



Reviewing NuGet Packages security easily using OpenSSF Scorecard

Niels Tanis
Sr. Principal Security Researcher



Who am I?



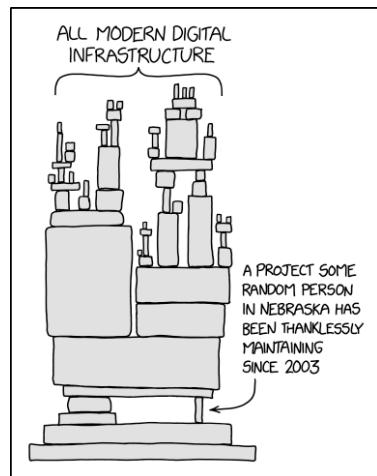
- Niels Tanis
- Sr. Principal Security Researcher
 - Background .NET Development, Pentesting/ethical hacking, and software security consultancy
 - Research on static analysis for .NET apps
 - Enjoying Rust!
- Microsoft MVP – Developer Technologies

VERACODE



@nielstanis@infosec.exchange

Modern Application Architecture XKCD 2347



@nielstanis@infosec.exchange

<https://xkcd.com/2347/>

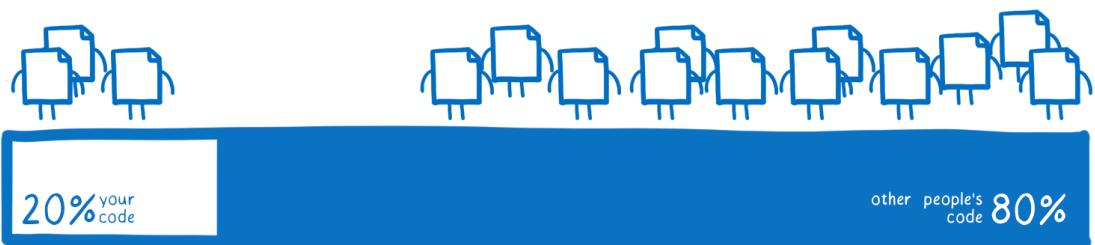
Agenda

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



@nielstanis@infosec.exchange

Average codebase composition



@nielstanis@infosec.exchange

<https://hacks.mozilla.org/2019/11/announcing-the-bytecode-alliance/>

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

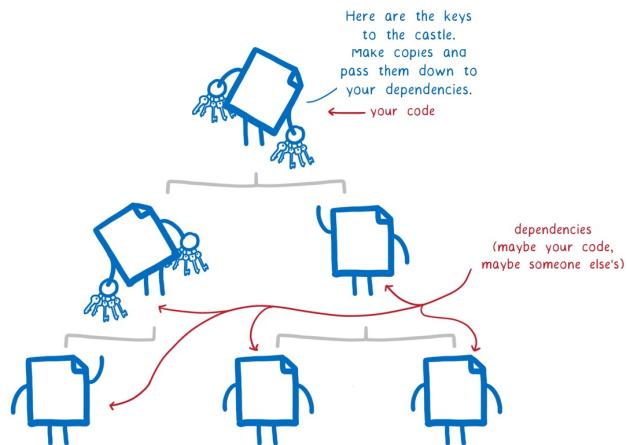


- Analysed our data August-November 2023
 - Total set of almost 39K unique applications scanned
- 2.8% run version vulnerable to Log4Shell
- 3.8% run version patched but vulnerable to other CVE
- 32% rely on a version that's end-of-life and have no support for any patches.

@nielstanis@infosec.exchange

<https://www.veracode.com/blog/research/state-log4j-vulnerabilities-how-much-did-log4shell-change>

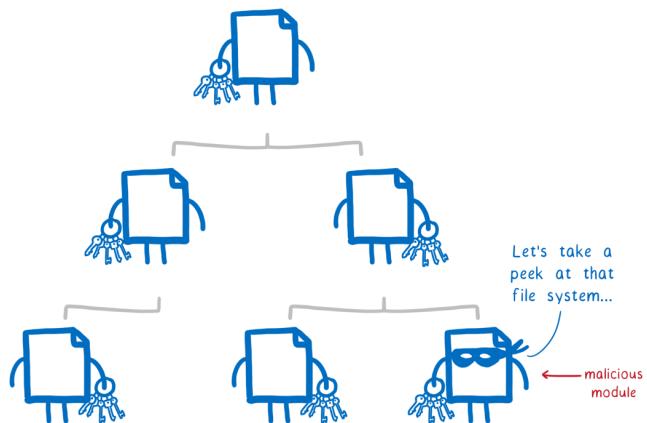
Average codebase composition



@nielstanis@infosec.exchange

<https://hacks.mozilla.org/2019/11/announcing-the-bytecode-alliance/>

Malicious Assembly



@nielstanis@infosec.exchange

<https://hacks.mozilla.org/2019/11/announcing-the-bytecode-alliance/>

The screenshot shows a Microsoft Edge browser window with a dark blue header. The header features the text "Malicious Package" on the left and the "dev .NET Noord" logo on the right. The main content area displays a news article from BleepingComputer.com. The title of the article is "Hackers target .NET developers with malicious NuGet packages". Below the title, it says "By Sergiu Gatlan" and "March 20, 2023 03:22 PM 0". 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 that over 150,000 packages have been downloaded within a month. Researchers Natan Nehorai and Brian Moussalli spotted the ongoing campaign. The text notes that while the massive number of downloads could point to a large number of .NET developers, it could also be explained by attackers' efforts to legitimize their malicious NuGet packages. Quotations from JFrog security researchers are included, along with a note about automatic download inflation by bots. At the bottom, it says "The threat actors also used typosquatting when creating their NuGet repository profiles to impersonate".

Malicious Package

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.

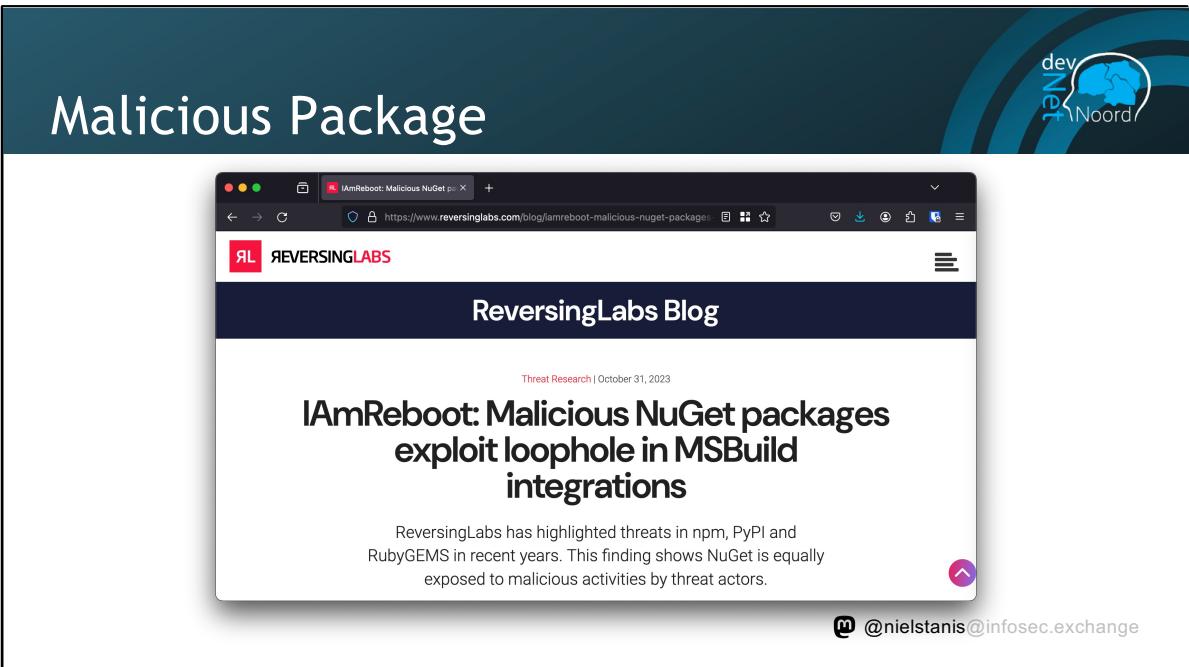
"The top three packages were downloaded an incredible amount of times – this could be an indicator that the attack was highly successful, infecting a large amount of machines," the [JFrog security researchers said](#).

"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."

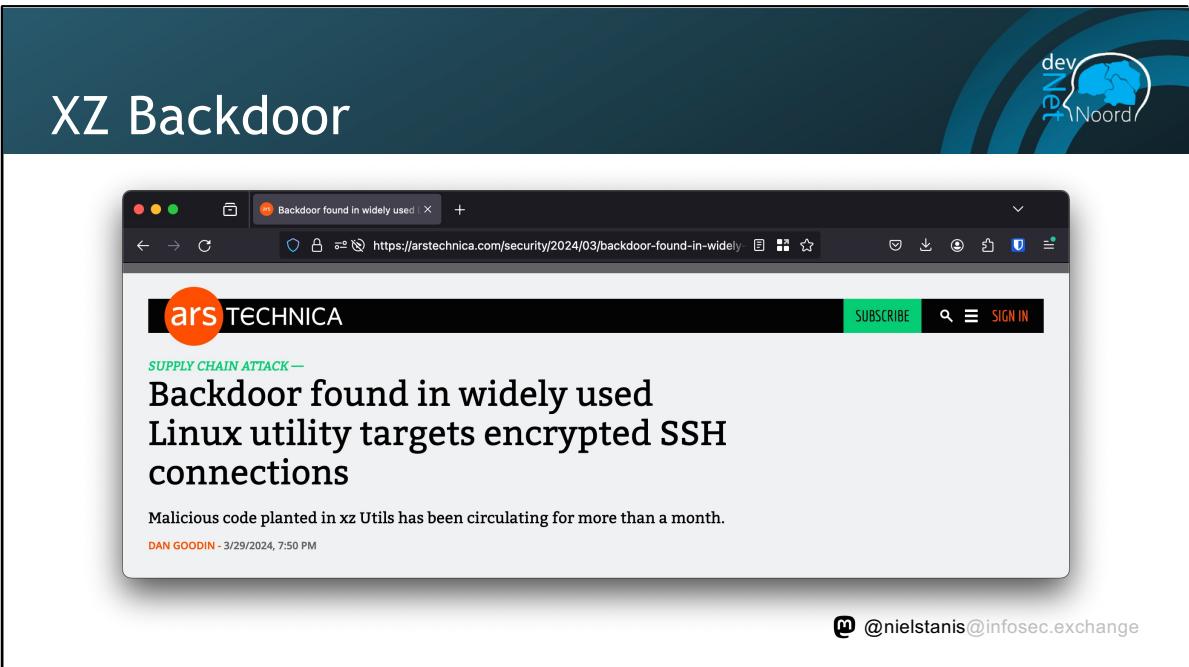
The threat actors also used typosquatting when creating their NuGet repository profiles to impersonate

@nielstanis@infosec.exchange

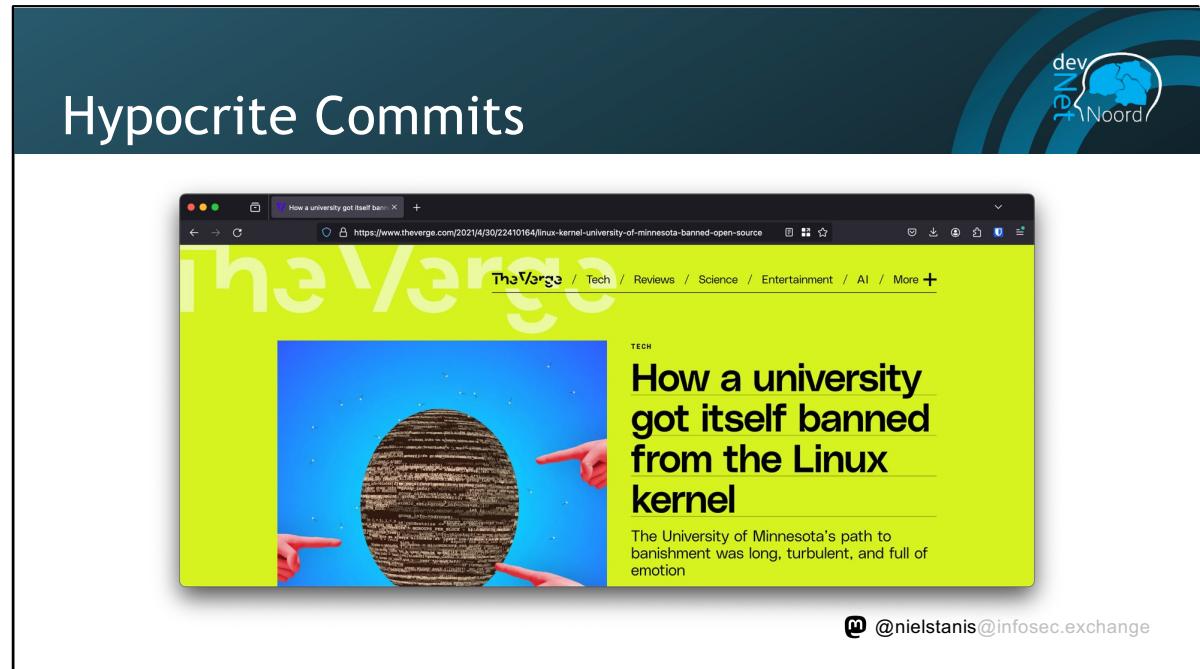
[https://www.bleepingcomputer.com/news/security/hackers-target-net-developers-with-malicious-nuget-packages//](https://www.bleepingcomputer.com/news/security/hackers-target-net-developers-with-malicious-nuget-packages/)



<https://www.reversinglabs.com/blog/iamreboot-malicious-nuget-packages-exploit-msbuild-loophole>

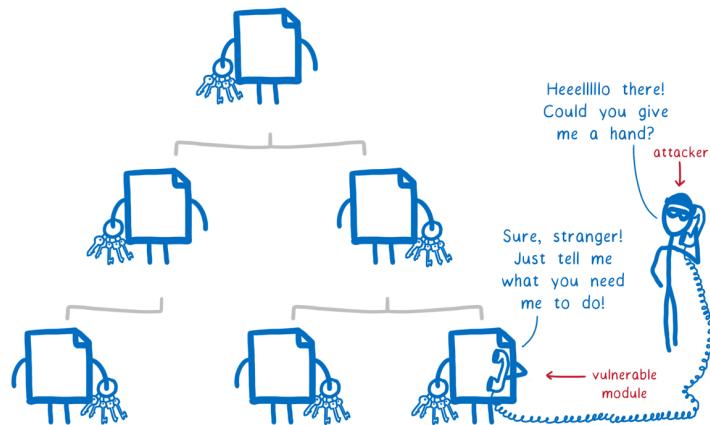


<https://arstechnica.com/security/2024/03/backdoor-found-in-widely-used-linux-utility-breaks-encrypted-ssh-connections/>



<https://www.theverge.com/2021/4/30/22410164/linux-kernel-university-of-minnesota-banned-open-source>

Vulnerable Assembly



@nielstanis@infosec.exchange

<https://hacks.mozilla.org/2019/11/announcing-the-bytecode-alliance/>

Vulnerabilities in Libraries

The screenshot shows a GitHub issue page for Microsoft Security Advisory CVE-2023-36558: .NET Security Feature Bypass Vulnerability. The page is titled "Microsoft Security Advisory CVE-2023-36558: .NET Security Feature Bypass Vulnerability" and is categorized under "Issues". The issue was opened by rbhanda on Nov 14, 2023, and has 0 comments. The issue body contains a summary of the vulnerability, stating that it exists in ASP.NET where an unauthenticated user can bypass validation on Blazor server forms. The discussion section links to dotnet/runtime#94726. The right sidebar shows project details: Assignees (None assigned), Labels (Security), Projects (None yet), Milestone (No milestone), Development (No branches or pull requests), and Notifications (Customize). The footer of the slide features the devNed logo and the text "@nielstanis@infosec.exchange".

<https://github.com/dotnet/announcements/issues/288>

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

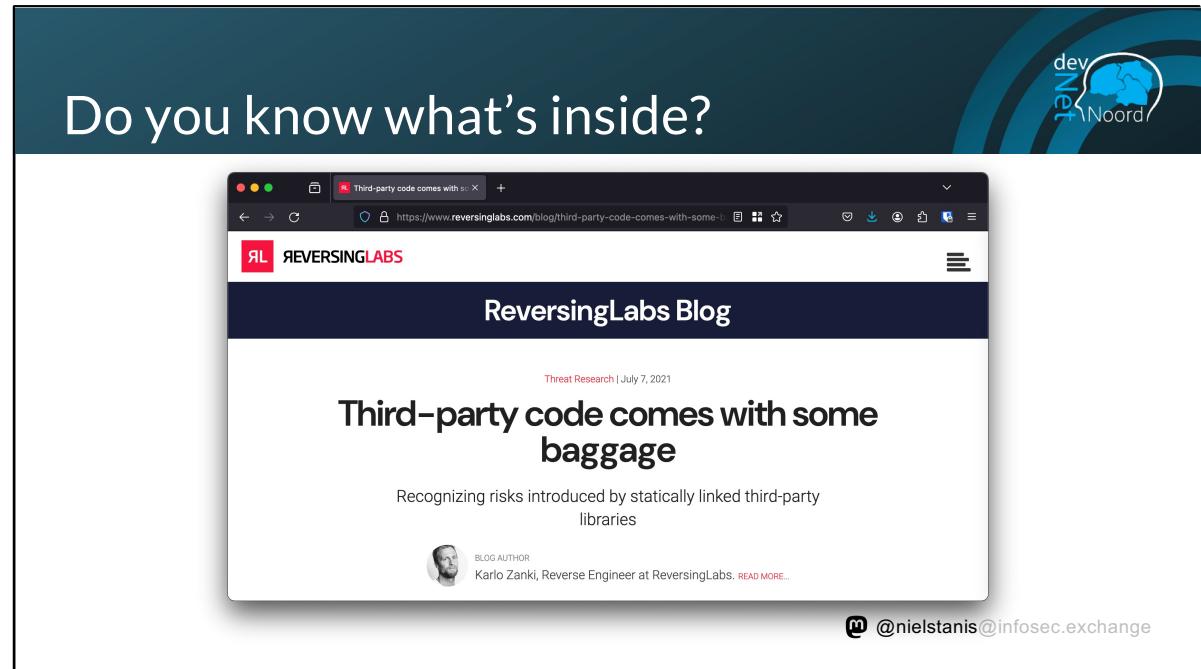


```
nelson@ghost-m2 ~/research/consoleapp $ dotnet list package --vulnerable --include-transitive
The following sources were used:
https://f.feedz.io/fennec/docgenerator/nuget/index.json
https://api.nuget.org/v3/index.json

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



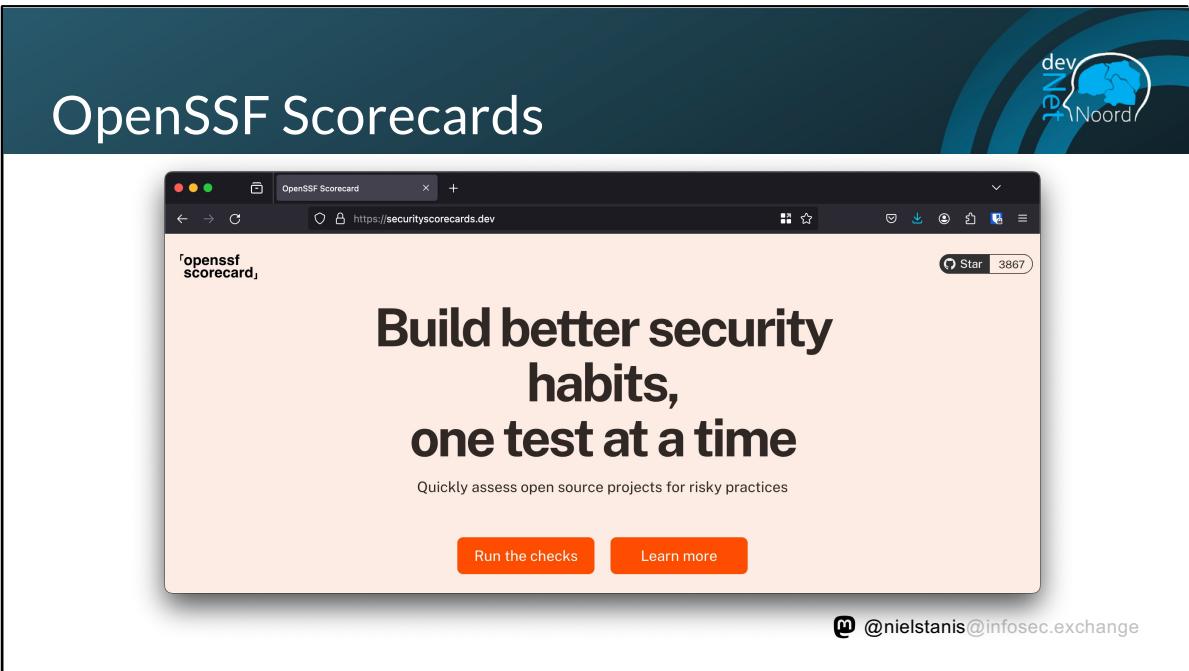
<https://www.reversinglabs.com/blog/third-party-code-comes-with-some-baggage>

Nutrition Label for Software?



@nielstanis@infosec.exchange

<https://securityscorecards.dev/>



<https://securityscorecards.dev/>

The screenshot shows a web browser window with the title 'OpenSSF Security Scorecards' at the top. The main content area displays the 'What is OpenSSF Scorecard?' page. On the left, there's a sidebar with sections for 'Run the checks' (Using the GitHub Action, Using the 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). The main content area has a heading 'What is OpenSSF Scorecard?' followed by a detailed description of what the scorecard does and its purpose. At the bottom right of the content area, there are two small icons: one of a person and another of a grid.

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:

 @nielstanis@infosec.exchange

<https://securityscorecards.dev/>

The screenshot shows a web browser window with the title 'OpenSSF Security Scorecard' and the URL 'https://securityscorecards.dev/#how-it-works'. The page has a dark blue header with the 'devNed Noord' logo. The main content area has a light orange background. It features a sidebar with links like 'Run the checks', 'Learn more', and 'How it works' (which is currently selected). The main content area is titled 'How it works' and contains text about scorecard checks, risk levels, and aggregate scores. At the bottom, there are three horizontal bars representing risk levels: 'CRITICAL RISK' (10), 'HIGH RISK' (7.5), and 'MEDIUM RISK' (5).

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.

CRITICAL RISK	10
HIGH RISK	7.5
MEDIUM RISK	5

@nielstanis@infosec.exchange

<https://securityscorecards.dev/>

OpenSSF Security Scorecards

A screenshot of a web browser displaying the OpenSSF Security Scorecard website at <https://securityscorecards.dev/#the-checks>. The page features a large diagram in the center consisting of five overlapping circles arranged in a pentagonal pattern, all contained within a larger orange circle. The circles are labeled: 'CODE VULNERABILITIES' (top), 'BUILD ASSESSMENT' (top-left), 'MAINTENANCE' (top-right), 'SOURCE RISK ASSESSMENT' (bottom-left), and 'CONTINUOUS TESTING' (bottom-right). In the center of the overlapping area, the text 'HOLISTIC SECURITY PRACTICES' is written in red capital letters. To the left of the diagram, there is a sidebar with 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
- The checks**
- Use cases
- About the project name
- Part of the OSS community
- Get involved

@nielstanis@infosec.exchange

<https://securityscorecards.dev/>

Code Vulnerabilities (High)



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

ID	Packages	Summary	Affected versions	Published	Fix
GHSA-hwcc-4cy8-cf3h	NuGet/Snowflake.Data	Snowflake Connector .NET does not properly check the Certificate Revocation List (CRL)	2.0.25 2.1.1 2.1.2 2.1.3 2.1.4	yesterday	Fix available
GHSA-6xmx-85x3-4cy2	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

@nielstanis@infosec.exchange

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.

@nielstanis@infosec.exchange

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.

 @nielstanis@infosec.exchange

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.

 @nielstanis@infosec.exchange

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

@nielstanis@infosec.exchange

<https://www.bestpractices.dev/en/criteria/0>

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).

 @nielstanis@infosec.exchange

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?

 @nielstanis@infosec.exchange

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!

 @nielstanis@infosec.exchange

Source Risk Assesement 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!

 @nielstanis@infosec.exchange

Source Risk Assesement 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!

 @nielstanis@infosec.exchange

Source Risk Assesement 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/>

@nielstanis@infosec.exchange

Source Risk Assesement 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)

 @nielstanis@infosec.exchange

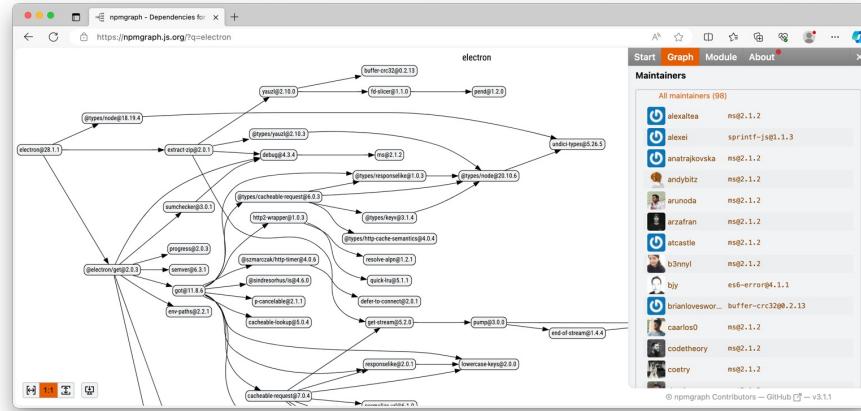
Source Risk Assessment Contributors (Low)



- 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?

 @nielstanis@infosec.exchange

Source Risk Assessment Contributors (Low)



 @nielstanis@infosec.exchange

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
- If Workflow is present what about the Actions used?

@nielstanis@infosec.exchange

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.

 @nielstanis@infosec.exchange

<https://securitylab.github.com/research/github-actions-preventing-pwn-requests/>

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.

@nielstanis@infosec.exchange

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



Running checks



 @nielstanis@infosec.exchange

<https://www.bestpractices.dev/en/criteria/0>

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**



 @nielstanis@infosec.exchange

<https://openssf.org/download-the-2023-openssf-annual-report/>

What can we improve?



 @nielstanis@infosec.exchange

<https://www.bestpractices.dev/en/criteria/0>

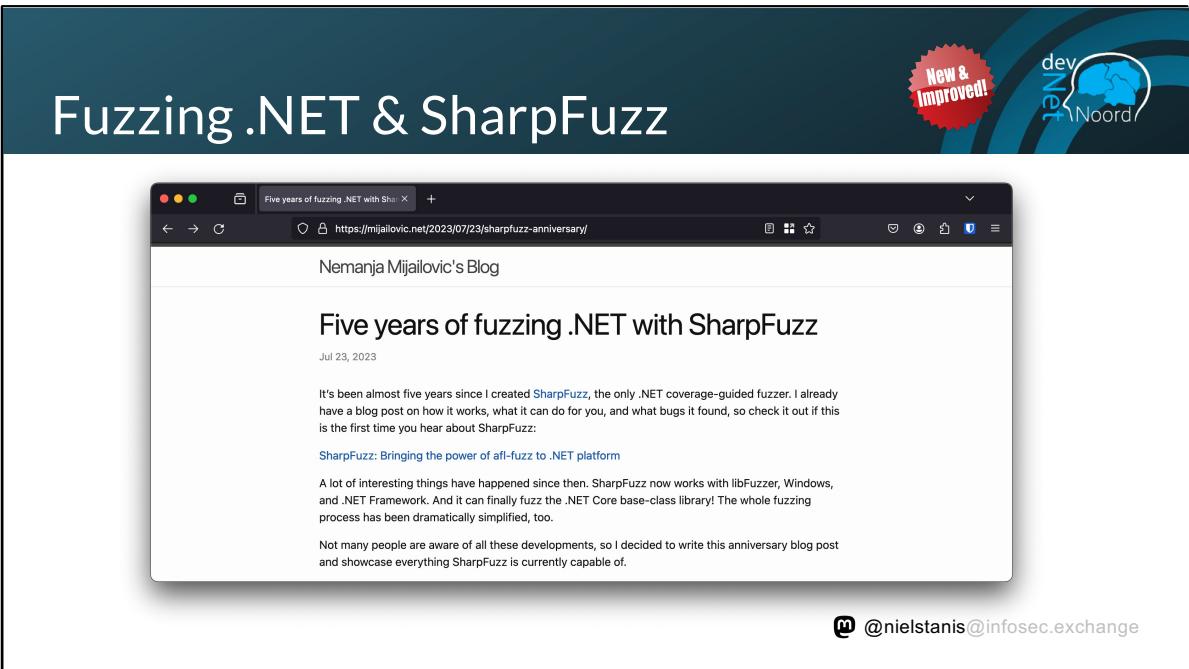
Fuzzing .NET



- 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?

 @nielstanis@infosec.exchange

<https://www.bestpractices.dev/en/criteria/0>



<https://mijailovic.net/2023/07/23/sharpfuzz-anniversary/>

The screenshot shows a blog post titled "Fuzzing .NET & SharpFuzz". In the top right corner, there is a red circular badge with the text "New & Improved!". Below the title, there is a banner for "devNet Noord" featuring a globe icon.

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.

@nielstanis@infosec.exchange

<https://mijailovic.net/2023/07/23/sharpfuzz-anniversary/>

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

<https://github.com/google/fuzzing/blob/master/docs/structure-aware-fuzzing.md>

Fuzzomatic: Using AI to Fuzz Rust

New & Improved!

devNord

```

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 | ...

@nielstanis@infosec.exchange

<https://research.kudelskisecurity.com/2023/12/07/introducing-fuzzomatic-using-ai-to-automatically-fuzz-rust-projects-from-scratch/>

Static Code Analysis (SAST)



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

@nielstanis@infosec.exchange

<https://www.bestpractices.dev/en/criteria/0>

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");
        }
    }
}
```

@nielstanis@infosec.exchange

<https://www.bestpractices.dev/en/criteria/0>

.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?

 @nielstanis@infosec.exchange

The screenshot shows the Microsoft Application Inspector application window. At the top, there's a banner with the text "Application Inspector" and a "New & Improved!" badge. On the right, there's a logo for "devNed Noord". The main content area is titled "Application Features" and contains a detailed description of what this section does. Below this, there are two main sections: "Feature Groups" and "Associated Rules".

Feature Groups:

- + Select Features
- + General Features
- + Development
- + Active Content
- + Data Storage
- + Sensitive Data
- + Cloud Services
- + OS Integration
- + OS System Changes
- + Other

Associated Rules:

- Name (click to view source)
 - Authentication: Microsoft (Identity)
 - Authentication: General
 - Authentication: (OAuth)

At the bottom right of the window, there's a footer with the Microsoft logo and the email address "@nielstanis@infosec.exchange".

<https://github.com/microsoft/ApplicationInspector>

The screenshot shows the Microsoft Application Inspector interface. At the top, there's a dark header with the title "Application Inspector". To the right of the title is a red circular badge with the text "New & Improved!". Below the header is a navigation bar with the text "Select Features". The main content area is a table with the following columns: "Feature", "Confidence", and "Details". The table lists six features:

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

At the bottom right of the interface, there's a footer with the text "@nielstanis@infosec.exchange" next to a small profile icon.

<https://github.com/microsoft/ApplicationInspector>

The screenshot shows a web browser displaying the [Cargo Vet](https://mozilla.github.io/cargo-vet/) documentation. The page has a dark theme with a sidebar on the left containing a table of contents. The main content area is titled "Cargo Vet" and discusses the tool's purpose and features, including sharing, relative audits, and deferred audits. A red circular badge in the top right corner says "New & Improved!". The Mozilla logo is visible in the bottom right corner of the page.

Community Review

New & Improved!

Cargo Vet

Introduction - Cargo Vet

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

4. Advanced Topics

Cargo Vet

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

<https://mozilla.github.io/cargo-vet/>

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 its own!
- NuGet Package Scoring (NET Score)
- Room for .NET specific improvements with Fennec CLI & contributions to OpenSSF Scorecard project



 @nielstanis@infosec.exchange

Questions?



 @nielstanis@infosec.exchange

Thanks!



- <https://github.com/nielstanis/devcampnoord2024/>
- ntanis at Veracode.com
- @nielstanis@infosec.exchange
- <https://www.fennec.dev> & <https://blog.fennec.dev>
- Bedankt! Thank you!

 @nielstanis@infosec.exchange