



Reviewing NuGet Packages security easily using OpenSSF Scorecard

Niels Tanis
Sr. Principal Security Researcher

NDC { Security }

Who am I?

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

VERACODE



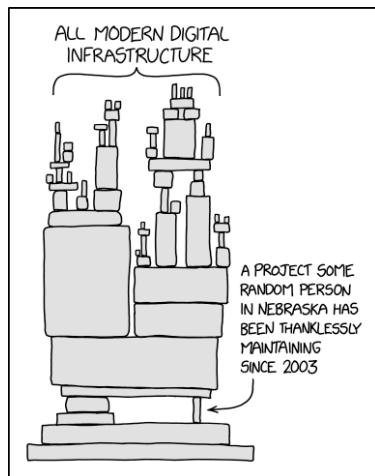
MVP Microsoft®
Most Valuable
Professional

NDC { Security }

 @nielstanis@infosec.exchange

Modern Application Architecture XKCD 2347

NDC { Security }



 @nielstanis@infosec.exchange

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

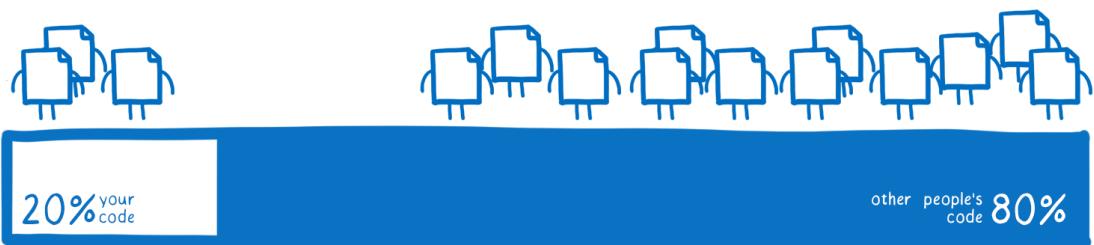
Agenda

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

NDC { Security }

 @nielstanis@infosec.exchange

Average codebase composition



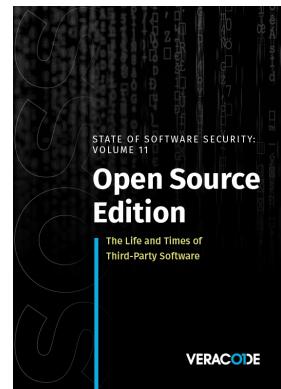
NDC { Security }

 @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 { Security }

@nielstanis@infosec.exchange

State of Log4j - 2 years later

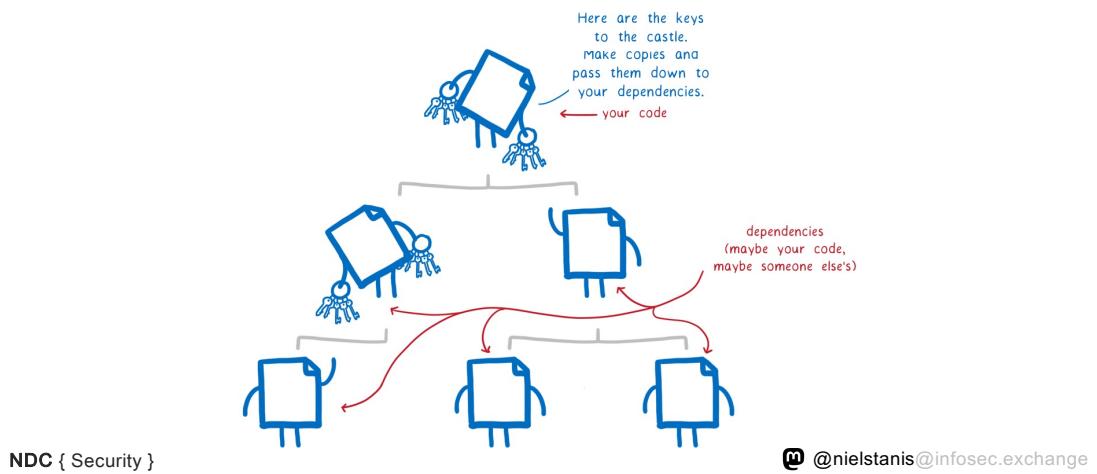
- Analysed our data August-November 2023
 - Total set of almost 39K unique applications scanned
- 2.8% run version vulnerable to Log4Shell
- 3.8% run version patched but vulnerable to other CVE
- 32% rely on a version that's end-of-life and have no support for any patches.

NDC { Security }

 @nielstanis@infosec.exchange

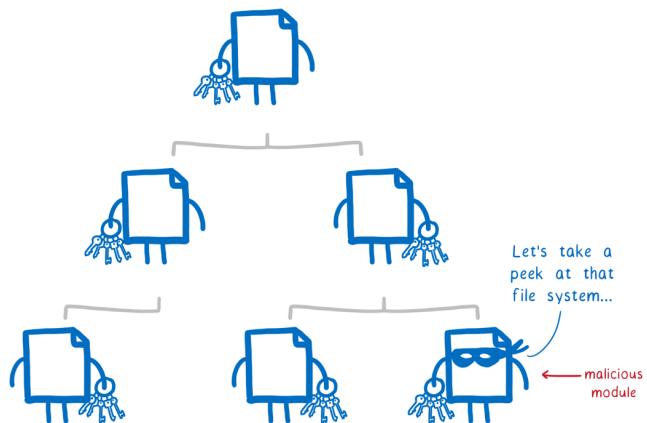
<https://www.veracode.com/blog/research/state-log4j-vulnerabilities-how-much-did-log4shell-change>

Average codebase composition



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

Malicious Assembly

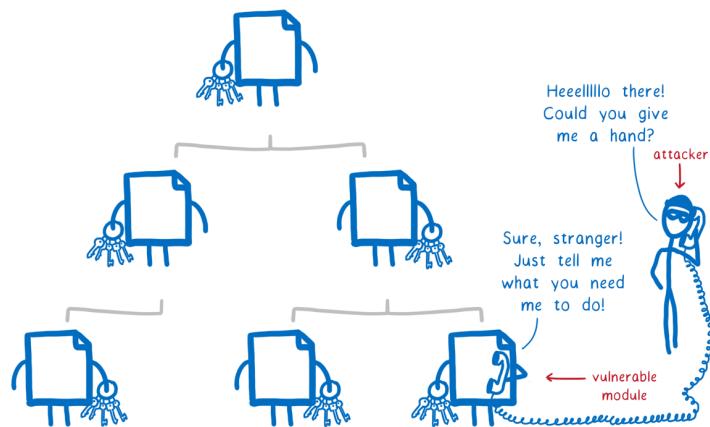


NDC { Security }

 @nielstanis@infosec.exchange

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

Vulnerable Assembly



NDC { Security }

@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 { Security }

@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 { Security }

 @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 { Security }

 @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 { Security }

 @nielstanis@infosec.exchange

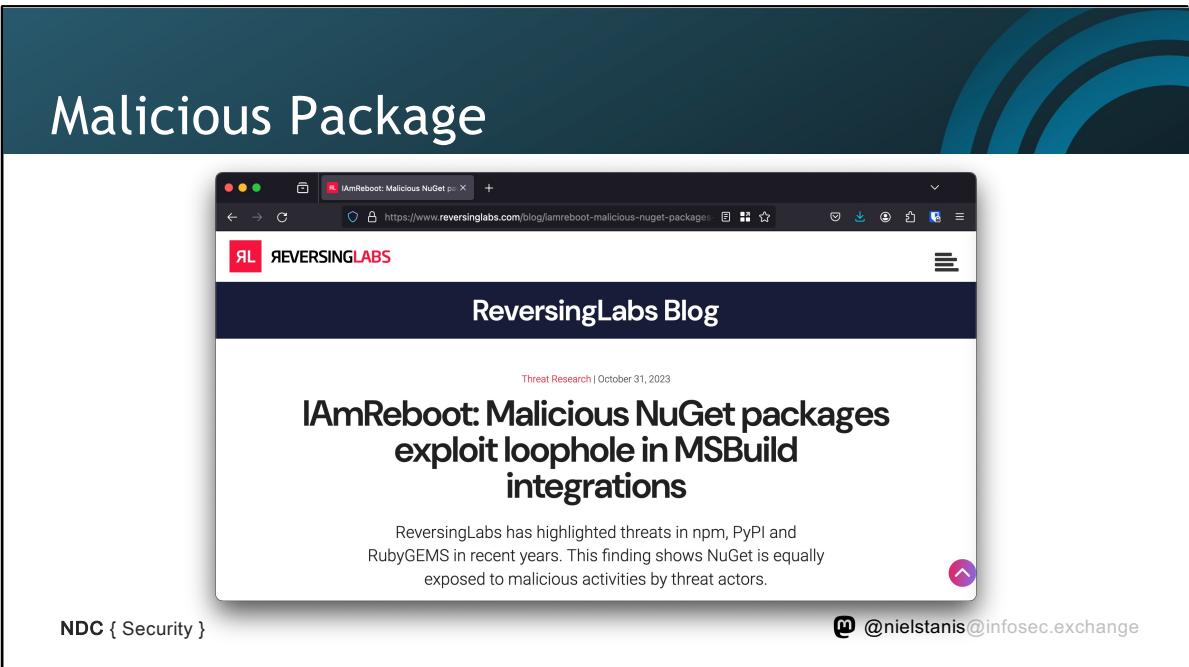
Malicious Package

The screenshot shows a news article titled "Hackers target .NET developers with malicious NuGet packages" published by Sergiu Gatlan on March 20, 2023, at 03:22 PM. 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 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. Quoted researchers say the top three packages were downloaded an incredible amount of times, which could indicate success. The threat actors used typosquatting when creating their NuGet repository profiles to impersonate legitimate ones.

NDC { Security }

@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>

Do you know what's inside?

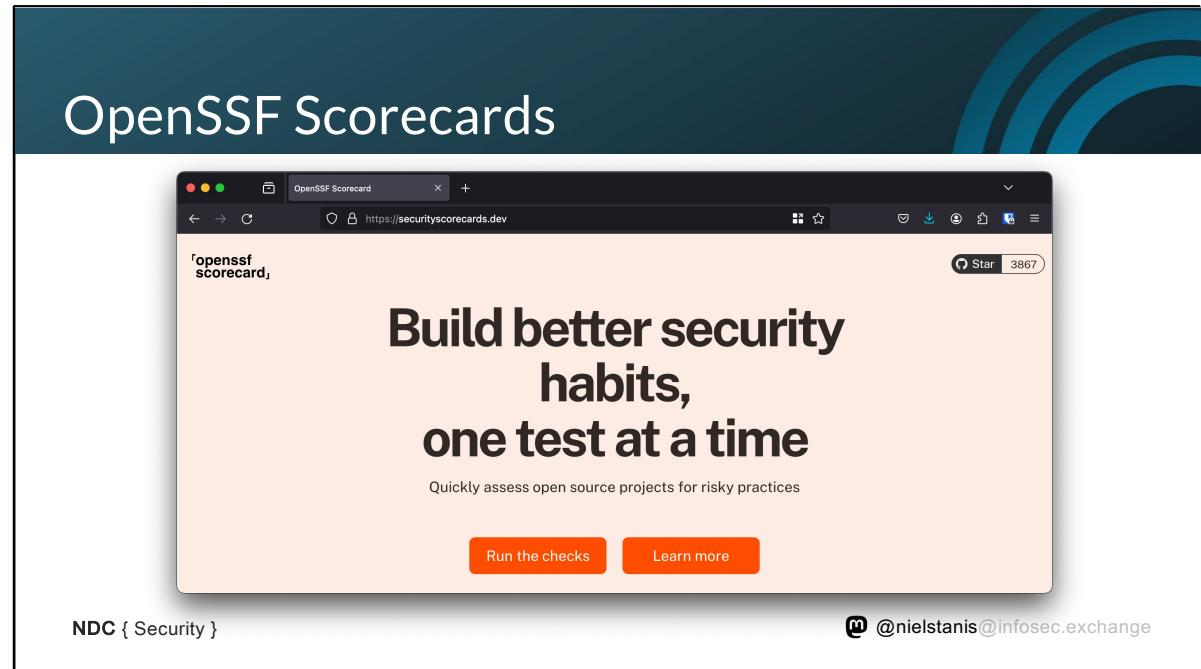
The screenshot shows a web browser window displaying a blog post from ReversingLabs. The title of the post is "Third-party code comes with some baggage". The author is Karlo Zanki, Reverse Engineer at ReversingLabs. The URL of the page is <https://www.reversinglabs.com/blog/third-party-code-comes-with-some-baggage>.

<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 three columns: "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.", and "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." Below the content are two small icons: a red circle with a white dot and a grid of squares.

NDC { Security }

@nielstanis@infosec.exchange

<https://securityscorecards.dev/>

OpenSSF Security Scorecards

The screenshot shows a web browser window for the 'OpenSSF Scorecard' at <https://securityscorecards.dev/#how-it-works>. The page has a dark blue header with the title 'OpenSSF Security Scorecards'. Below the header is a light orange section titled 'How it works'. On the left, there are two columns of links: '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 right side contains explanatory text and three horizontal bars representing risk levels: 'CRITICAL RISK' (10), 'HIGH RISK' (7.5), and 'MEDIUM RISK' (5). At the bottom left is the text 'NDC { Security }' and at the bottom right is a Twitter handle '@nielstanis@infosec.exchange'.

<https://securityscorecards.dev/>

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 large graphic 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 these circles is the text 'HOLISTIC SECURITY PRACTICES'. To the left of the graphic is a sidebar with two sections: 'Run the checks' and 'Learn more'. The 'Run the checks' section includes links for 'Using the GitHub Action' and 'Using the CLI'. The 'Learn more' section includes links for 'The problem', 'What is OpenSSF Scorecard?', 'How it works', 'The checks' (which is highlighted in red), 'Use cases', 'About the project name', 'Part of the OSS community', and 'Get involved'.

NDC { Security }

@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	yesterday	<button>Fix available</button>
GHSAA-6xmx-85x3-4cy2	NuGet/Umbraco.CMS	Stored XSS via SVG File Upload	10.0.0 10.0.0-rc2 10.0.0-rc3	last week	<button>Fix available</button>

NDC { Security }

 @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.

NDC { Security }

 @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 { Security }

 @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 { Security }

 @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 { Security }

@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).

NDC { Security }

 @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?

NDC { Security }

 @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 { Security }

 @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 { Security }

 @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 { Security }

 @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 { Security }

 @nielstanis@infosec.exchange

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

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

NDC { Security }

 @nielstanis@infosec.exchange

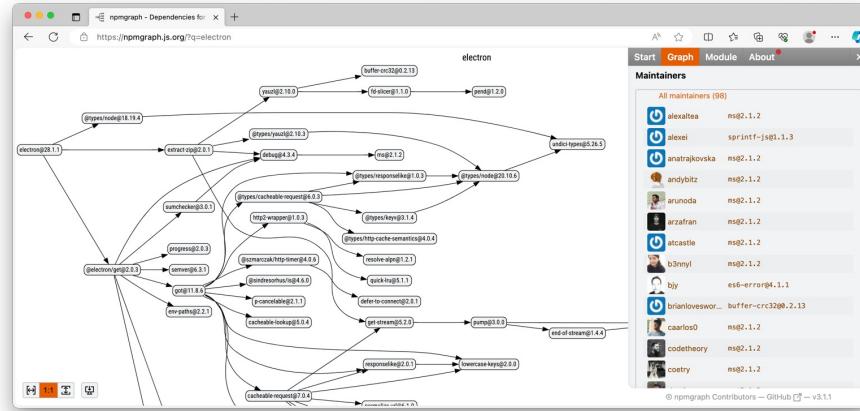
Source Risk Assessment Contributors (Low)

- 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?

NDC { Security }

 @nielstanis@infosec.exchange

Source Risk Assessment Contributors (Low)



NDC { Security }

@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

NDC { Security }

 @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 { Security }

 @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.

NDC { Security }

 @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 { Security }

 @nielstanis@infosec.exchange

Demo OpenSSF Scorecard

Running checks



NDC { Security }

 @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**



NDC { Security }

@nielstanis@infosec.exchange

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

What can we improve?



NDC { Security }

 @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?

NDC { Security }

 @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 { Security }

@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 { Security }

@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 { Security }
```

@nielstanis@infosec.exchange

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

Fuzzomatic: Using AI to Fuzz Rust

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.

NDC { Security }

@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 { Security }

 @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 { Security }

 @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?

NDC { Security }

 @nielstanis@infosec.exchange

Application Inspector

New & Improved!

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 { Security }

@nielstanis@infosec.exchange

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

Application Inspector

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

NDC { Security } @nielstanis@infosec.exchange

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

Community Review



The screenshot shows a dark-themed web browser window displaying the [Cargo Vet](https://mozilla.github.io/cargo-vet/) documentation. The left sidebar contains a table of contents with sections like 1. Introduction, 2. Tutorial, and 3. Reference. The main content area is titled "Cargo Vet" and discusses the tool's purpose of auditing third-party Rust dependencies. It highlights features such as sharing findings between projects, inspecting relative audit differences, and adding deferred audits. A red starburst badge in the top right corner of the content area says "New & Improved!".

NDC { Security }

 @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 it's own!
- Room for .NET specific improvements with Fennec CLI & contributions to OpenSSF Scorecard project

NDC { Security }

@nielstanis@infosec.exchange

Questions?

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

NDC { Security }

 @nielstanis@infosec.exchange