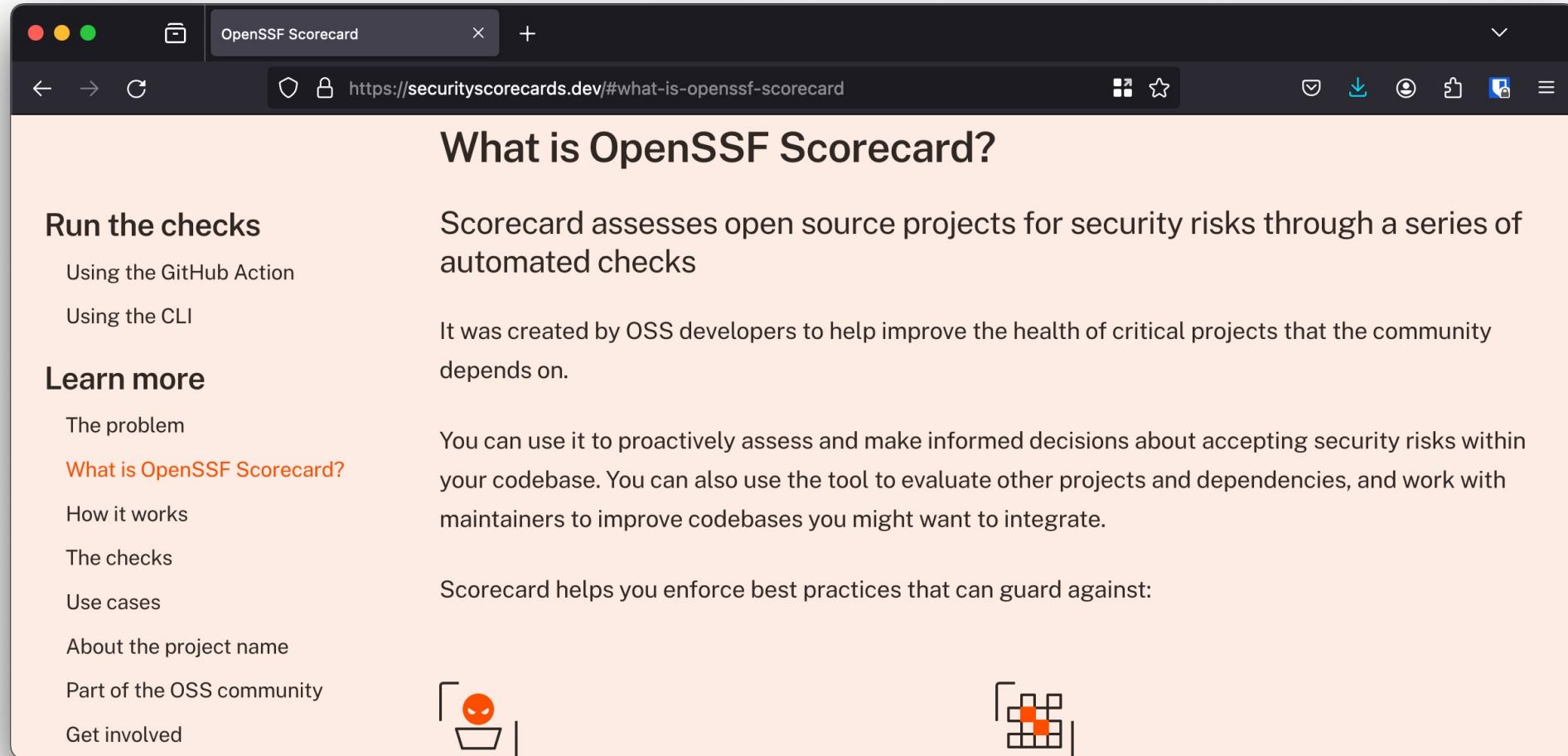


OpenSSF Scorecards



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

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, dark font. 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 two small icons at the bottom: one of a plant in a pot and one of a grid of squares.

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

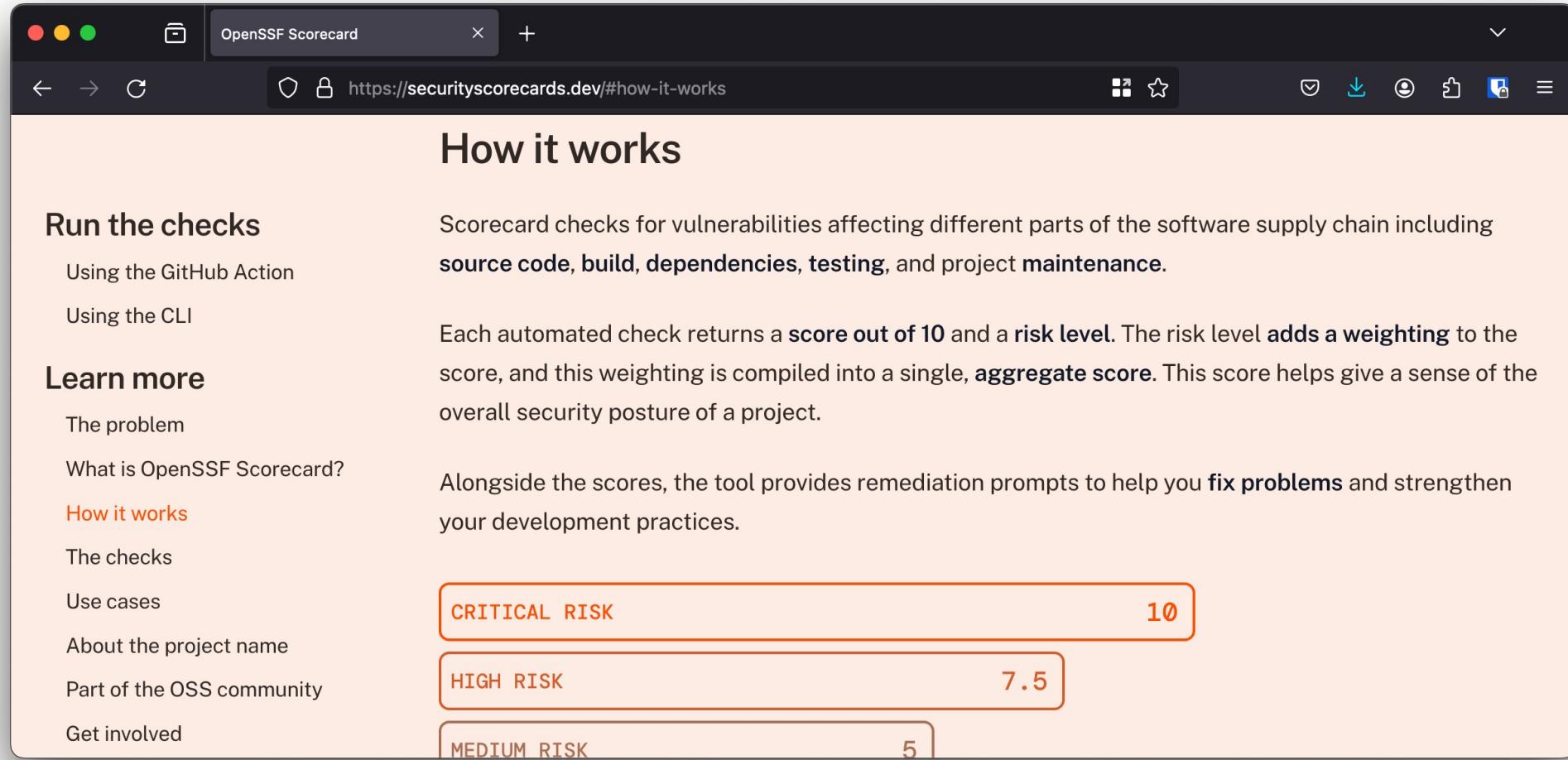
It was created by OSS developers to help improve the health of critical projects that the community depends on.

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 "How it works" section of the website at <https://securityscorecards.dev/#how-it-works>. The page has a light orange background. On the left, there's a sidebar with links under "Run the checks" and "Learn more". The main content area starts with a large heading "How it works". It then describes the scorecard's function: checking for vulnerabilities across source code, build, dependencies, testing, and project maintenance. It explains that each check returns a score out of 10 and a risk level, which is weighted to create an aggregate score. Below this, it mentions remediation prompts. At the bottom, there are three horizontal bars representing risk levels: CRITICAL RISK (10), HIGH RISK (7.5), and MEDIUM RISK (5).

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

How it works

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

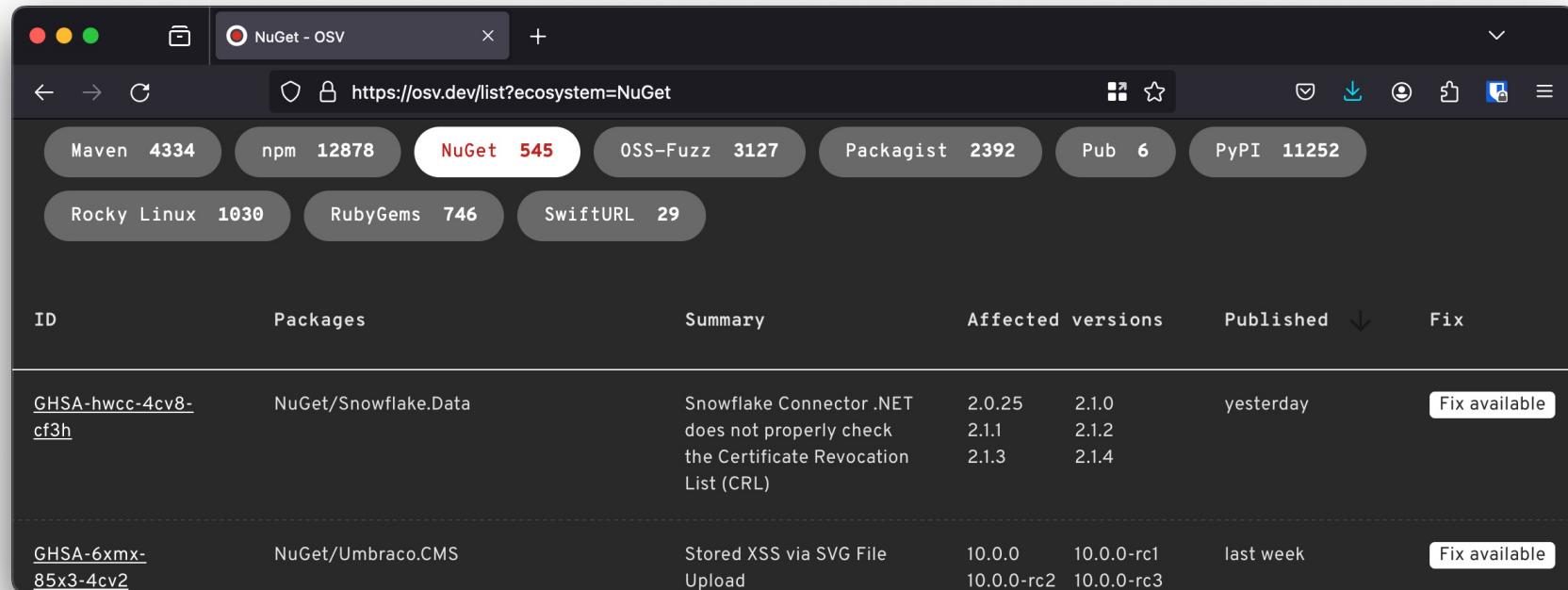
OpenSSF Security Scorecards

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" (problem, what is it, how it works, the checks, use cases, project name, community, get involved). To the right is a large graphic featuring six interconnected circles within a larger orange-bordered circle. The circles are labeled: "BUILD RISK ASSESSMENT", "CODE VULNERABILITIES", "MAINTENANCE", "CONTINUOUS TESTING", "SOURCE RISK ASSESSMENT", and "RISK ASSESSMENT". In the center of these circles is the text "HOLISTIC SECURITY PRACTICES" in orange.

- 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

Code Vulnerabilities (High)

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



The screenshot shows a dark-themed web browser window with the title "NuGet - OSV". The URL in the address bar is <https://osv.dev/list?ecosystem=NuGet>. Above the main table, there are several badge-like buttons representing different ecosystems and their counts: Maven (4334), npm (12878), NuGet (545), OSS-Fuzz (3127), Packagist (2392), Pub (6), PyPI (11252), Rocky Linux (1030), RubyGems (746), and SwiftURL (29). The main content is a table with the following columns: ID, Packages, Summary, Affected versions, Published (with a downward arrow), and Fix. Two rows are visible:

ID	Packages	Summary	Affected versions	Published	Fix	
GHSA-hwcc-4cv8-cf3h	NuGet/Snowflake.Data	Snowflake Connector .NET does not properly check the Certificate Revocation List (CRL)	2.0.25 2.1.1 2.1.3	2.1.0 2.1.2 2.1.4	yesterday	Fix available
GHSA-6xmx-85x3-4cv2	NuGet/Umbraco.CMS	Stored XSS via SVG File Upload	10.0.0 10.0.0-rc2	10.0.0-rc1 10.0.0-rc3	last week	Fix available

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

- This check tries to determine if the project uses a dependency update tool, use of: Dependabot, 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**)

- This check tries to determine if the project runs 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;
 - If there are user-defined language-specified fuzzing functions in the repository.
- Does it make sense to do fuzzing on .NET projects?

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/>

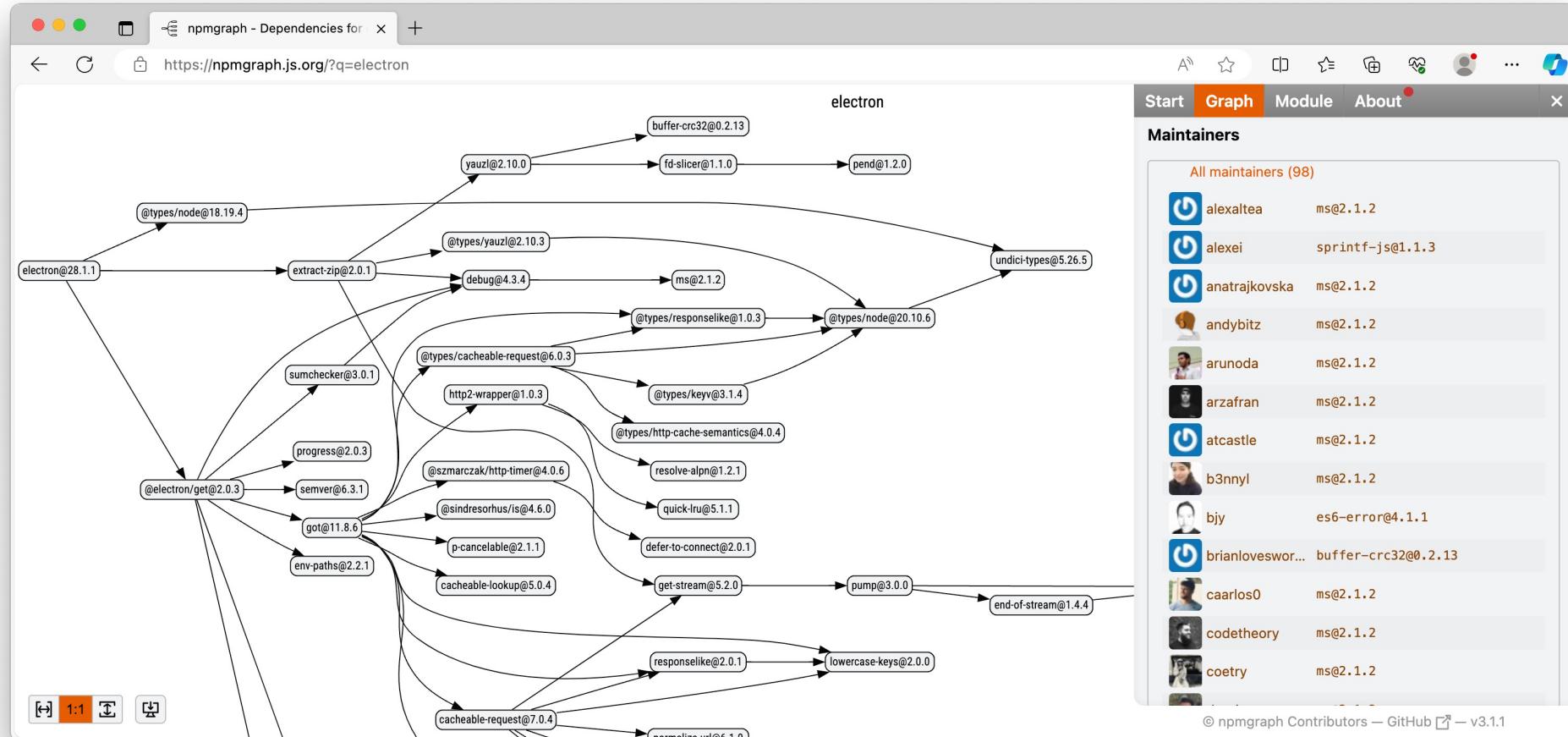
Source Risk Assessment Code Review (**Low**)

- This check determines whether the project requires human code review before pull requests (merge requests) are merged.
- The check determines whether the most recent changes (over the last ~30 commits) have an approval on GitHub or if the merger is different from the 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
- But is a large list of contributors good?

Source Risk Assessment Contributors (Low)



Build Risk Assessment

Pinned Dependencies (**High**)

- This check tries to determine if the project pins dependencies used during its build and release process.
- **RestorePackagesWithLockFile** in MSBuild results in packages.lock.json file containing versioned dependency tree with hashes

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



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**



What can we improve?



Fuzzing .NET



- Fuzzing, or fuzz testing, is defined as an automated software testing method that uses a wide range of *invalid* and unexpected data as input to find flaws in the software undergoing the test.
- Used a lot for finding C/C++ memory issues
- Can it be of any value with managed languages like .NET?

Fuzzing .NET & SharpFuzz



A screenshot of a web browser window showing a blog post. The title of the post is "Five years of fuzzing .NET with SharpFuzz". The post is dated Jul 23, 2023. The content discusses the history and evolution of SharpFuzz, mentioning its creation, initial capabilities, and recent improvements like support for libFuzzer, Windows, and .NET Framework, as well as the .NET Core base-class library. It also notes the simplification of the fuzzing process. The browser interface includes a navigation bar, a toolbar with various icons, and a sidebar on the left.

Fuzzing .NET & SharpFuzz



The screenshot shows a web browser window with the title "Five years of fuzzing .NET with SharpFuzz". The page content is titled "Trophies" and discusses the growth of bugs found by SharpFuzz, mentioning several specific bugs like BigInteger.TryParse out-of-bounds access and Double.Parse throwing AccessViolationException on .NET Core 3.0. It also notes two CVEs related to .NET Denial of Service Vulnerabilities. The browser interface includes standard navigation buttons, a search/address bar with the URL https://mijailovic.net/2023/07/23/sharpfuzz-anniversary/, and a toolbar with various icons.

Trophies

The list of bugs found by SharpFuzz has been growing steadily and it now contains more than 80 entries. I'm pretty confident that some of the bugs in the .NET Core standard library would have been impossible to discover using any other testing method:

- [BigInteger.TryParse out-of-bounds access](#)
- [Double.Parse throws AccessViolationException on .NET Core 3.0](#)
- [G17 format specifier doesn't always round-trip double values](#)

As you can see, SharpFuzz is capable of finding not only crashes, but also correctness bugs—the more creative you are in writing your fuzzing functions, the higher your chances are for finding an interesting bug.

SharpFuzz can also find serious security vulnerabilities. I now have two CVEs in my trophy collection:

- [CVE-2019-0980: .NET Framework and .NET Core Denial of Service Vulnerability](#)
- [CVE-2019-0981: .NET Framework and .NET Core Denial of Service Vulnerability](#)

If you were ever wondering if fuzzing managed languages makes sense, I think you've got your answer right here.

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

The screenshot shows a web browser window with the URL <https://research.kudelskisecurity.com/2023/12/07/introducing-fuzzomatic-using-ai-to-automatically-fuzz-rust-projects-from-scratch/>. The page title is "Introducing Fuzzomatic: Using / X". The main content starts with a section titled "How does it work?". It explains that Fuzzomatic relies on libFuzzer and cargo-fuzz as a backend, and uses AI and deterministic techniques. It mentions the use of OpenAI API models like gpt-3.5-turbo and gpt-3.5-turbo-16k. Below this, there's a section titled "Fuzz targets and coverage-guided fuzzing" which includes a code snippet:

```
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.

Comment Reblog Subscribe ...

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");
        }
    }
}
```

.NET Reproducibility



- Reproducible builds are a set of software development practices that create an independently-verifiable path from source to binary code.
- .NET Roslyn Deterministic Inputs
- How reproducible is a simple console app?

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 application characteristics by feature group. Below this is a "Feature Groups" section containing ten categories with their respective icons: Select Features, General Features, Development, Active Content, Data Storage, Sensitive Data, Cloud Services, OS Integration, OS System Changes, and Other. To the right of this 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

Community Review



The screenshot shows a dark-themed web browser window displaying the [Cargo Vet](https://mozilla.github.io/cargo-vet/) documentation. The page title is "Cargo Vet". The left sidebar contains a table of contents with sections 1. Introduction, 2. Tutorial, and 3. Reference. The main content area starts with a heading "Cargo Vet" and a paragraph explaining the tool's purpose: "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." It then describes how the tool matches project dependencies against audits from authors or trusted entities. The page also lists several key ways the tool reduces developer effort:

- **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 out to security review a NuGet 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!
- Room for .NET specific improvements with Fennec CLI & contributions to OpenSSF Scorecard project

Questions?

- <https://github.com/nielstanis/ndcsecurity2024/>
- ntanis at Veracode.com
- @nielstanis@infosec.exchange
- <https://www.fennec.dev> & <https://blog.fennec.dev>
- Takk! Thank you!