



# Reviewing NuGet Packages security easily using OpenSSF Scorecard

Niels Tanis  
Sr. Principal Security Researcher

NDC { Oslo }

## 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**



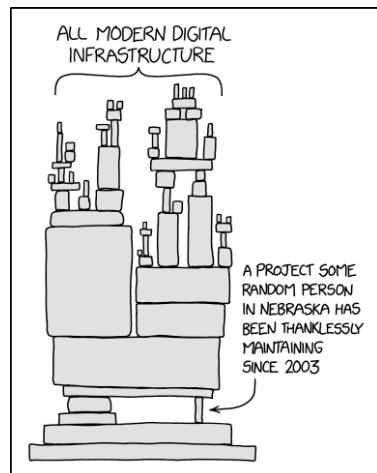
**MVP** Microsoft®  
Most Valuable  
Professional

NDC { Oslo }

 [@nielstanis@infosec.exchange](mailto:@nielstanis@infosec.exchange)

# Modern Application Architecture XKCD 2347

NDC { Oslo }



 @nielstanis@infosec.exchange

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

# Agenda

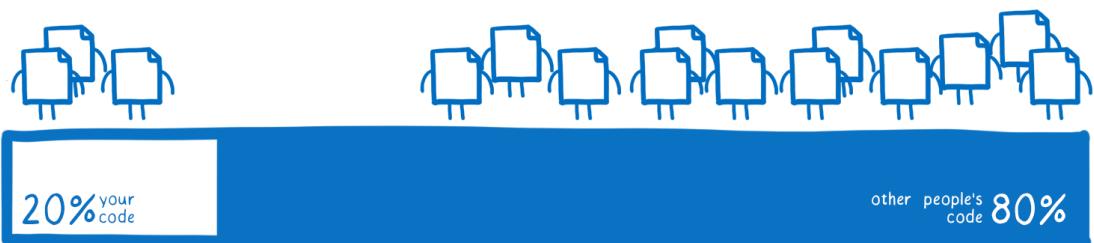
- Risks in 3<sup>rd</sup> party NuGet Packages
- OpenSFF Scorecard
- New & Improved
- Conclusion - Q&A



NDC { Oslo }

@nielstanis@infosec.exchange

## Average codebase composition



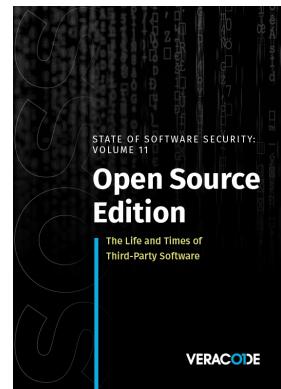
NDC { Oslo }

 @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."*



NDC { Oslo }

@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.

NDC { Oslo }

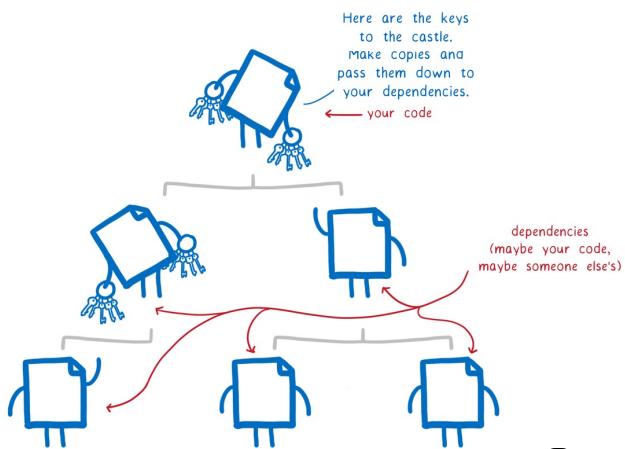
 [@nielstanis@infosec.exchange](mailto:@nielstanis@infosec.exchange)

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

# Average codebase composition

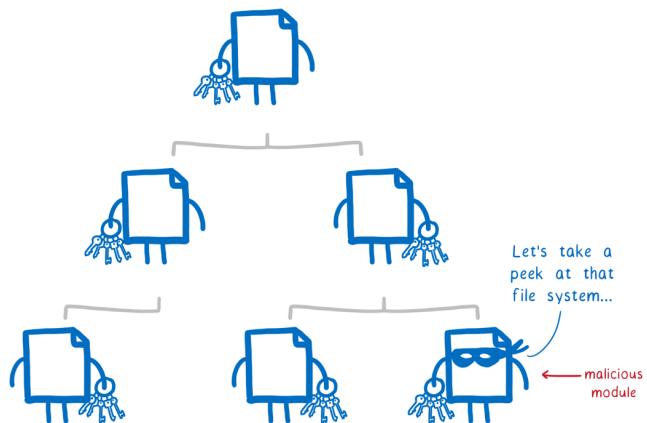
NDC { Oslo }

 @nielstanis@infosec.exchange



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

# Malicious Assembly



NDC { Oslo }

 @nielstanis@infosec.exchange

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

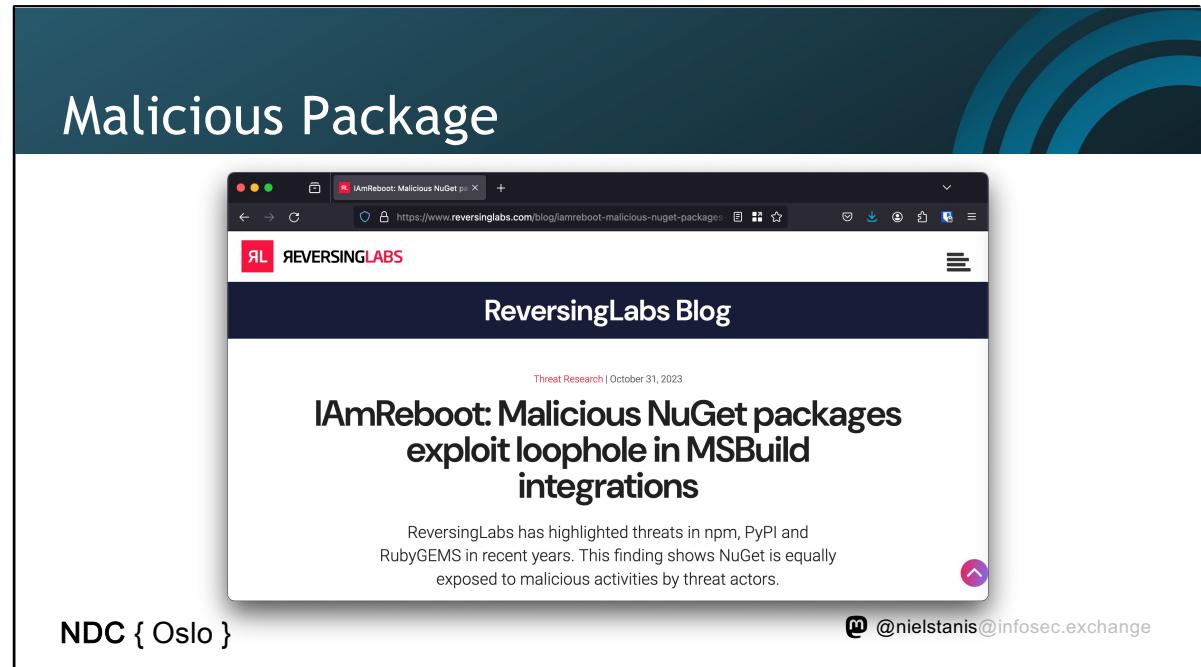
# Malicious Package

The screenshot shows a web browser window displaying an 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 by injecting cryptocurrency stealers through the NuGet repository and impersonating 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 campaign. The article 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, stating that the top three packages were downloaded an incredible amount of times, which could indicate success. The article also mentions that attackers inflated download counts using bots to make the packages seem more legitimate. It concludes that threat actors used typosquatting when creating their NuGet repository profiles to impersonate legitimate ones.

NDC { Oslo }

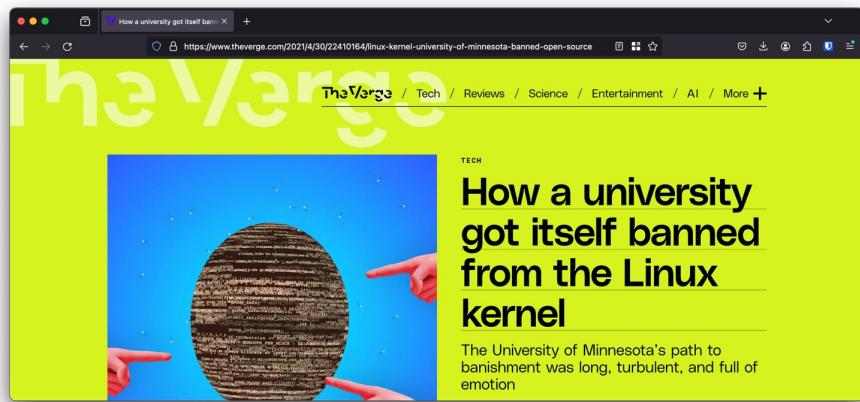
 @nielstanis@infosec.exchange

[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>

## Hypocrite Commits



NDC { Oslo }

@nielstanis@infosec.exchange

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

# XZ Backdoor

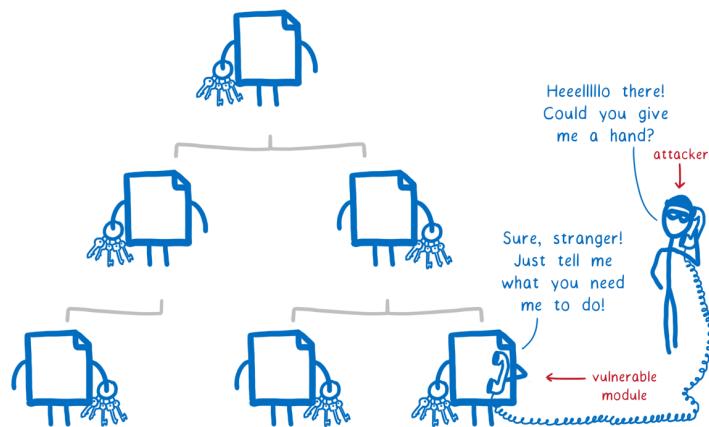
The screenshot shows a web browser window displaying an Ars Technica article. The title of the article is "Backdoor found in widely used Linux utility targets encrypted SSH connections". The article is categorized under "SUPPLY CHAIN ATTACK". A sub-headline states: "Malicious code planted in xz Utils has been circulating for more than a month." The author is listed as "DAN GOODIN - 3/29/2024, 7:50 PM". The Ars Technica logo is visible at the top left, and there are "SUBSCRIBE", "SIGN IN", and search icons at the top right.

NDC { Oslo }

@nielstanis@infosec.exchange

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

# Vulnerable Assembly



NDC { Oslo }

@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 title is "Microsoft Security Advisory CVE-2023-36558: .NET Security Feature Bypass Vulnerability #288". The issue was opened by rbhanda on Nov 14 and has 0 comments. The advisory summary states: "Microsoft is releasing this security advisory to provide information about a vulnerability in ASP.NET Core 6.0, ASP.NET Core 7.0 and, ASP.NET Core 8.0 RC2. This advisory also provides guidance on what developers can do to update their applications to address this vulnerability. A security feature bypass vulnerability exists in ASP.NET where an unauthenticated user is able to bypass validation on Blazor server forms which could trigger unintended actions." The discussion section links to dotnet/runtime#94726. The sidebar on the right shows no assignees, labels (Security), projects (None yet), milestones (No milestone), development (No branches or pull requests), and notifications (Customize).

NDC { Oslo }

@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 $
```

NDC { Oslo }

 @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
```

NDC { Oslo }

 @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 $
```

NDC { Oslo }

 @nielstanis@infosec.exchange

Do you know what's inside?

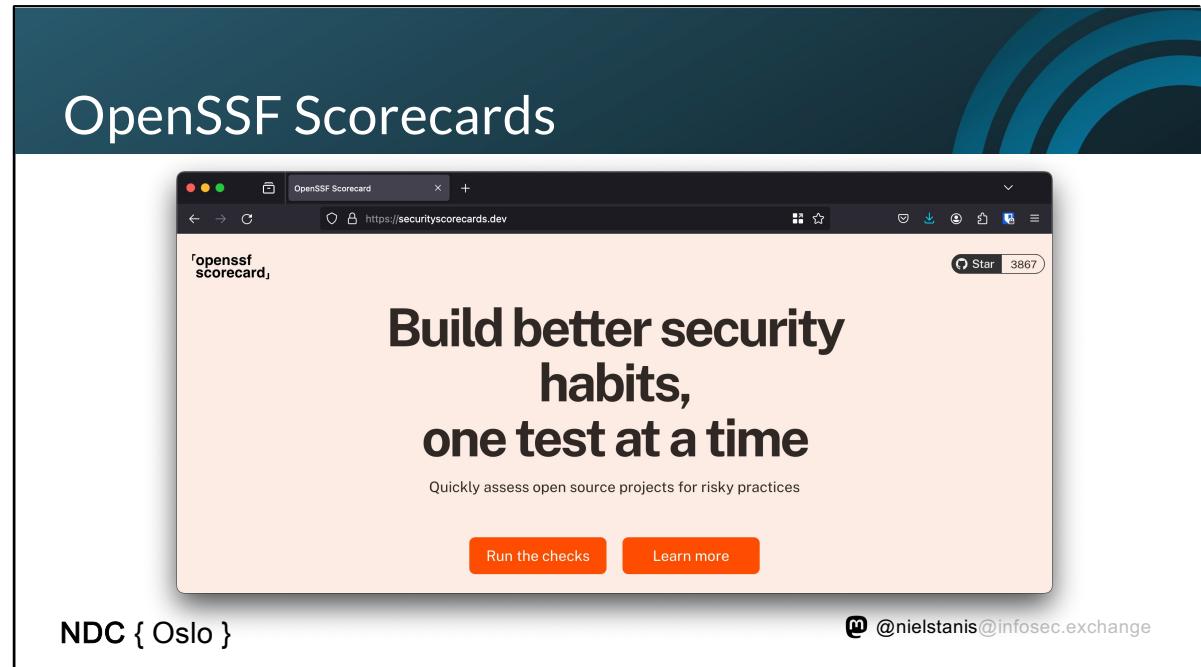
The screenshot shows a web browser window displaying a blog post. The header of the browser says "Third-party code comes with some baggage". The page itself has a dark blue header with the "REVERSINGLABS" logo and a menu icon. The main content area has a white background. At the top of the content area, there is a small red banner with the text "Threat Research | July 7, 2021". Below this, the title of the blog post is displayed in large, bold, black font: "Third-party code comes with some baggage". Underneath the title, there is a subtitle in smaller black font: "Recognizing risks introduced by statically linked third-party libraries". Below the subtitle, there is a small profile picture of a man, followed by the text "BLOG AUTHOR" and "Karlo Zanki, Reverse Engineer at ReversingLabs. [READ MORE...](#)". At the bottom of the content area, there is a footer with the text "NDC { Oslo }" on the left and "@nielstanis@infosec.exchange" on the right, accompanied by a Twitter icon.

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

## Nutrition Label for Software?



<https://securityscorecards.dev/>



<https://securityscorecards.dev/>

The screenshot shows a web browser window with the title "OpenSSF Security Scorecards". The main content area displays the "What is OpenSSF Scorecard?" page. On the left, there is 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. Below the text are two small icons: a red circle with a white dot and a grid of squares.

NDC { Oslo }

@nielstanis@infosec.exchange

<https://securityscorecards.dev/>

The screenshot shows a web browser window with the title "OpenSSF Security Scorecards" at the top. The main content area is titled "How it works". On the left, there's a sidebar with sections like "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), and a "Scorecard" section. The main content area describes the tool's functionality, mentioning automated checks for vulnerabilities across the software supply chain, a score out of 10, risk levels (CRITICAL RISK, HIGH RISK, MEDIUM RISK), and aggregate scores. At the bottom right, there's a footer with the text "NDC { Oslo }" and a Twitter handle "@nielstanis@infosec.exchange".

<https://securityscorecards.dev/>

# OpenSSF Security Scorecards



NDC { Oslo }

[@nielstanis@infosec.exchange](mailto:@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-32fb-hm:3-7vxp | NuGet/Microsoft.Azure.Storage.DataMovement   | Azure Storage Movement Client Library Denial of Service Vulnerability                              | 0.1.0<br>0.10.1<br>0.11.0<br>0.12.0<br>0.2.0<br>0.3.0<br>—    | yesterday   | <a href="#">Fix available</a> |
| GHSA-m5cv-6rdh-3yj9 | PIP/Azure.Identity<br>npm/@azure/identity<br>Maven/com.azure.azure-identity<br>npm/@azure/msal-node<br>NuGet/Microsoft.Identity.Client<br>GojiProject/Azure/azure-sdk-for-go/sdk/azidentity<br>Maven/com.microsoft.azure/msal4<br>NuGet/Azure.Identity | Azure Identity Libraries and Microsoft Authentication Library Elevation of Privilege Vulnerability | 1.0.0<br>1.0.0b2<br>1.0.0b3<br>1.0.0b4<br>1.0.1<br>1.1.0<br>— | yesterday   | <a href="#">Fix available</a> |
| GHSA-rq9-xiam-wrfw  | NuGet/Umbraco.Commerce   | Umbraco Commerce vulnerable to Stored Cross-site Scripting on Print Functionality                  | 12.0.0<br>12.1.0<br>12.1.0-re1<br>12.1.1                      | 2 weeks ago | <a href="#">Fix available</a> |

NDC { Oslo }

 @nielstanis@infosec.exchange

<https://osv.dev/list?ecosystem=NuGet>

## 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.

NDC { Oslo }

 [@nielstanis@infosec.exchange](mailto:@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.

NDC { Oslo }

 [@nielstanis@infosec.exchange](mailto:@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.

NDC { Oslo }

 [@nielstanis@infosec.exchange](mailto:@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

NDC { Oslo }

 @nielstanis@infosec.exchange

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

## 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).

NDC { Oslo }

 [@nielstanis@infosec.exchange](mailto:@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;
- Does it make sense to do fuzzing on .NET projects?

NDC { Oslo }

 [@nielstanis@infosec.exchange](mailto:@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!

NDC { Oslo }

 [@nielstanis@infosec.exchange](mailto:@nielstanis@infosec.exchange)

## 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!**

NDC { Oslo }

 [@nielstanis@infosec.exchange](mailto:@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!

NDC { Oslo }

 [@nielstanis@infosec.exchange](mailto:@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/>

NDC { Oslo }

 [@nielstanis@infosec.exchange](mailto:@nielstanis@infosec.exchange)

## Source Risk Assesement 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)

NDC { Oslo }

 [@nielstanis@infosec.exchange](mailto:@nielstanis@infosec.exchange)

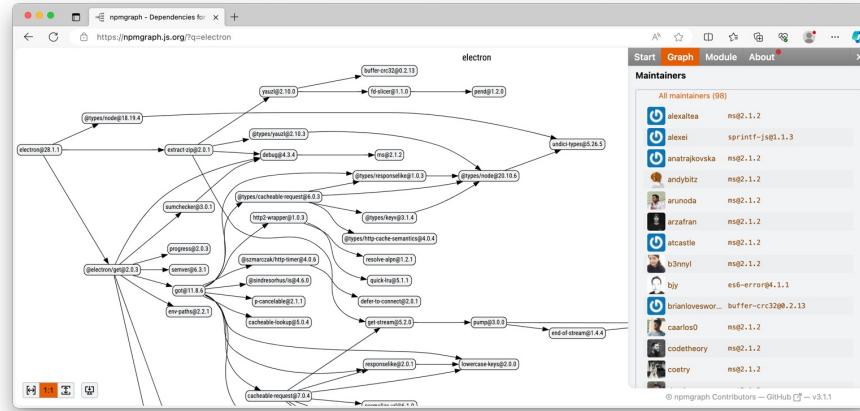
## Source Risk Assessment Contributors (Low)

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

NDC { Oslo }

 @nielstanis@infosec.exchange

# Source Risk Assessment Contributors (Low)



NDC { Oslo }

@nielstanis@infosec.exchange

## Build Risk Assessment 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?

NDC { Oslo }

 [@nielstanis@infosec.exchange](mailto:@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.

NDC { Oslo }

 @nielstanis@infosec.exchange

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

## 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.

NDC { Oslo }

 [@nielstanis@infosec.exchange](mailto:@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

NDC { Oslo }

 [@nielstanis@infosec.exchange](mailto:@nielstanis@infosec.exchange)

# Demo OpenSSF Scorecard Fennec CLI

Running checks



NDC { Oslo }

[@nielstanis@infosec.exchange](mailto:@nielstanis@infosec.exchange)

# 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**



NDC { Oslo }

[@nielstanis@infosec.exchange](mailto:@nielstanis@infosec.exchange)

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

## What can we improve?



NDC { Oslo }

 @nielstanis@infosec.exchange

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

## 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?

NDC { Oslo }

 [@nielstanis@infosec.exchange](mailto:@nielstanis@infosec.exchange)

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

**Fuzzing .NET & SharpFuzz**

New & Improved!

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.

NDC { Oslo }

@nielstanis@infosec.exchange

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

**Fuzzing .NET & SharpFuzz**

New & Improved!

**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.

NDC { Oslo }

@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) { }
    });
}
```

NDC { Oslo }

 @nielstanis@infosec.exchange

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

# Fuzzomatic: Using AI to Fuzz Rust

New & Improved!

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  extern crate libfuzzer_sys;
3  use mylib_under_test::MyModule;
4  use libfuzzer_sys::fuzz_target;
5
6  fuzz_target!(data: [u8]) {
7      // Fuzzed code goes here
8      if libfuzzer_sys::is_black_box() {
9          let mut input = std::str::from_utf8(data).unwrap();
10         MyModule::target_function(input);
11     }
12 }
```

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.

NDC { Oslo }

@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);
}
```

NDC { Oslo }

 @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");
        }
    }
}
```

NDC { Oslo }

@nielstanis@infosec.exchange

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

## .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)

NDC { Oslo }

 [@nielstanis@infosec.exchange](mailto:@nielstanis@infosec.exchange)

**New & Improved!**

**Application Inspector**

**Application Features**

This section reports the major characteristics of the application or its primary features organized by customizable Feature Groups. Click any of the highlighted icons below (indicating at least one match) to view additional details or expand a feature group for more information. To view where in source code a specific feature was found, click the Rule name link shown on the right. A disabled icon indicates a not found status for that feature.

| Feature Groups      | Associated Rules                     |
|---------------------|--------------------------------------|
| + Select Features   | Name (click to view source)          |
| + General Features  | Authentication: Microsoft (Identity) |
| + Development       | Authentication: General              |
| + Active Content    | Authentication: (OAuth)              |
| + Data Storage      |                                      |
| + Sensitive Data    |                                      |
| + Cloud Services    |                                      |
| + OS Integration    |                                      |
| + OS System Changes |                                      |
| + Other             |                                      |

NDC { Oslo }

@nielstanis@infosec.exchange

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

# Application Inspector



| Feature                   | Confidence | Details              |
|---------------------------|------------|----------------------|
| Authentication            |            | <a href="#">View</a> |
| Authorization             |            | <a href="#">View</a> |
| Cryptography              |            | <a href="#">View</a> |
| Object Deserialization    |            | N/A                  |
| AV Media Parsing          |            | N/A                  |
| Dynamic Command Execution |            | N/A                  |

NDC { Oslo }

@nielstanis@infosec.exchange

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

# 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

NDC { Oslo }

@nielstanis@infosec.exchange

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

## 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!



NDC { Oslo }

 @nielstanis@infosec.exchange

## Conclusion

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



NDC { Oslo }

 [@nielstanis@infosec.exchange](mailto:@nielstanis@infosec.exchange)

## Questions?



NDC { Oslo }

 [@nielstanis@infosec.exchange](mailto:@nielstanis@infosec.exchange)

## Links

- <https://github.com/nielstanis/ndcoslo2024/>
- ntanis at Veracode.com
- @nielstanis@infosec.exchange
- <https://www.fennec.dev> & <https://blog.fennec.dev>

NDC { Oslo }

 [nielstanis@infosec.exchange](mailto:nielstanis@infosec.exchange)