



Reviewing NuGet Packages security easily using OpenSSF Scorecard

Niels Tanis
Sr. Principal Security Researcher



PLATINUM



GOLD



PARTNER



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

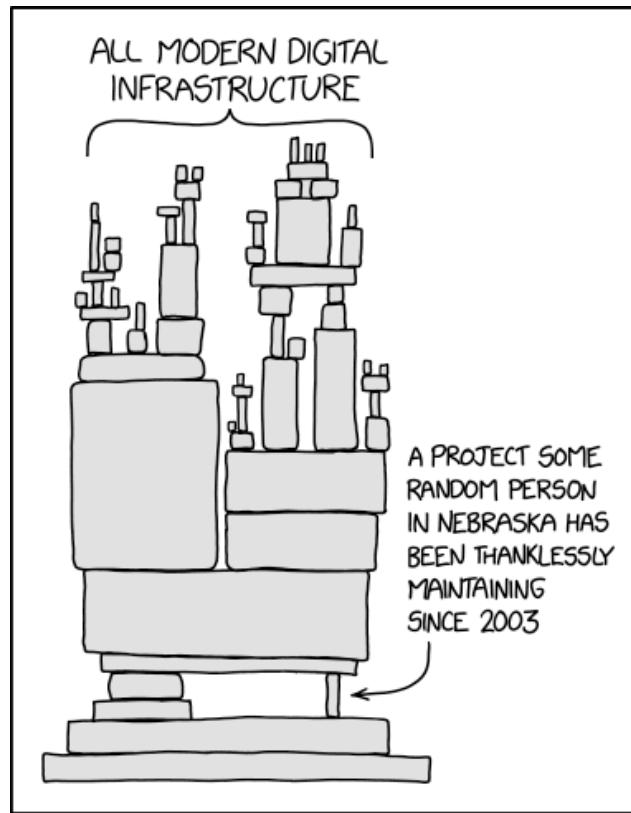
VERACODE



@nielstanis@infosec.exchange

Modern Application Architecture

XKCD 2347

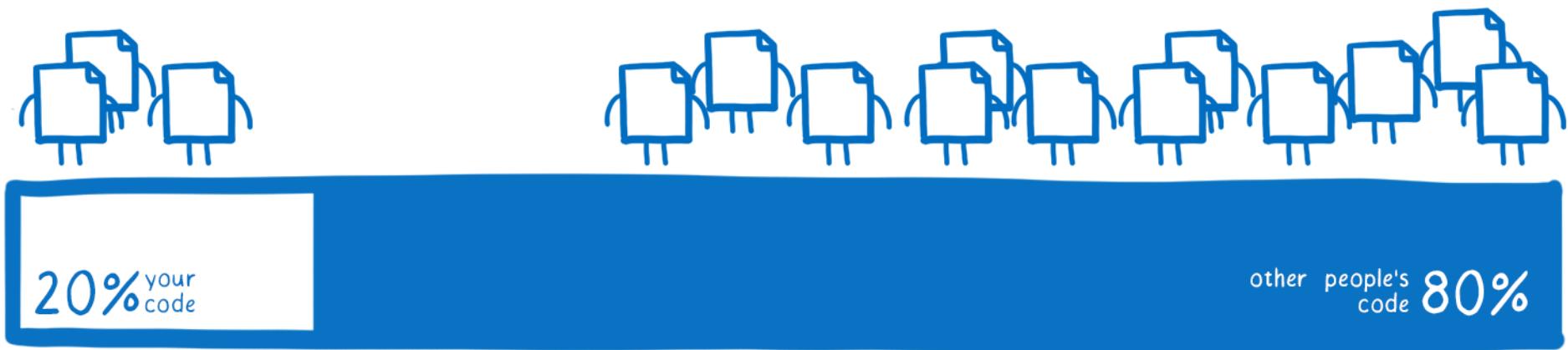


Agenda

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

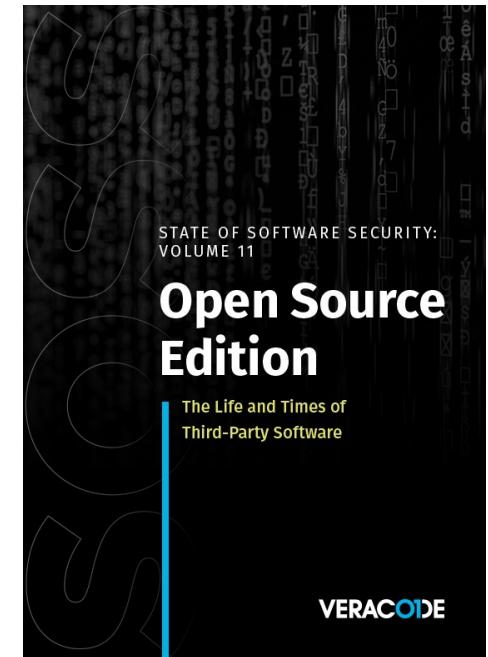


Average codebase composition



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."*

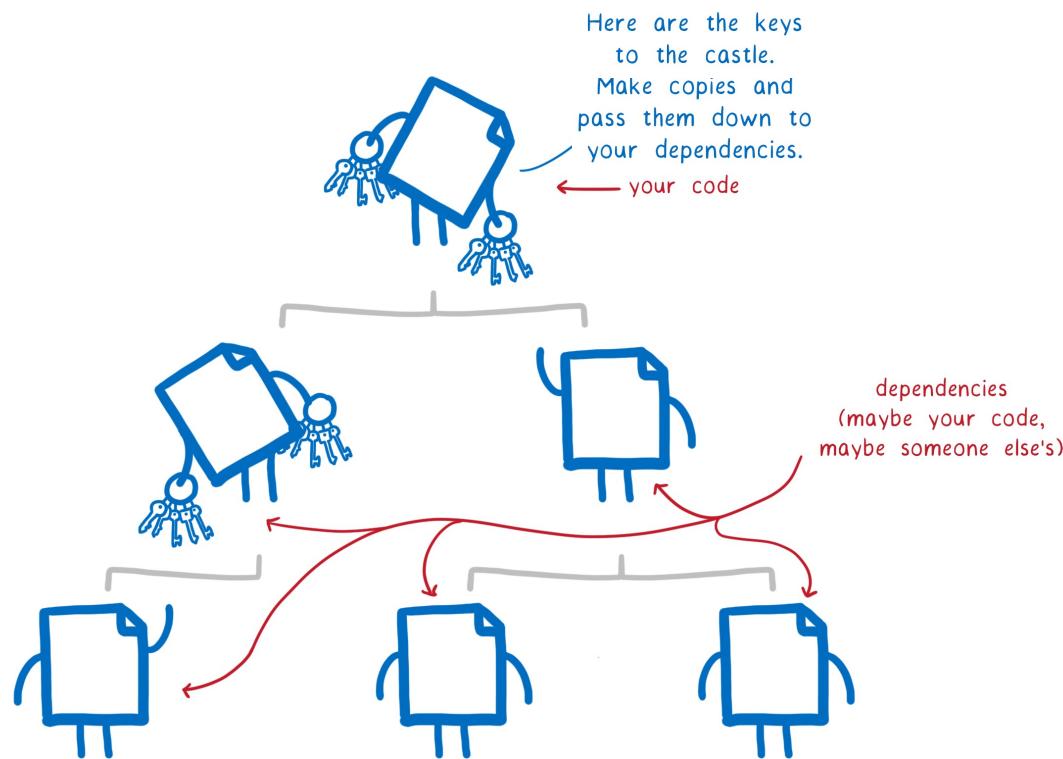


@nielstanis@infosec.exchange

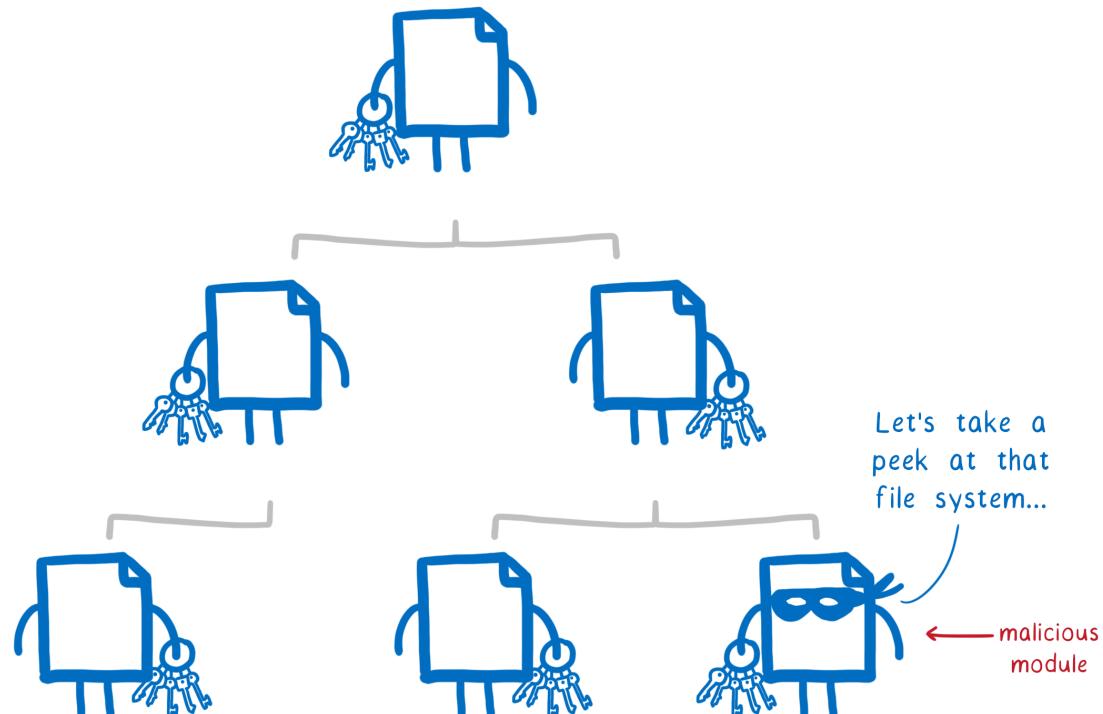
State of Log4j - 2 years later

- 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

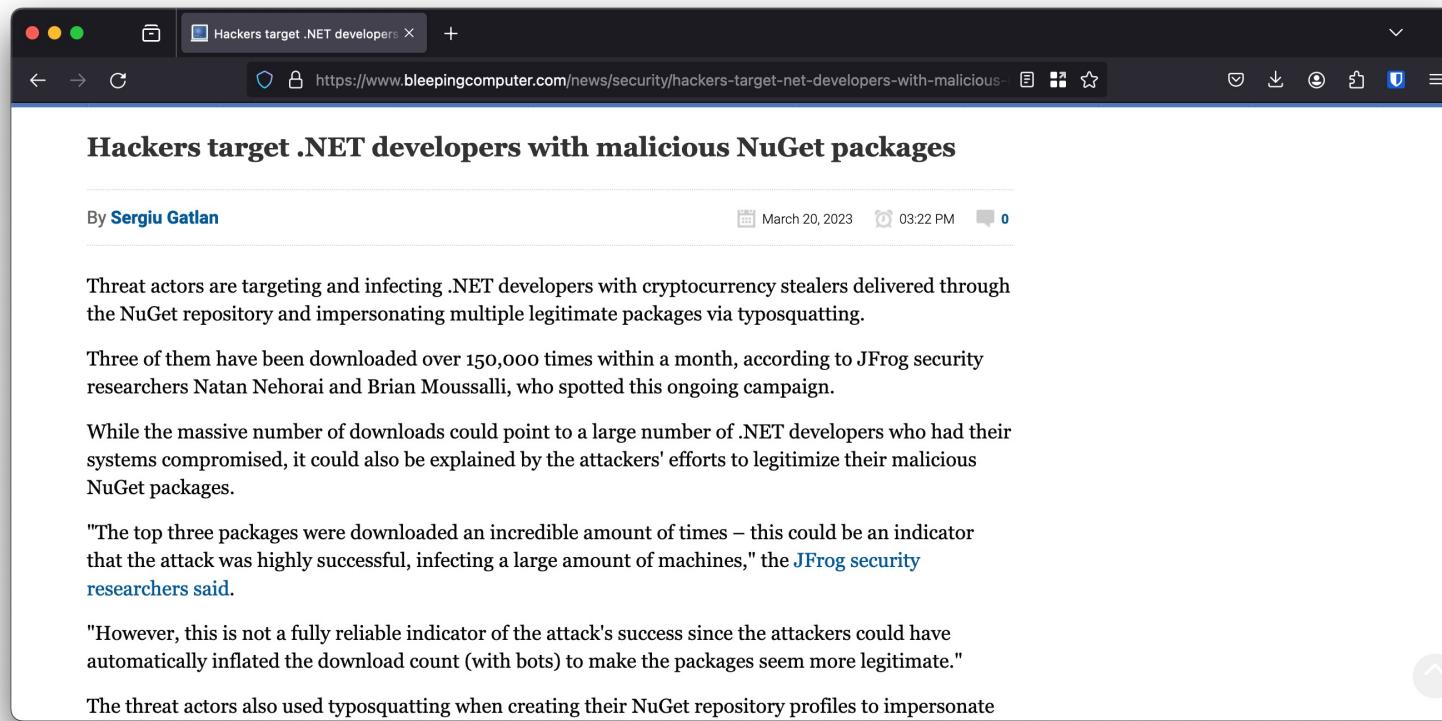


Malicious Assembly



@nielstanis@infosec.exchange

Malicious Package



A screenshot of a web browser window showing a news article from BleepingComputer.com. The title of the article is "Hackers target .NET developers with malicious NuGet packages". 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 browser interface includes a dark header bar, a tab labeled "Hackers target .NET developers", and various browser controls like back, forward, and search.

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 "Malicious NuGet campaign uses homoglyphs and IL weaving to fool devs". The post discusses how malware authors used homoglyphs and IL weaving to inject malicious code. The browser's address bar shows the URL: <https://www.reversinglabs.com/blog/malicious-nuget-campaign-uses-homoglyphs-and-il-weaving-to-fool-devs>. The page includes a sidebar with a search bar and a list of topics.

Threat Research | July 11, 2024

Malicious NuGet campaign uses homoglyphs and IL weaving to fool devs

Malware authors upped their game, using homoglyphs to impersonate a protected NuGet prefix and IL weaving to inject malicious code, RL researchers found.

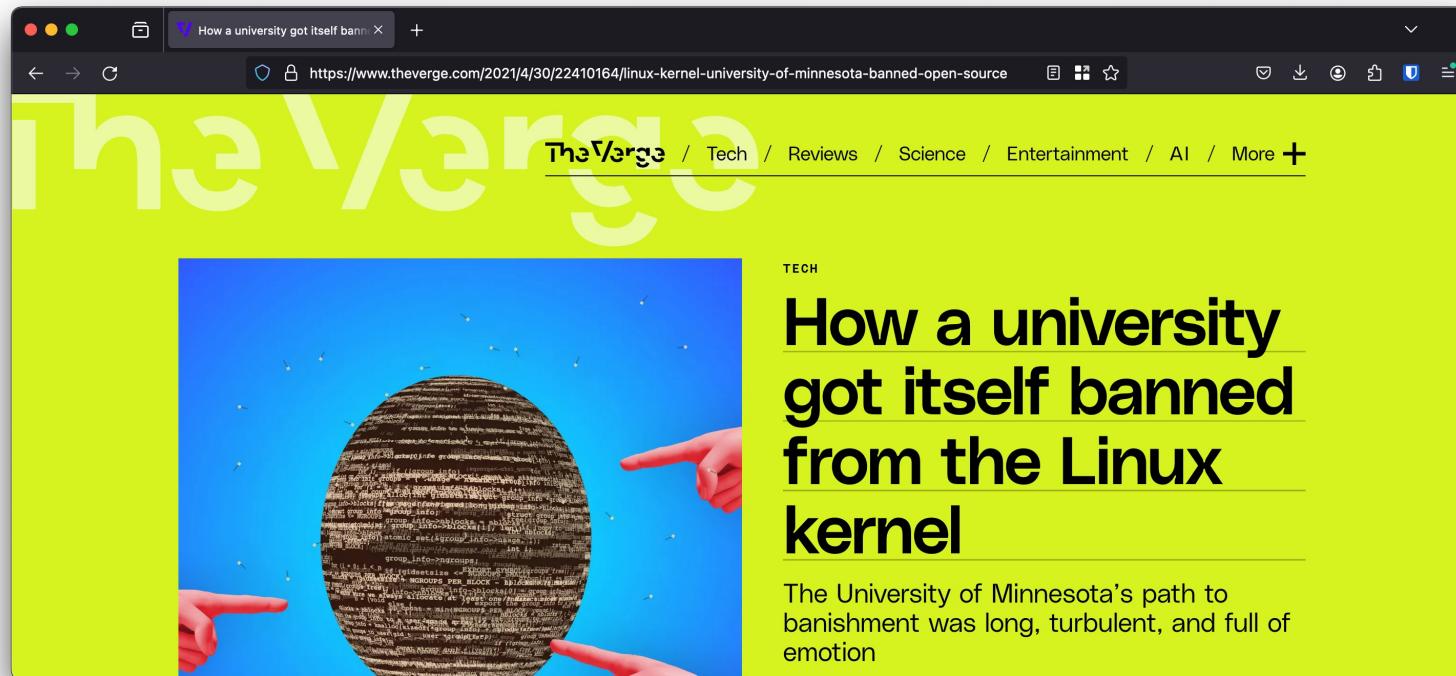
Topics

- All Blog Posts
- AppSec & Supply Chain
- Security
- Dev & DevSecOps
- Threat Research
- Security Operations
- Products & Technology
- Company & Events



@nielstanis@infosec.exchange

Hypocrite Commits



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 is categorized under "SUPPLY CHAIN ATTACK". Below the title, there is a short summary: "Malicious code planted in xz Utils has been circulating for more than a month." The author's name is DAN GOODIN, and the publication date is 3/29/2024, 7:50 PM.

ars TECHNICA

SUPPLY CHAIN ATTACK —

Backdoor found in widely used Linux utility targets encrypted SSH connections

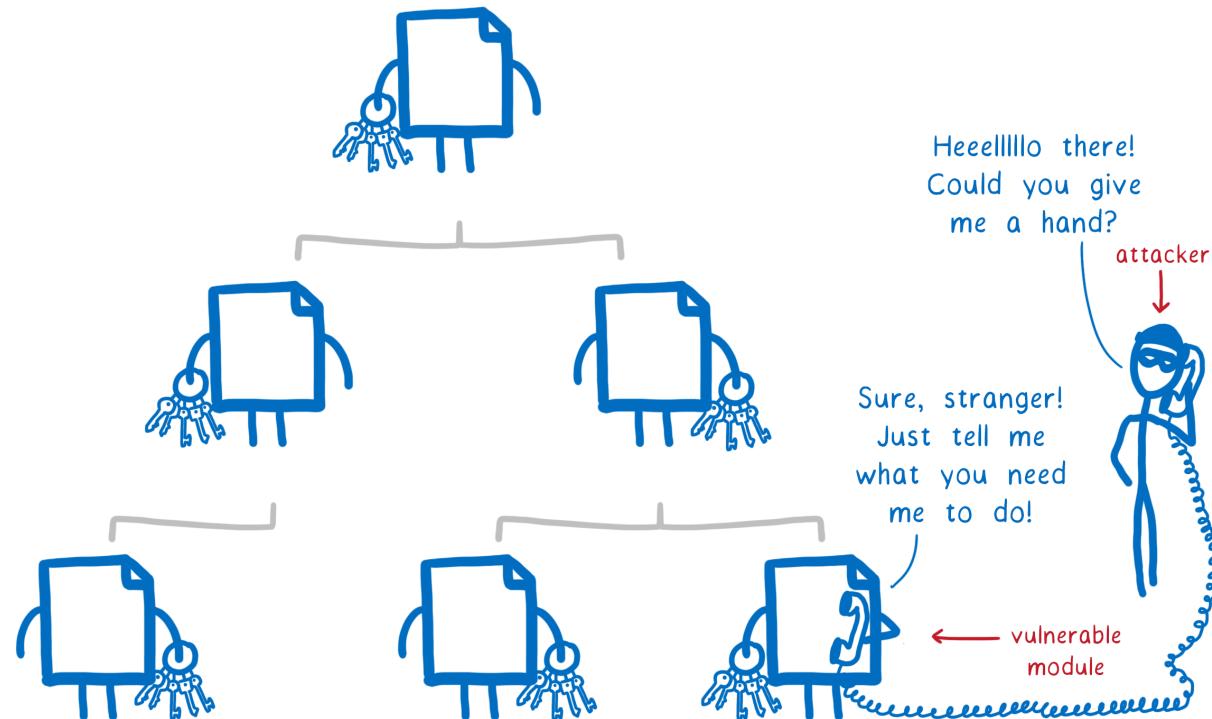
Malicious code planted in xz Utils has been circulating for more than a month.

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

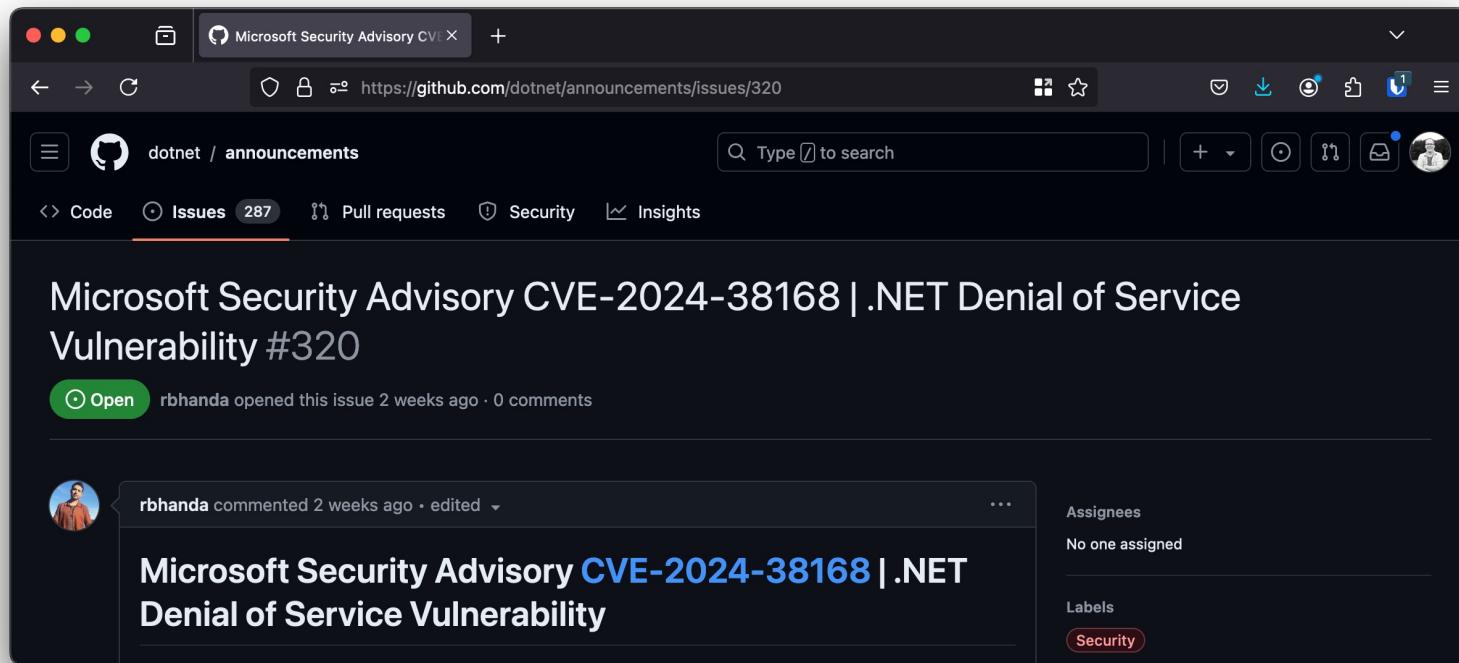


@nielstanis@infosec.exchange

Vulnerable Assembly



Vulnerabilities in Libraries



@nielstanis@infosec.exchange

DotNet CLI

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



@nielstanis@infosec.exchange

DotNet CLI

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



@nielstanis@infosec.exchange

DotNet CLI

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

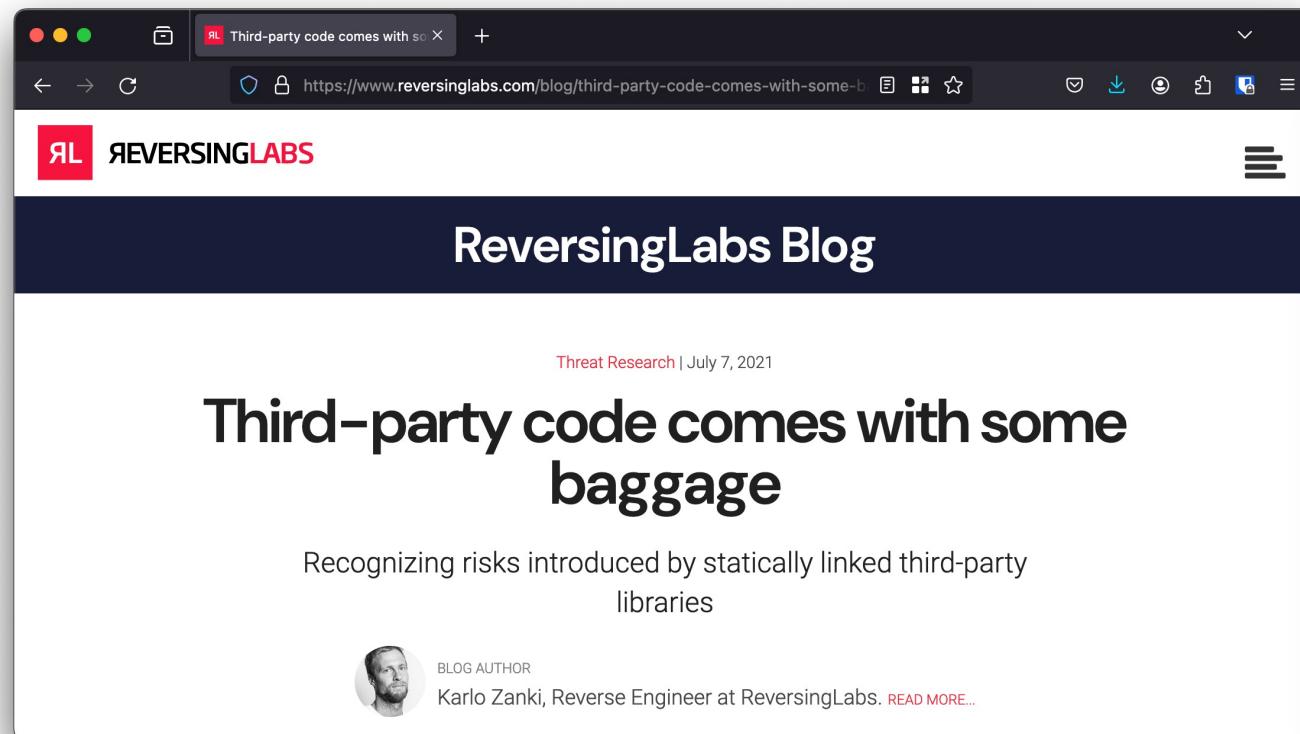
Project `consoleapp` has the following vulnerable packages
[net8.0]:
Transitive Package      Resolved    Severity   Advisory URL
> Newtonsoft.Json       9.0.1       High        https://github.com/advisories/GHSA-5crp-9r3c-p9vr

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



@nielstanis@infosec.exchange

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 page title is "Third-party code comes with some baggage" by Karlo Zanki. The URL in the address bar is <https://www.reversinglabs.com/blog/third-party-code-comes-with-some-b>. The page features the ReversingLabs logo at the top left and a navigation menu at the top right. The main content area includes the blog title, a brief description, and author information.

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...](#)

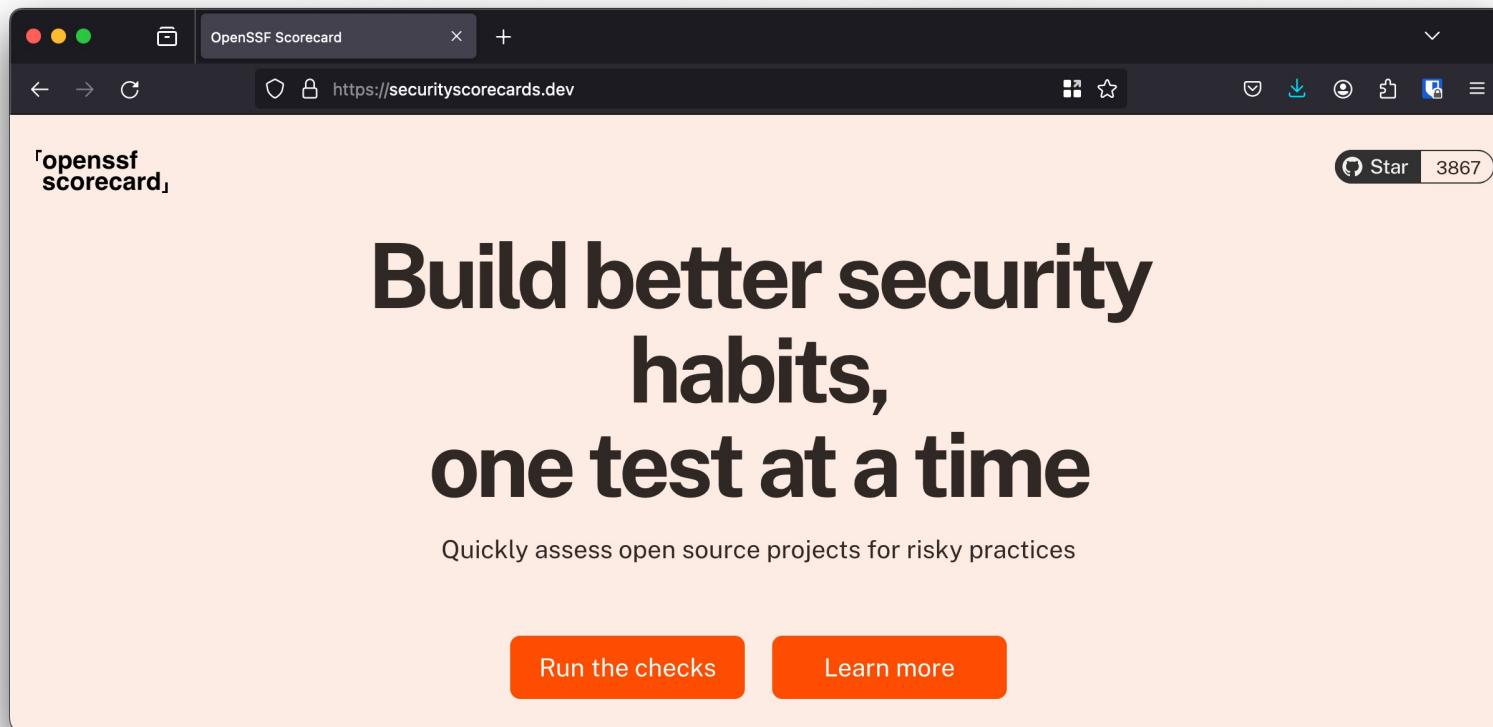


@nielstanis@infosec.exchange

Nutrition Label for Software?

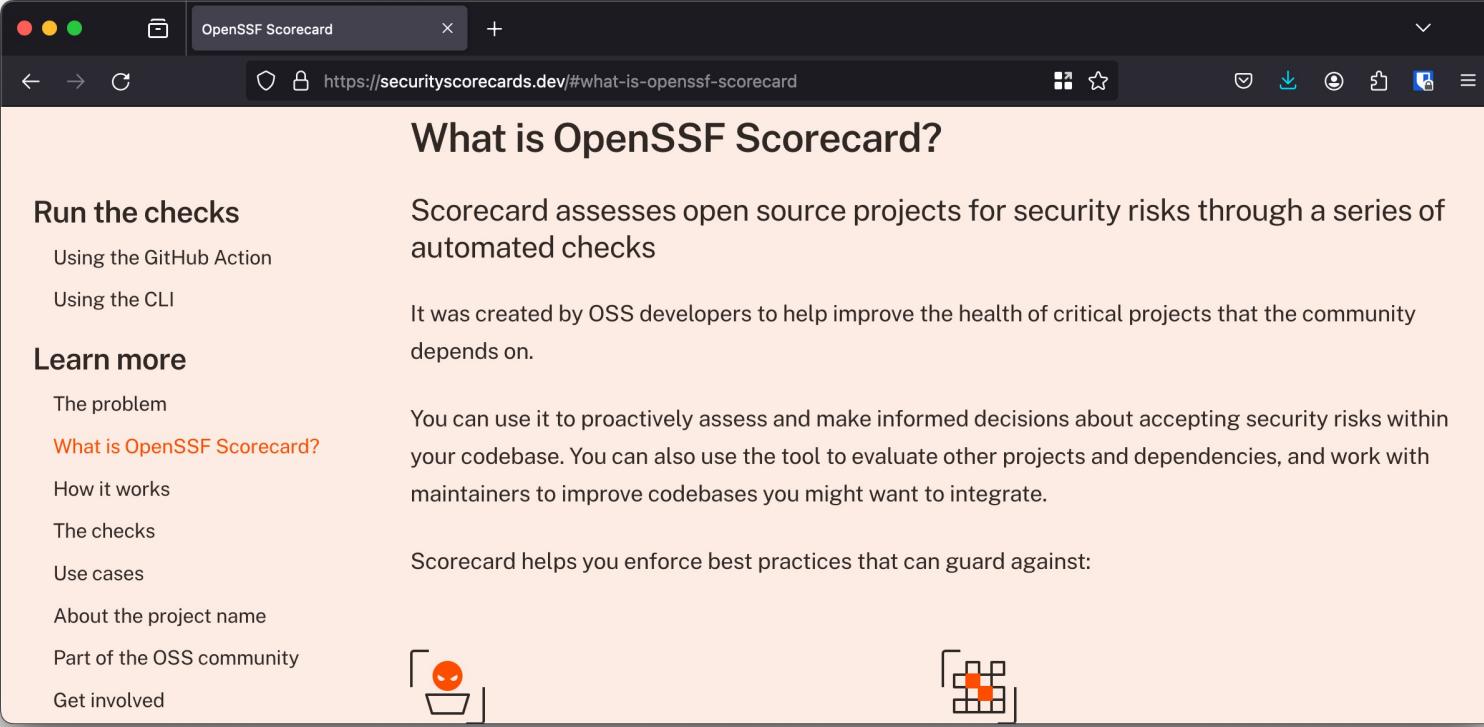


OpenSSF Scorecards



@nielstanis@infosec.exchange

OpenSSF Security Scorecards



The screenshot shows a web browser window with the title bar "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 following text and sections:

What is OpenSSF Scorecard?

Run the checks

- Using the GitHub Action
- Using the CLI

Learn more

- The problem
- What is OpenSSF Scorecard?** (This item is highlighted in orange)
- 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.

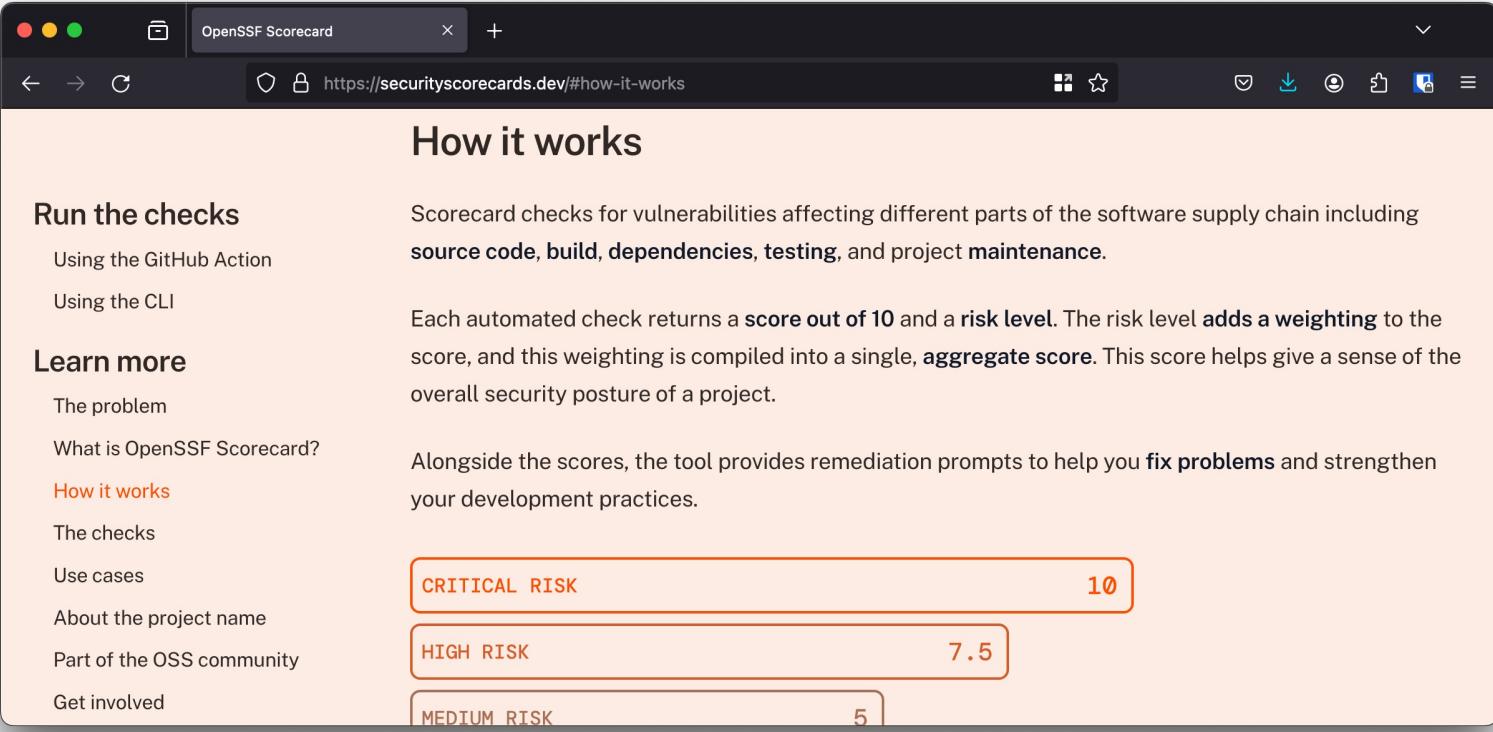
Scorecard helps you enforce best practices that can guard against:

Two small icons are shown at the bottom: a red face with a black outline and a grid of squares with one red square in the middle.



@nielstanis@infosec.exchange

OpenSSF Security Scorecards



The screenshot shows a web browser window with the title bar "OpenSSF Scorecard". The URL in the address bar is <https://securityscorecards.dev/#how-it-works>. The main content area has a light orange background and features the heading "How it works". Below this, there are two sections: "Run the checks" and "Learn more".

Run the checks

- Using the GitHub Action
- Using the CLI

Learn more

- The problem
- What is OpenSSF Scorecard?
- How it works** (This item is highlighted in red)
- The checks
- Use cases
- About the project name
- Part of the OSS community
- Get involved

On the right side of the page, there is a summary of the score results:

CRITICAL RISK	10
HIGH RISK	7.5
MEDIUM RISK	5

OpenSSF Security Scorecards

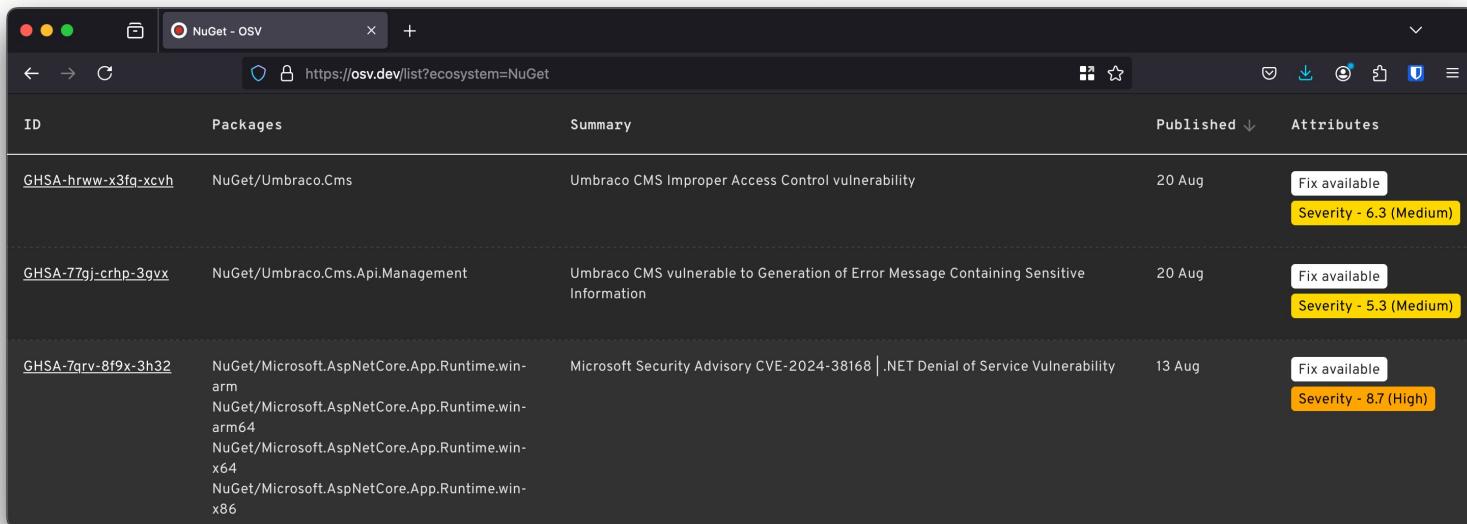
The screenshot shows a web browser window for the OpenSSF Scorecard. The URL is <https://securityscorecards.dev/#the-checks>. The page features a sidebar on the left with sections for "Run the checks" (GitHub Action, CLI), "Learn more" (problem, what is it, how it works, the checks, use cases, project name, community, get involved), and "The checks" (with links to each of the five categories shown in the diagram). The main content area contains a large diagram consisting of a large orange circle enclosing five smaller black circles. The circles are labeled: "CODE VULNERABILITIES" (top), "MAINTENANCE" (right), "CONTINUOUS TESTING" (bottom right), "SOURCE RISK ASSESSMENT" (bottom left), and "RISK ASSESSMENT" (left). In the center of the orange circle is the text "HOLISTIC SECURITY PRACTICES" in orange capital letters.



@nielstanis@infosec.exchange

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>. The page displays a table of vulnerabilities:

ID	Packages	Summary	Published	Attributes
GHSA-hrww-x3fq-xcvh	NuGet/Umbraco.Cms	Umbraco CMS Improper Access Control vulnerability	20 Aug	Fix available Severity - 6.3 (Medium)
GHSA-77gj-crhp-3gvx	NuGet/Umbraco.Cms.Api.Management	Umbraco CMS vulnerable to Generation of Error Message Containing Sensitive Information	20 Aug	Fix available Severity - 5.3 (Medium)
GHSA-7qrv-8f9x-3h32	NuGet/Microsoft.AspNetCore.App.Runtime.win-arm NuGet/Microsoft.AspNetCore.App.Runtime.win-arm64 NuGet/Microsoft.AspNetCore.App.Runtime.win-x64 NuGet/Microsoft.AspNetCore.App.Runtime.win-x86	Microsoft Security Advisory CVE-2024-38168 .NET Denial of Service Vulnerability	13 Aug	Fix available Severity - 8.7 (High)

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 .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/>

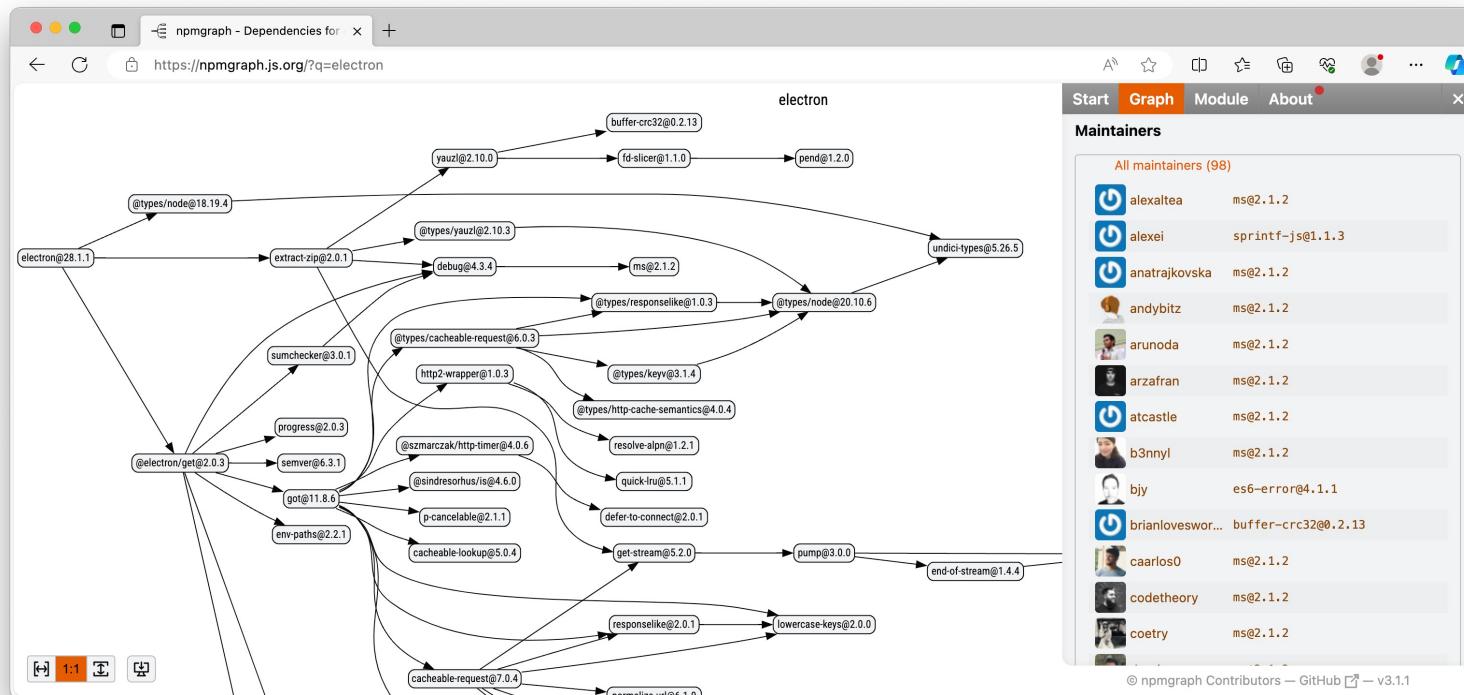
Source Risk Assessment Code Review (**Low**)

- This check determines whether the project requires human code review before pull requests 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 Assesement Pinned Dependencies (**High**)

- Does the project pin dependencies used during its build and release process.
- **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 Assessment 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



@nielstanis@infosec.exchange

Demo OpenSSF Scorecard Fennec CLI



@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**

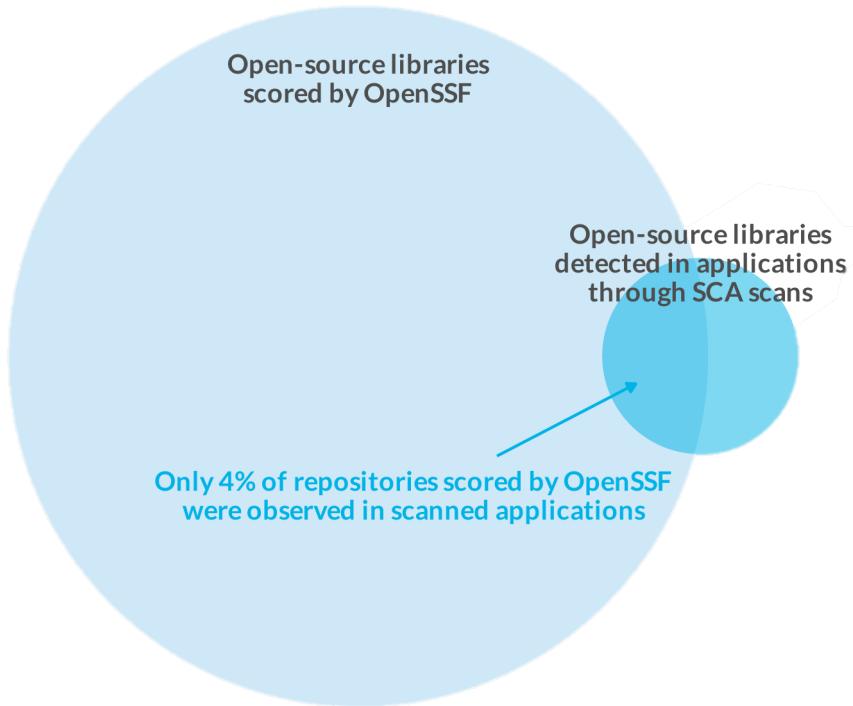


SOSS & OpenSSF Scorecard

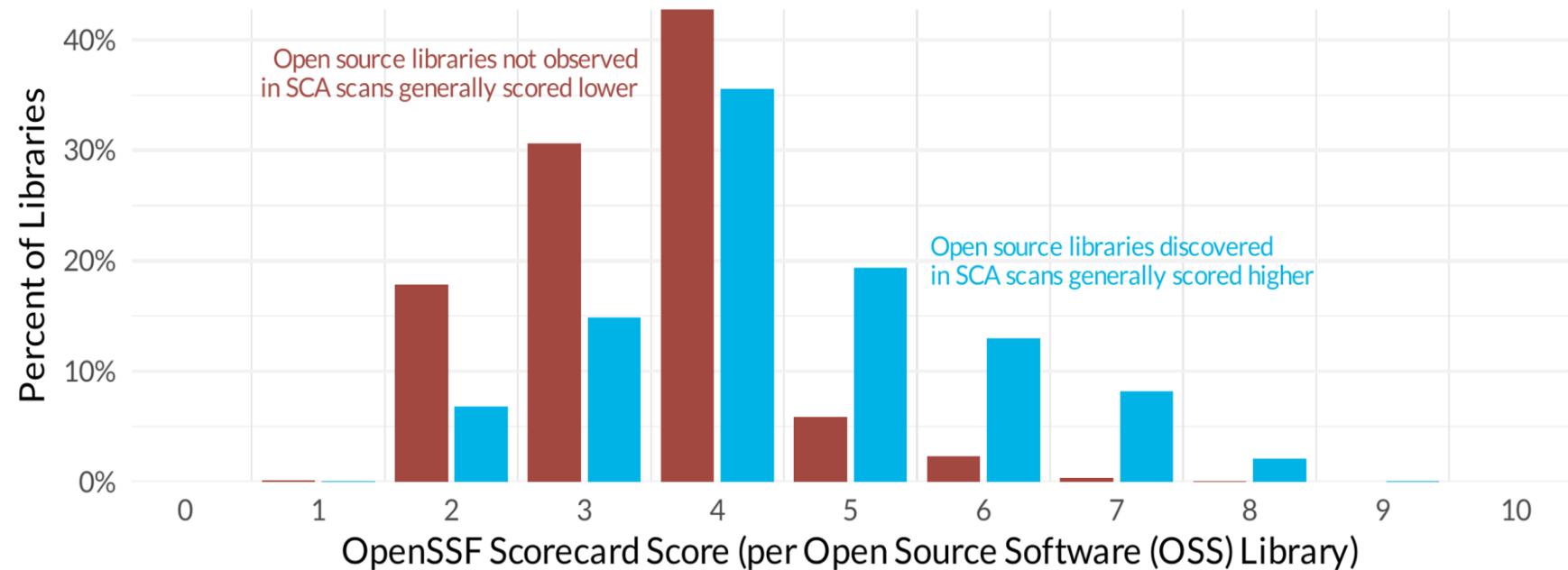


@nielstanis@infosec.exchange

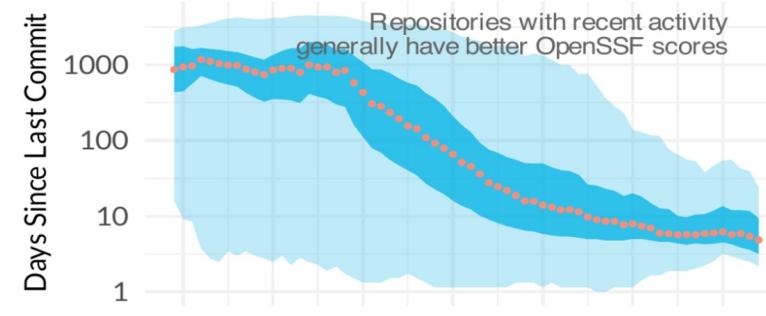
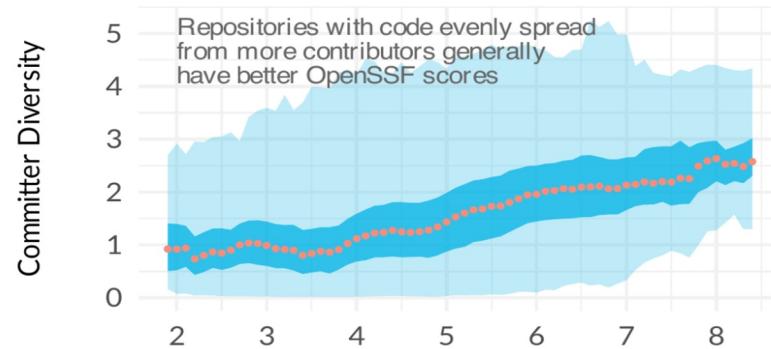
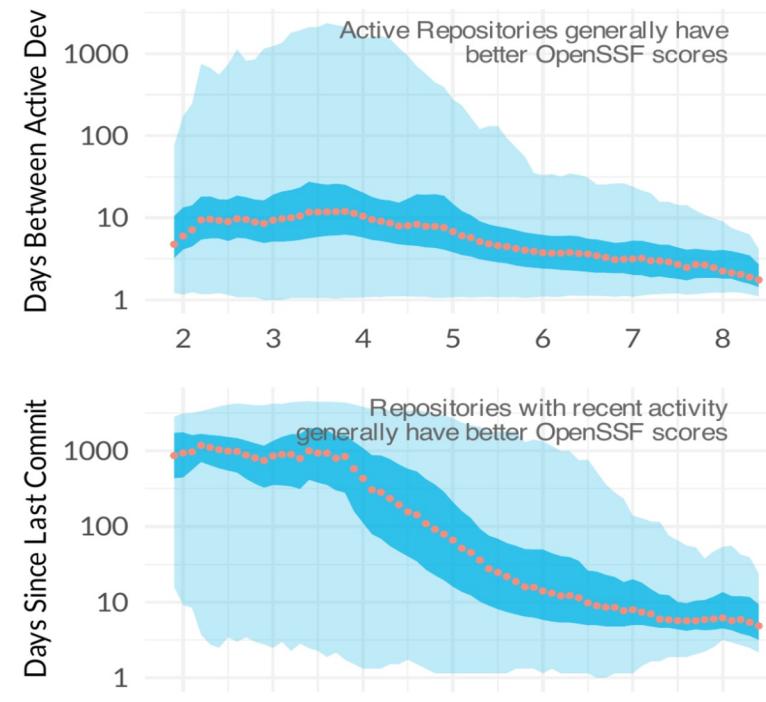
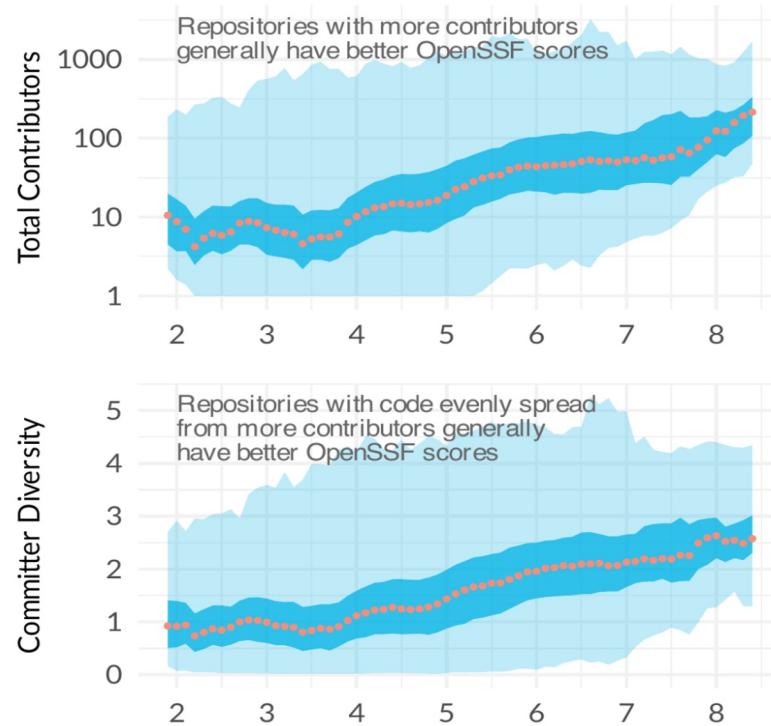
SOSS & OpenSSF Scorecard



Correlation between SOSS



Github commits vs OpenSSF



What really contributes to OSS Security?



What can we improve?



Fuzzing .NET



- 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?

Fuzzing .NET & SharpFuzz

New &
Improved!

The screenshot shows a web browser window with a dark theme. The title bar reads "Five years of fuzzing .NET with Shar X". The address bar shows the URL <https://mijailovic.net/2023/07/23/sharpfuzz-anniversary/>. The main content area is a blog post by "Nemanja Mijailovic's Blog". The title of the post is "Five years of fuzzing .NET with SharpFuzz". Below the title is the date "Jul 23, 2023". The post content discusses the history and evolution of SharpFuzz, mentioning its creation in 2018, its initial focus on libFuzzer and Windows, and its recent capabilities in fuzzing .NET Core base-class library. It also notes that many people are unaware of these developments.

Nemanja Mijailovic's Blog

Five years of fuzzing .NET with SharpFuzz

Jul 23, 2023

It's been almost five years since I created **SharpFuzz**, the only .NET coverage-guided fuzzer. I already have a blog post on how it works, what it can do for you, and what bugs it found, so check it out if this is the first time you hear about SharpFuzz:

[SharpFuzz: Bringing the power of afl-fuzz to .NET platform](#)

A lot of interesting things have happened since then. SharpFuzz now works with libFuzzer, Windows, and .NET Framework. And it can finally fuzz the .NET Core base-class library! The whole fuzzing process has been dramatically simplified, too.

Not many people are aware of all these developments, so I decided to write this anniversary blog post and showcase everything SharpFuzz is currently capable of.



@nielstanis@infosec.exchange

Fuzzing .NET & SharpFuzz



The screenshot shows a web browser window with the title bar "Five years of fuzzing .NET with SharpFuzz". The address bar shows the URL <https://mijailovic.net/2023/07/23/sharpfuzz-anniversary/>. The main content is a blog post titled "Trophies".

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) { }
    });
}
```



@nielstanis@infosec.exchange

Fuzzomatic: Using AI to Fuzz Rust

A screenshot of a web browser window displaying a blog post titled "Introducing Fuzzomatic: Using AI to automatically fuzz Rust projects from scratch". The post discusses how Fuzzomatic works, mentioning libFuzzer and cargo-fuzz as backends, and how it uses AI and deterministic techniques. It also notes the use of OpenAI API models like gpt-3.5-turbo and gpt-3.5-turbo-16k. A code snippet shows a Rust libFuzzer fuzz target. The browser interface includes a navigation bar, a sidebar with social sharing icons, and a footer with a "Comment", "Reblog", "Subscribe", and "..." button.

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.

Static Code Analysis (SAST)



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



@nielstanis@infosec.exchange

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 → independently-verifiable path from source to binary code.
- .NET Roslyn Deterministic Inputs
- How reproducible is a simple console app?
- Fennec Diff (work-in-progress)

Application Inspector

New &
Improved!

The screenshot shows the Microsoft Application Inspector interface. At the top, there's a navigation bar with links for Overview, Summary, Features, and About. Below this is a section titled "Application Features" which contains a detailed description of the application's characteristics and how to view additional details or expand feature groups. To the right of this description is a "Feature Groups" section listing various categories with corresponding icons. Further down is an "Associated Rules" section listing specific rules like "Authentication: Microsoft (Identity)", "Authentication: General", and "Authentication: (Oauth)".



@nielstanis@infosec.exchange

Application Inspector

New &
Improved!

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

New &
Improved!

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



@nielstanis@infosec.exchange

NuGet Blog

A screenshot of a Microsoft Dev Blogs page. The title of the post is "OpenSSF Scorecard for .NET and the NuGet ecosystem". The post was published on November 4th, 2024. It features three authors' profile pictures and their names: Ioana, Avishay, and Mélanie. Below the authors, there are two interactive icons: a blue heart icon with the number "1" and a blue speech bubble icon with the number "0". To the right of the post, there is a sidebar titled "Table of contents" which includes links to "Overview", "What is Scorecard and why should you use it", and "What are the checks run by Scorecard". A "Feedback" button is also visible in the sidebar. The URL in the browser bar is <https://devblogs.microsoft.com/nuget/openssf-scorecard-for-net-nuget/>.

Conclusion

- Scorecard helps security reviewing 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 its own!

Conclusion

- NuGet Package Scoring (NET Score)
- Room for .NET specific improvements with Fennec CLI
 - Tools (diff, insights)
 - Trust Graph
 - Contribute back to OpenSSF Scorecard



```
dotnet tool install -g fennec
```



@nielstanis@infosec.exchange

Questions?



Links & Feedback

- <https://github.com/nielstanis/devday2024>
- ntanis at Veracode.com
- @nielstanis@infosec.exchange
- <https://www.fennec.dev>
<https://blog.fennec.dev>



@nielstanis@infosec.exchange