

# **Beyond Trust: Building Community-Driven Security Analysis for Your .NET Software Supply Chain**

**Niels Tanis**

**Sr. Principal Security Researcher**

**VERACODE NDC { Manchester }**

# Who am I?

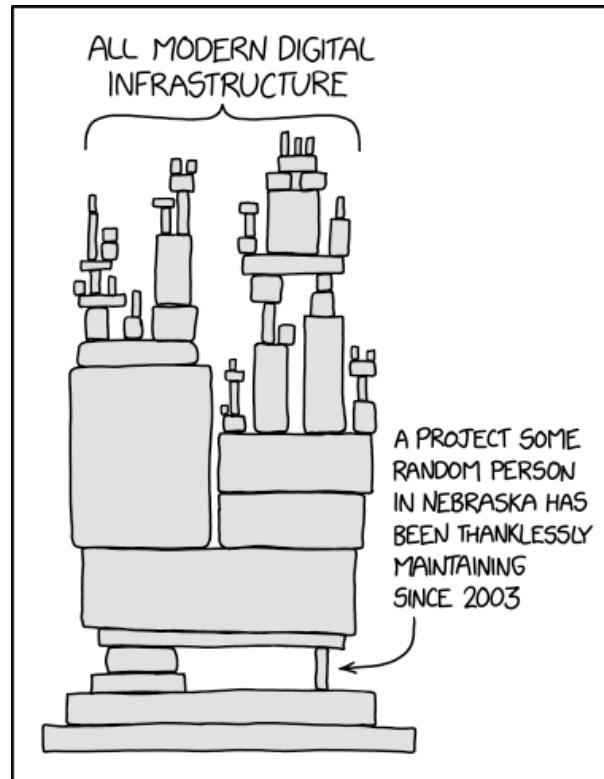
- Niels Tanis
- Sr. Principal Security Researcher
  - Background .NET Development,  
Pentesting/ethical hacking,  
and software security consultancy
  - Research on static analysis for .NET apps
  - Enjoying Rust!
- Microsoft MVP – Developer Technologies

VERACODE



# Modern Application Architecture

## XKCD 2347



NDC { Manchester }

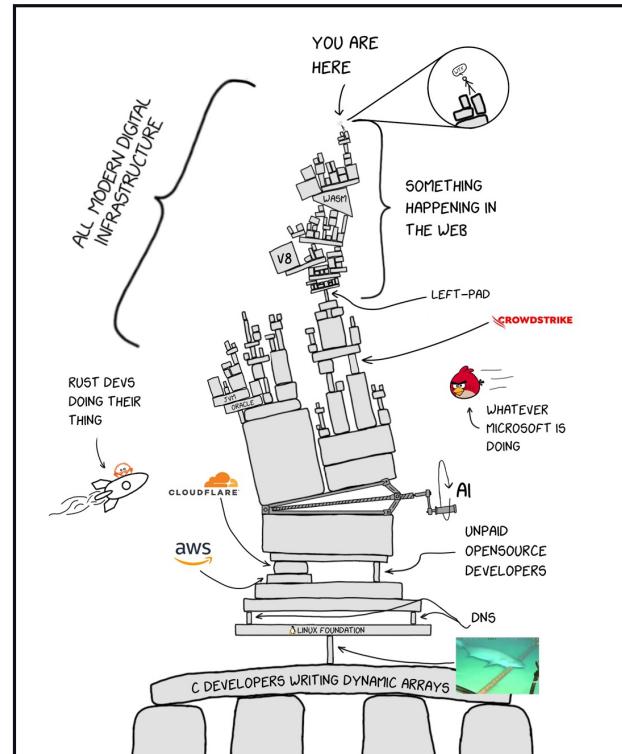


@niels.fennec.dev



@nielstanis@infosec.exchange

# You are here



NDC { Manchester }



@niels.fennec.dev



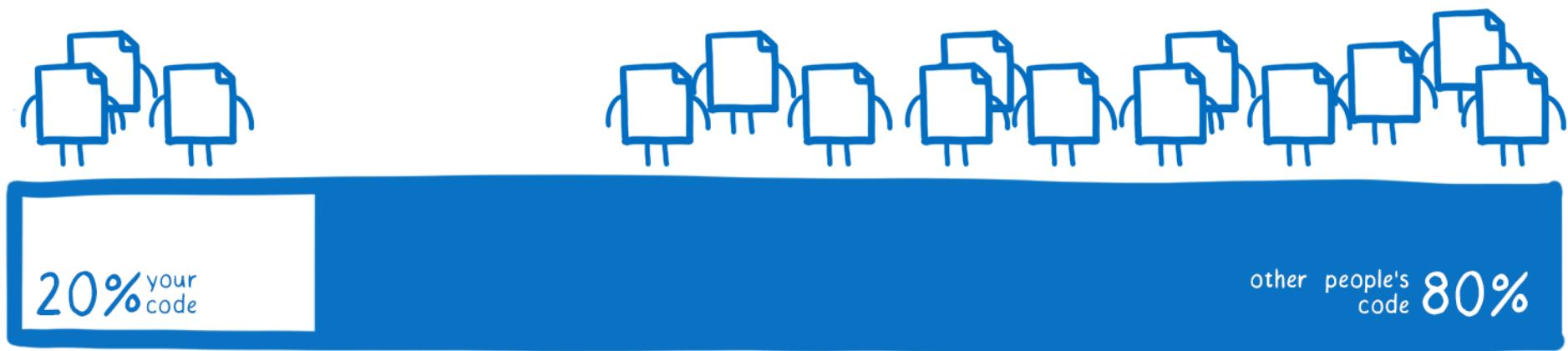
@nielstanis@infosec.exchange

# Agenda

- Security challenges in our .NET software supply chain
- How can we start securing our supply-chain?
- Fennec Labs
  - Understanding what projects do for security
  - Know what's inside package
  - Review package
  - Feedback & sharing
  - Future and idea's
- Conclusion and Q&A



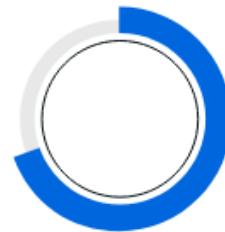
# Average codebase composition



# State Of Software Security 2025



**64%**  
of applications have  
flaws in **first-party code**



**70%**  
of applications have  
flaws in **third-party code**



NDC { Manchester }

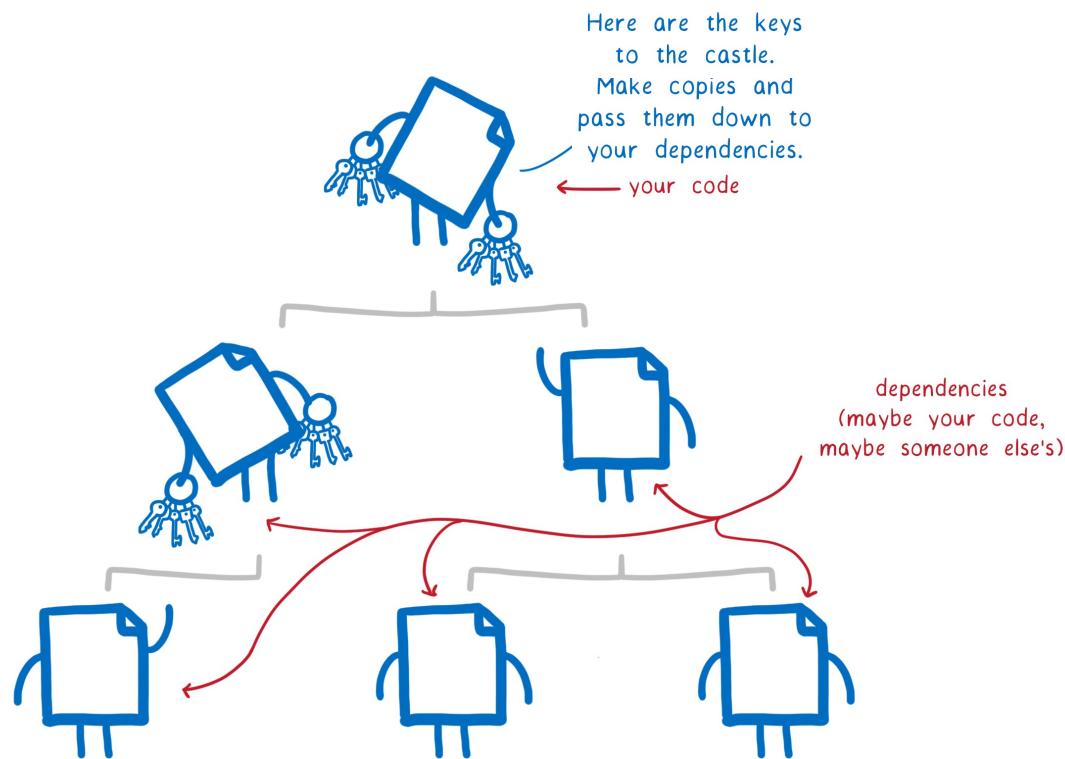


@niels.fennec.dev

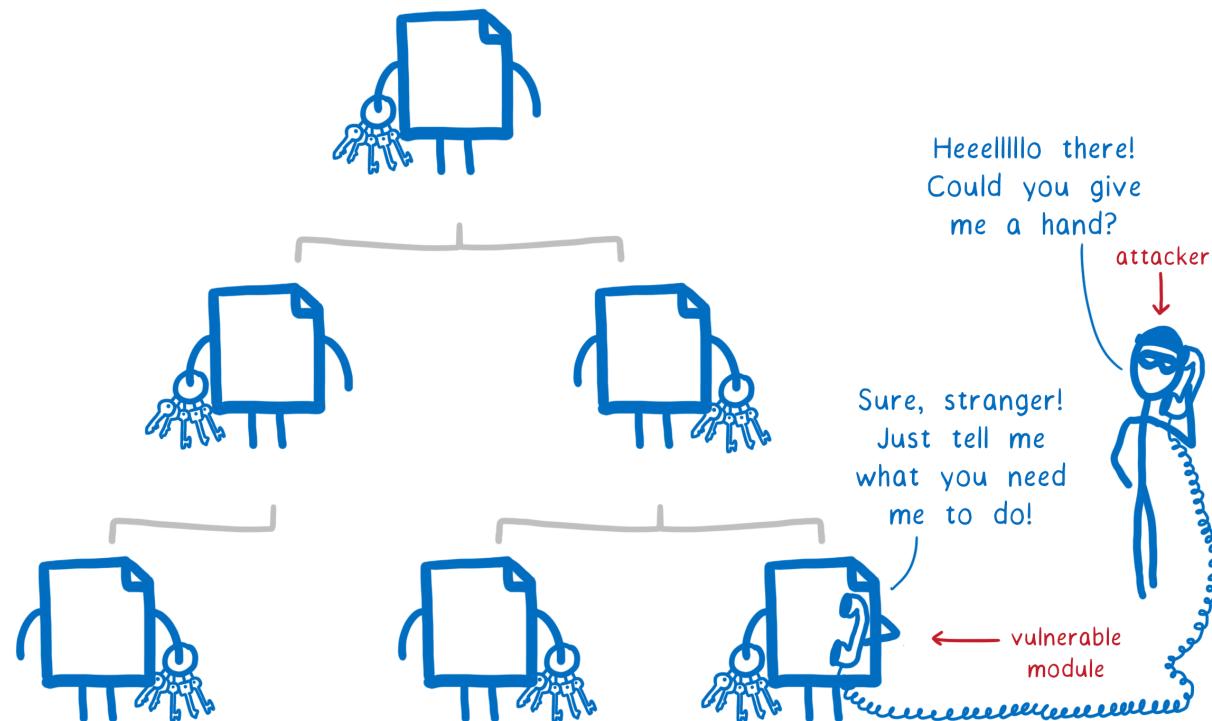


@nielstanis@infosec.exchange

# Average codebase composition



# Vulnerable Assembly



# DotNet CLI

```
nelson@ghost-m2:~/research/consoleapp
└─nelson at ghost-m2 in ~/research/consoleapp
    └─dotnet list package
        Project 'consoleapp' has the following package references
            [net9.0]:
            Top-level Package      Requested   Resolved
            > itext7                9.4.0       9.4.0

nelson@ghost-m2:~/research/consoleapp
└─dotnet list package --vulnerable

The following sources were used:
https://api.nuget.org/v3/index.json

The given project `consoleapp` has no vulnerable packages given the current sources.
└─nelson at ghost-m2 in ~/research/consoleapp
    └─
```

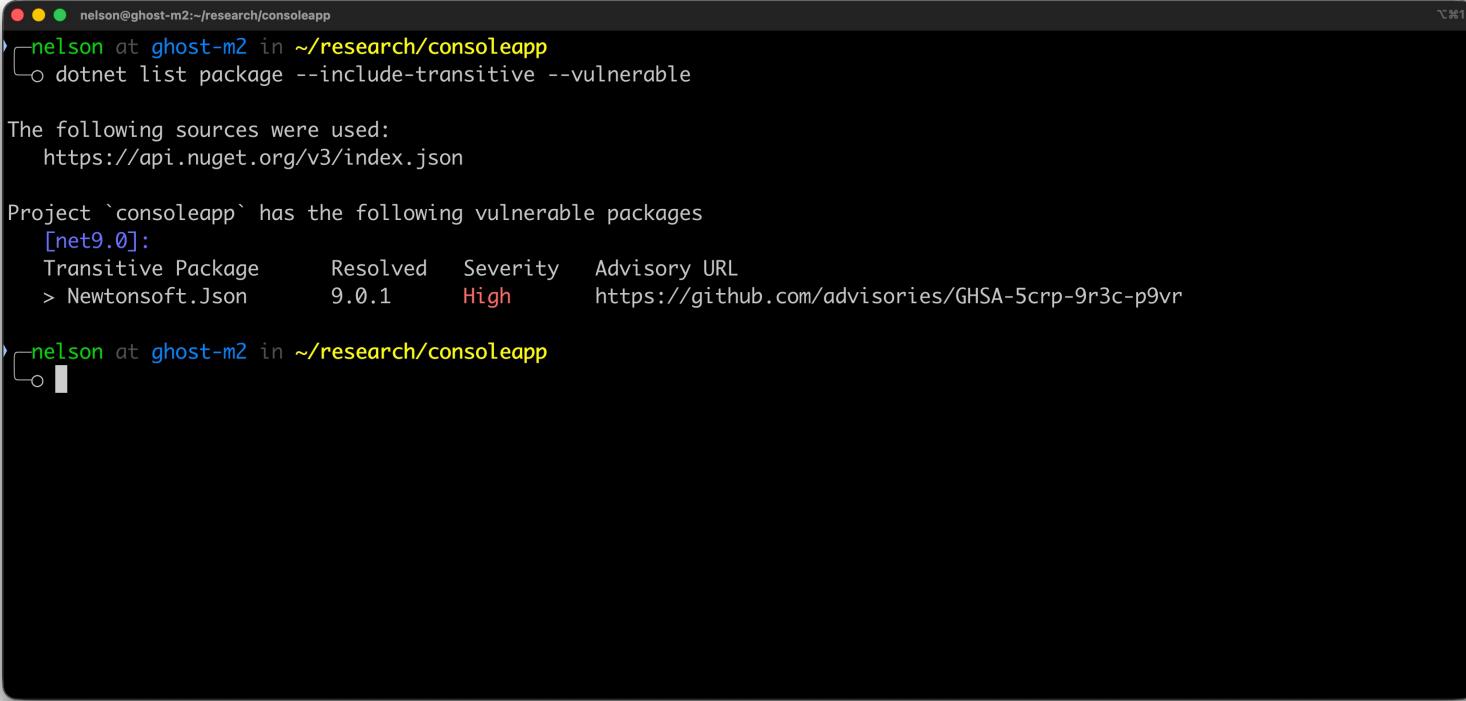
# DotNet CLI

```
nelson at ghost-m2 in ~/research/consoleapp
└─$ dotnet list package --include-transitive
Project 'consoleapp' has the following package references
[net9.0]:
  Top-level Package      Requested   Resolved
  > itext7              9.4.0       9.4.0

  Transitive Package                               Resolved
  > itext                           9.4.0
  > itext.common                     9.4.0
  > 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
  > Microsoft.NETCore.Platforms          1.1.1
  > Microsoft.NETCore.Targets            1.1.3
  > Microsoft.Win32.Primitives          4.3.0
  > Microsoft.Win32.Registry            4.3.0
  > Newtonsoft.Json                    9.0.1
```



# DotNet CLI



```
nelson at ghost-m2 in ~/research/consoleapp
└─o dotnet list package --include-transitive --vulnerable

The following sources were used:
https://api.nuget.org/v3/index.json

Project `consoleapp` has the following vulnerable packages
[net9.0]:
Transitive Package      Resolved  Severity  Advisory URL
> Newtonsoft.Json        9.0.1     High       https://github.com/advisories/GHSA-5crp-9r3c-p9vr

nelson at ghost-m2 in ~/research/consoleapp
└─o
```

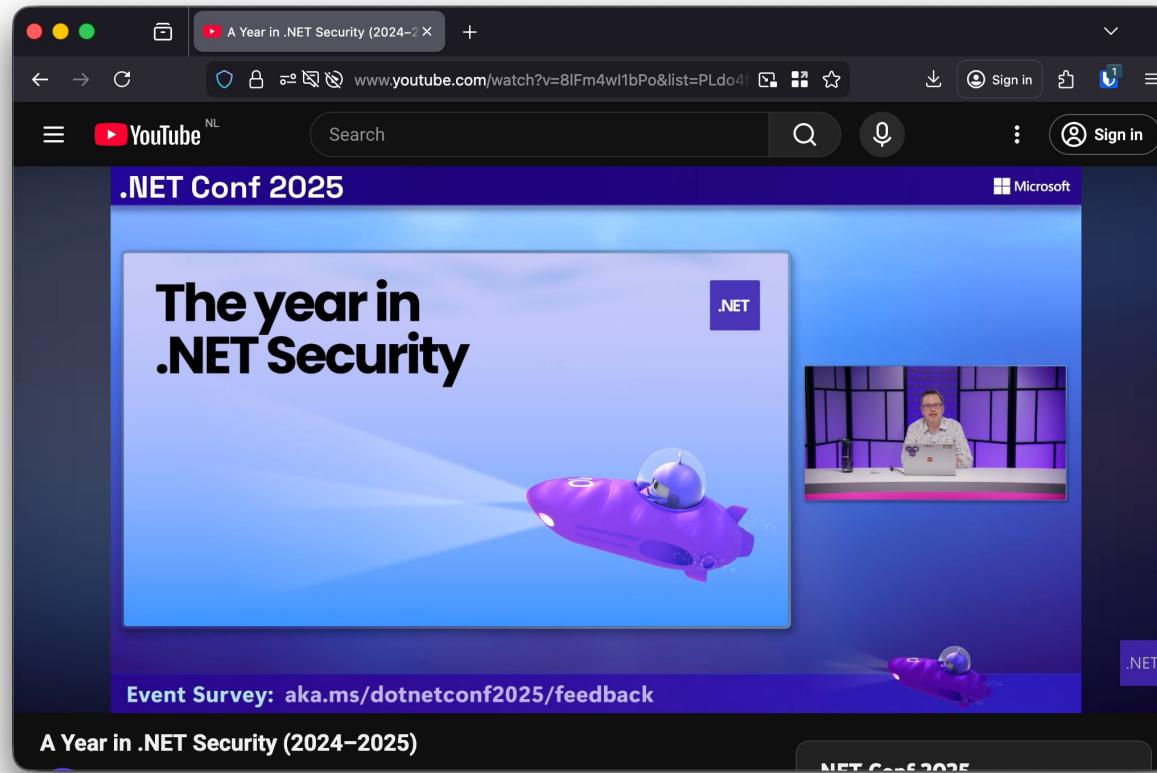
# DotNet CLI - .NET 10

```
nelson@ghost-m2:~/research/consoleapp
└─ nelson at ghost-m2 in ~/research/consoleapp
    └─ dotnet --version
        10.0.100
└─ nelson at ghost-m2 in ~/research/consoleapp
    └─ dotnet build
        Restore succeeded with 1 warning(s) in 0.3s
            /Users/nelson/Research/consoleapp/consoleapp.csproj : warning NU1903: Package 'Newtonsoft.Json' 9.0.1 has a known high
            severity vulnerability, https://github.com/advisories/GHSA-5crp-9r3c-p9vr
            consoleapp net10.0 succeeded with 1 warning(s) (1.5s) → bin/Debug/net10.0/consoleapp.dll
            /Users/nelson/Research/consoleapp/consoleapp.csproj : warning NU1903: Package 'Newtonsoft.Json' 9.0.1 has a known high
            severity vulnerability, https://github.com/advisories/GHSA-5crp-9r3c-p9vr

        Build succeeded with 2 warning(s) in 2.0s
└─ nelson at ghost-m2 in ~/research/consoleapp
    └─
```



# .NET Conf 2025 – The year in .NET Security



NDC { Manchester }

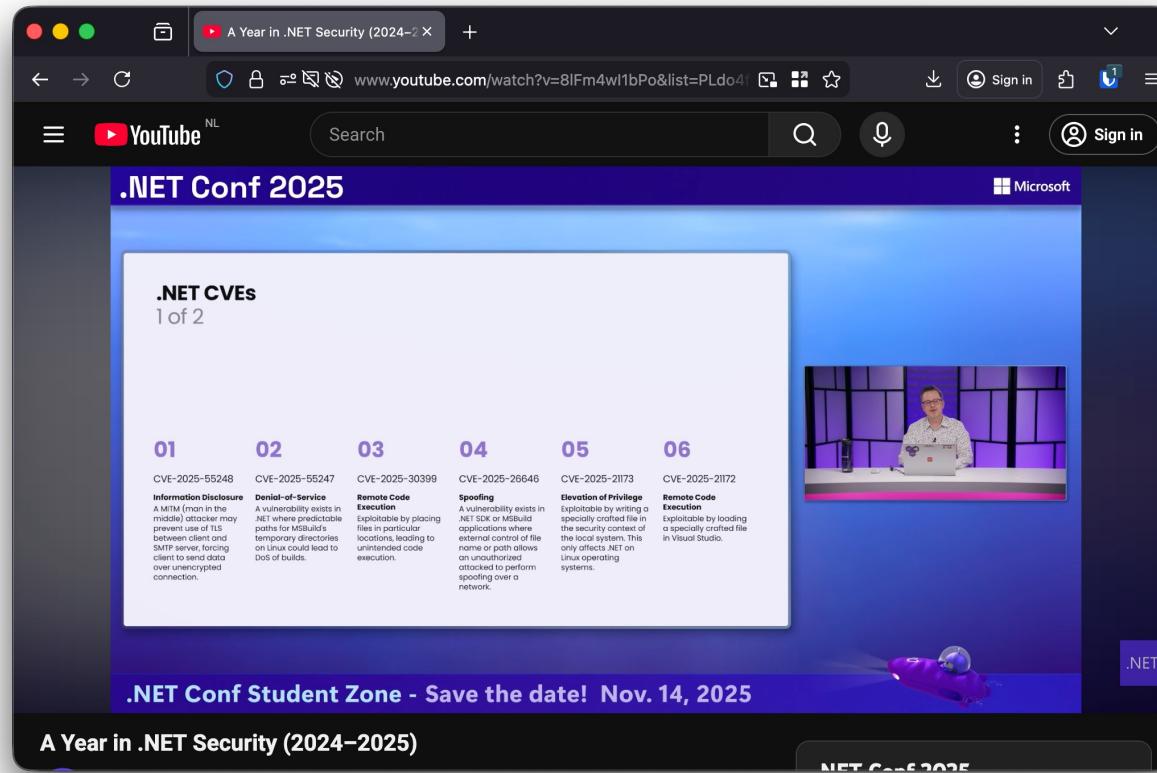


@niels.fennec.dev



@nielstanis@infosec.exchange

# .NET Conf 2025 – The year in .NET Security



NDC { Manchester }

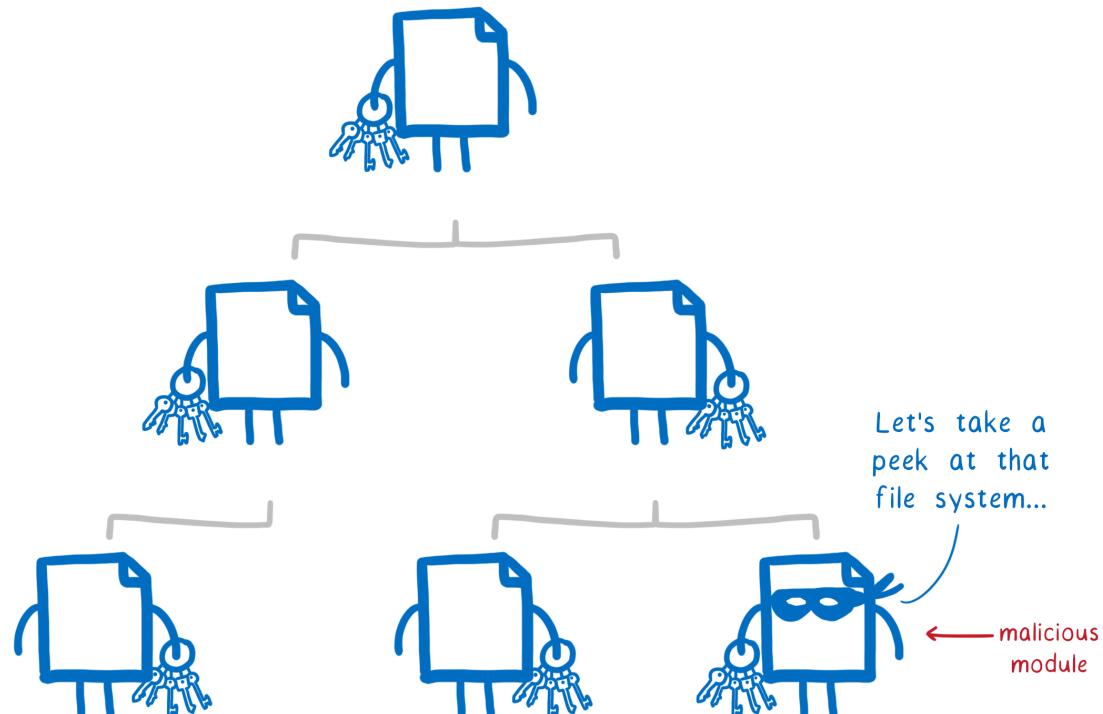


@niels.fennec.dev



@nielstanis@infosec.exchange

# Malicious Assembly



NDC { Manchester }

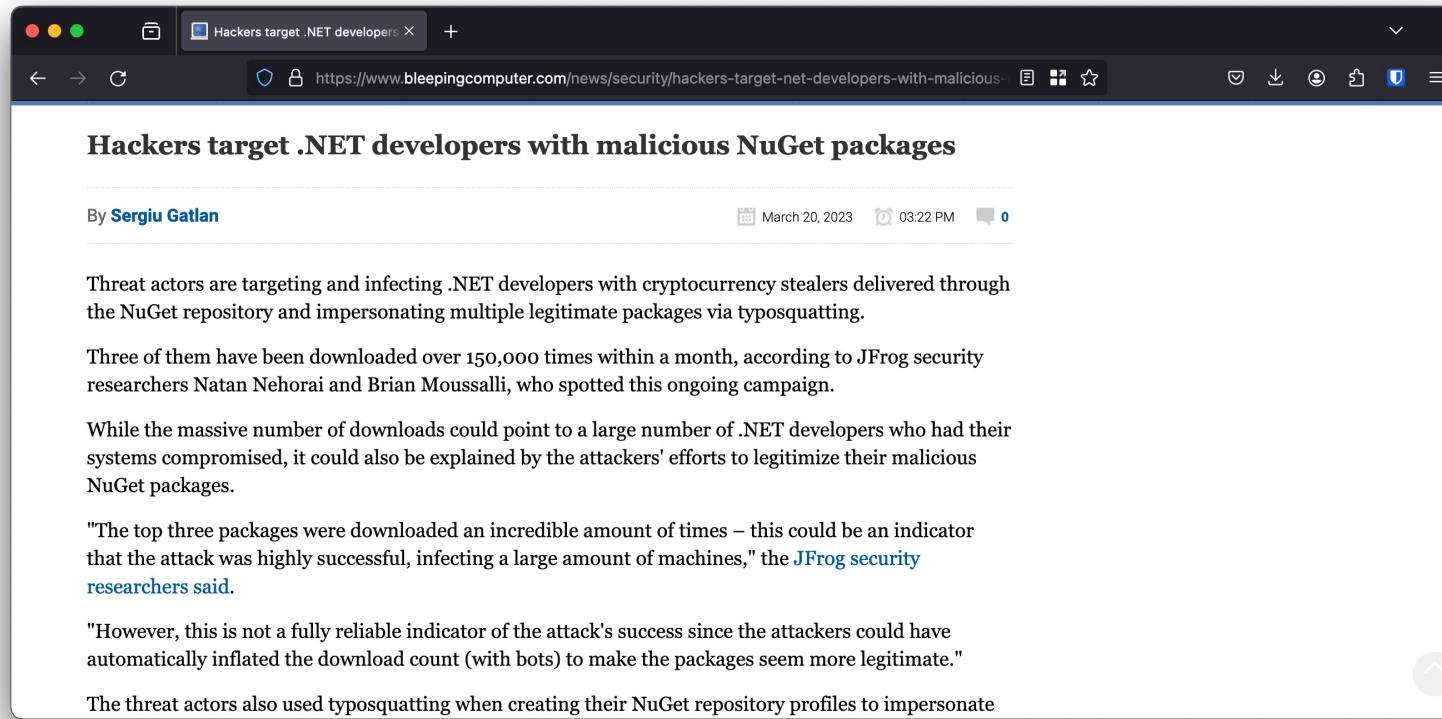


@niels.fennec.dev



@nielstanis@infosec.exchange

# Malicious Package



The screenshot shows a web browser window with a dark theme. The title bar reads "Hackers target .NET developers x". The main content area displays an article from bleepingcomputer.com. The headline is "Hackers target .NET developers with malicious NuGet packages". Below the headline, it says "By Sergiu Gatlan" and "March 20, 2023 03:22 PM 0". The article text 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. The text also quotes JFrog security researchers Natan Nehorai and Brian Moussalli. The bottom of the article ends with "The threat actors also used typosquatting when creating their NuGet repository profiles to impersonate".



# Malicious Package

The screenshot shows a web browser window with a dark blue header and a white content area. The URL in the address bar is [www.reversinglabs.com/blog/malicious-nuget-campaign-uses-homoglyphs-and-il-weaving-to-fool-devs](http://www.reversinglabs.com/blog/malicious-nuget-campaign-uses-homoglyphs-and-il-weaving-to-fool-devs). The page title is "Malicious NuGet campaign uses homoglyphs and IL weaving to fool devs". The main content discusses malware authors using homoglyphs and IL weaving to inject malicious code. The sidebar on the right includes a search bar and a "Topics" section with links to "All Blog Posts", "AppSec & Supply Chain Secur...", "Dev & DevSecOps", "Products & Technology", "Security Operations", and "Threat Research".

NDC { Manchester }



@niels.fennec.dev



@nielstanis@infosec.exchange

# Malicious Package

The screenshot shows a web browser window displaying a blog post from socket.dev. The title of the post is "9 Malicious NuGet Packages Deliver Time-Delayed Destructive Payloads". Below the title, there is a brief summary: "Socket researchers discovered nine malicious NuGet packages that use time-delayed payloads to crash applications and corrupt industrial control systems." To the right of the text, there is a stylized illustration of a bomb connected to a computer monitor and keyboard, set against a dark background. At the bottom of the page, there is author information for "Kush Pandya" and social media sharing icons.

NDC { Manchester }



@niels.fennec.dev



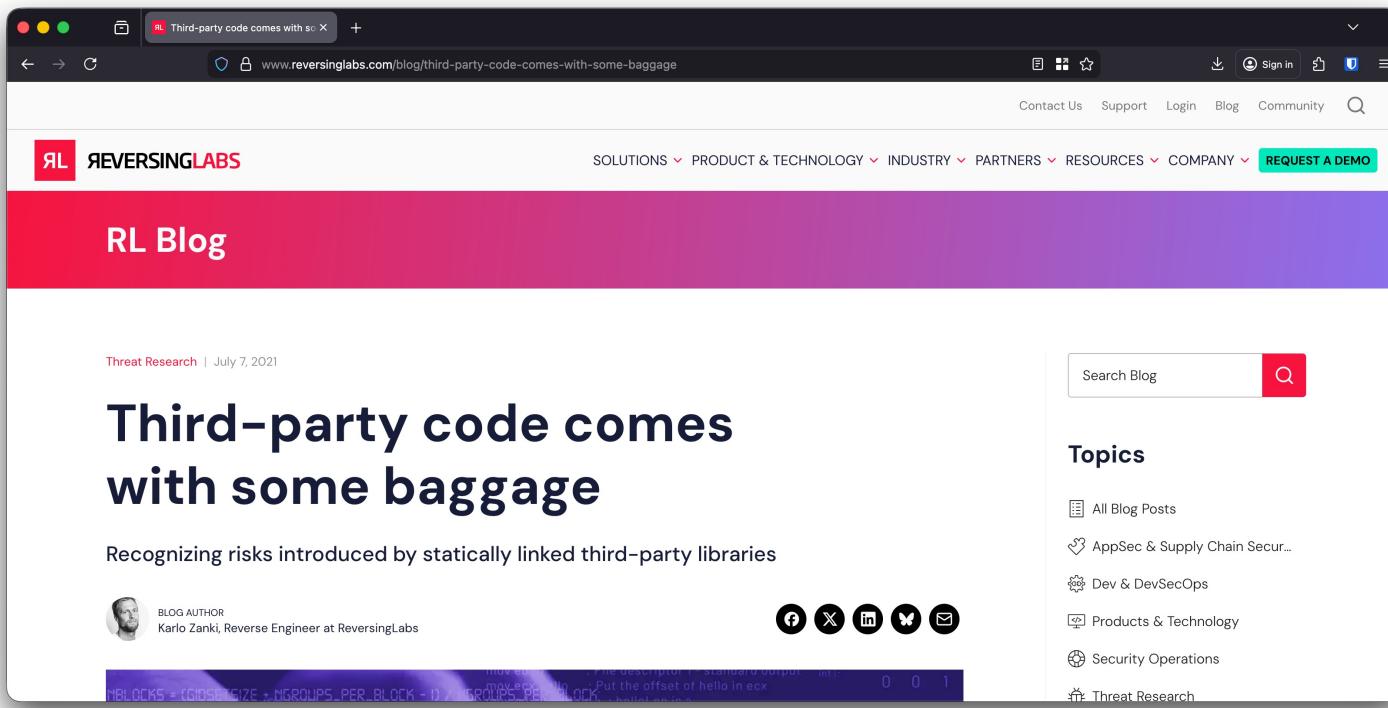
@nielstanis@infosec.exchange

# Malicious Package

- Single publisher called **shanghai666**
- 99% code is valid & functional focusing on databases and industrial PLC's, malicious extension methods
- Delayed logic for Aug '27 & Nov '28
- 20% of database queries are terminated
- Sharp7Extend (typo squatting) for PLC, immediate activation mimics hardware failures



# Do you know what's inside?



NDC { Manchester }



@niels.fennec.dev



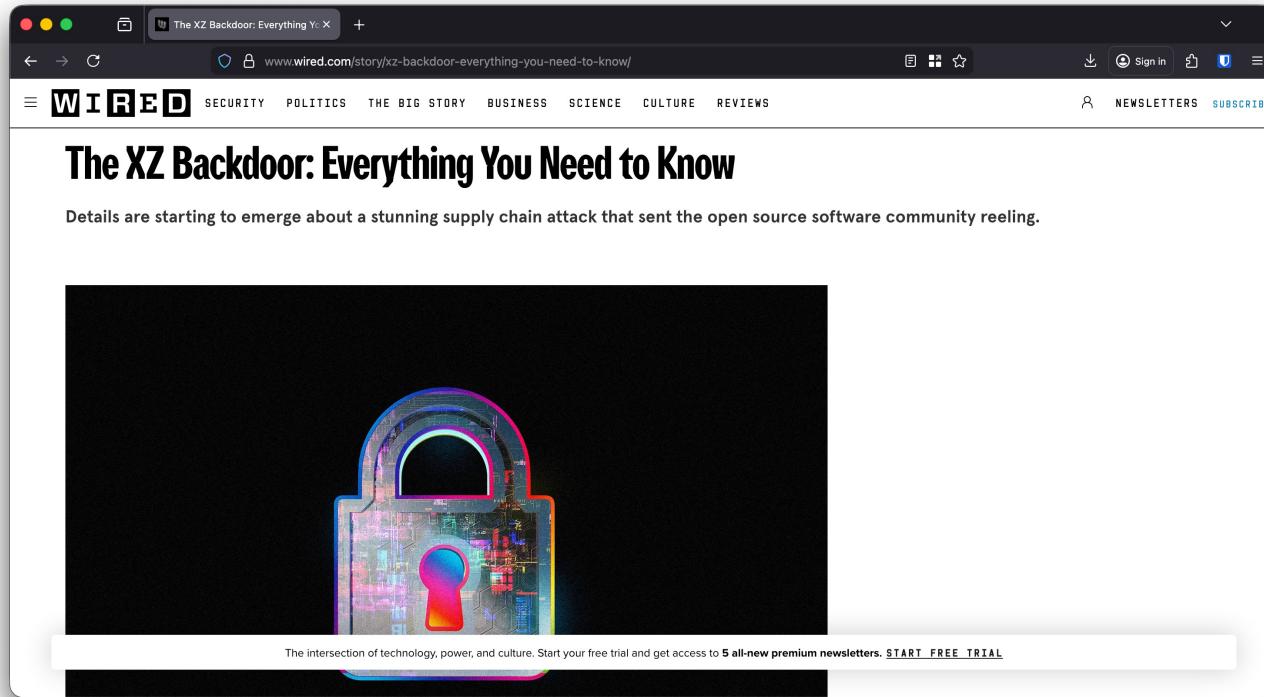
@nielstanis@infosec.exchange

# Do you know what's inside?

- Unmanaged components interop
- Precompiled version of 7Zip, WinSCP and PuTTYgen
- Zlib compression library
  - Only source code distributed
  - Used in medical appliances
  - Severe vulnerabilities since 2005!



# Do you know what's inside? - xz utils



NDC { Manchester }

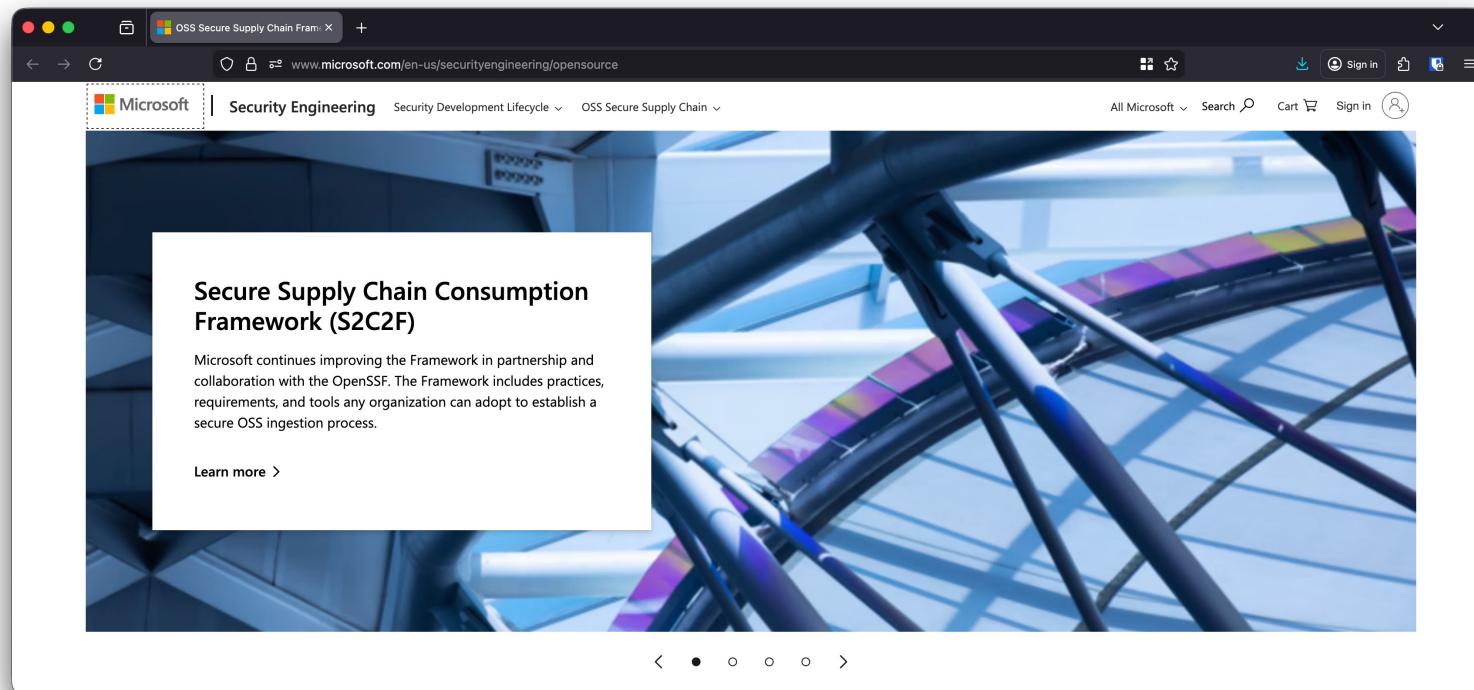


@niels.fennec.dev



@nielstanis@infosec.exchange

# Secure Supply Chain Consumption Framework (S2C2F)



NDC { Manchester }



@niels.fennec.dev



@nielstanis@infosec.exchange

# Secure Supply Chain Consumption Framework (S2C2F)



Level 1	Level 2	Level 3	Level 4
<b>Minimum OSS Governance Program</b> <ul style="list-style-type: none"><li>Use package managers [ING-1]</li><li>Local copy of artifact [ING-2]</li><li>Scan with known vulns [SCA-1]</li><li>Scan for software licenses [SCA-2]</li><li>Inventory OSS [INV-1]</li><li>Manual OSS updates [UPD-1]</li></ul>	<b>Secure Consumption and Improved MTTR</b> <ul style="list-style-type: none"><li>Scan for end of life [SCA-3]</li><li>Have an incident response plan [INV-2]</li><li>Auto OSS updates [UPD-2]</li><li>Alerts on vulns at PR time [UPD-3]</li><li>Audit that consumption is through approved ingestion method [AUD-2]</li><li>Validate integrity of OSS [AUD-3]</li><li>Secure package source file configuration [ENF-1]</li></ul>	<b>Malware Defense and Zero-Day Detection</b> <ul style="list-style-type: none"><li>Deny list capability [ING-3]</li><li>Clone OSS source [ING-4]</li><li>Scan for malware [SCA-4]</li><li>Proactive security reviews [SCA-5]</li><li>Enforce OSS provenance [AUD-1]</li><li>Enforce consumption from curated feed [ENF-2]</li></ul>	<b>Advanced Threat Defense</b> <ul style="list-style-type: none"><li>Validate the SBOMs of OSS consumed [AUD-4]</li><li>Rebuild OSS on trusted infrastructure [REB-1]</li><li>Digitally sign rebuilt OSS [REB-2]</li><li>Generate SBOM for rebuilt OSS [REB-3]</li><li>Digitally sign protected SBOMs [REB-4]</li><li>Implement fixes [FIX-1]</li></ul>

# Hot take; dependency cooldowns

A screenshot of a web browser window displaying a blog post. The browser has a dark theme with a blue header bar. The title bar shows the URL: [blog.yossarian.net/2025/11/21/We-should-all-be-using-dependency-cooldowns](https://blog.yossarian.net/2025/11/21/We-should-all-be-using-dependency-cooldowns). The main content area shows the following:

**ENOSUCHBLOG**  
*Programming, philosophy, pedaling.*

Navigation links: Home, Tags, Series, Favorites, Archive, Main Site, TILs

**We should all be using dependency cooldowns**

**Nov 21, 2025** Tags: [oss](#), [security](#)

---

**TL;DR:** Dependency cooldowns are a free, easy, and **incredibly effective** way to mitigate the *large majority* of open source supply chain attacks. More individual projects should apply cooldowns (via tools like Dependabot and Renovate) to their dependencies, and packaging ecosystems should invest in first-class support for cooldowns directly in their package managers.

Some resources for adding cooldowns:

- [Dependabot](#)
- [Renovate](#)
- [zizmor: dependabot-cooldown](#)



# What's inside? (Audit)



NDC { Manchester }



@niels.fennec.dev



@nielstanis@infosec.exchange

# OpenSSF Security Scorecard

- Set Automated Security Checks
- Scoring System 0-10
- Security Best Practices
- Ease of Use
- Community Support



# Maintenance

- Vulnerabilities (**High**)
  - Does the project have unfixed vulnerabilities?
- Maintained (**High**)
  - Is the project at least 90 days old, and maintained?
- Security Policy (**Medium**)
  - Does project contain security policy?



# Maintenance

- License (**Low**)
  - Does the project declare a licence?
- CII Best Practices (**Low**)
  - Does the project have a CII Best Practices Badge?
- Dependency Update Tool (**High**)
  - Does the project use tools to help update its dependencies e.g. Dependabot, RenovateBot?



# Continuous Testing

- Continuous Integration Tests (Low)
  - Does the project run tests in CI?
- Fuzzing (**Medium**)
  - Does the project use fuzzing tools, e.g. OSS-Fuzz?
- Static Analysis (**Medium**)
  - Does the project use static code analysis tools?



# Source Risk Assessment

- Binary Artifacts (**High**)
  - Does repository contain binaries/compiled artifacts?
- Branch Protection (**High**)
  - GitHub best practices branch protection
- Dangerous Workflow (**Critical**)
  - GitHub best practices for Workflow Actions

# Source Risk Assessment

- Code Review (**High**)
  - Does the project require code review before code is merged?
- Contributors (**Low**)
  - Does the project have contributors from multiple organizations?

# Build Risk Assessment

- Pinned Dependencies (**Medium**)
  - Does the project declare and pin dependencies?
- Token Permission (**High**)
  - Does the project declare GitHub workflow tokens as read only?



# Build Risk Assessment

- Packaging (**Medium**)
  - Does the project build and publish official packages from CI/CD?
- Signed Releases (**High**)
  - Does the project cryptographically sign releases?



# Demo OpenSSF Scorecard

Running checks

NDC { Manchester }



@niels.fennec.dev



@nielstanis@infosec.exchange

# Fennec Labs Scorecard



NDC { Manchester }



@niels.fennec.dev



@nielstanis@infosec.exchange

# .NET Reproducibility

- Reproducible builds
  - Independent path from source to binary code.
- Roslyn Compiler Deterministic Compilation
- How reproducible is a simple console app?

# Fennec Labs Reproduce



NDC { Manchester }



@niels.fennec.dev



@nielstanis@infosec.exchange

# Fennec Labs Instrument



NDC { Manchester }

 @niels.fennec.dev  @nielstanis@infosec.exchange

# Application Inspector

The screenshot shows the Microsoft Application Inspector tool window. The title bar reads "Microsoft Application Inspector". The main content area has a header "Application Features" with a sub-section about feature groups and associated rules. Below this 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)

NDC { Manchester }



@niels.fennec.dev



@nielstanis@infosec.exchange

# Application Inspector

— Select Features

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



# Fennec Labs Compare



NDC { Manchester }

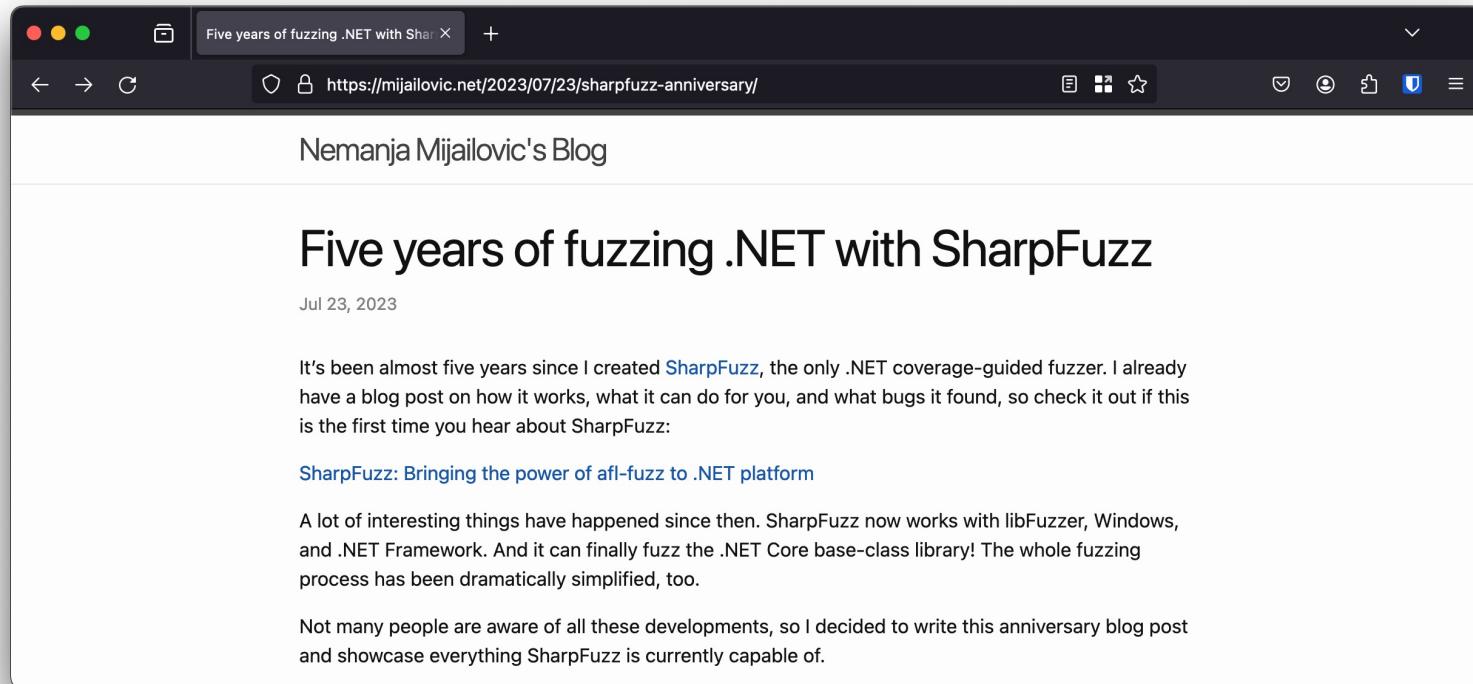
 @niels.fennec.dev  @nielstanis@infosec.exchange

# Future Idea - Fuzzing

- Fuzzing, or fuzz testing
  - Automated software testing method that uses a wide range of *invalid* and unexpected data as input to find flaws
- Good in C/C++ memory issues
- Can it be of any value with managed languages like .NET?



# Fuzzing .NET & SharpFuzz



A screenshot of a web browser window showing a blog post. The title of the browser tab is "Five years of fuzzing .NET with SharpFuzz". The URL in the address bar is <https://mijailovic.net/2023/07/23/sharpfuzz-anniversary/>. The page content is from "Nemanja Mijailovic's Blog" and is titled "Five years of fuzzing .NET with SharpFuzz". It was posted on Jul 23, 2023. The text discusses the creation of SharpFuzz and its evolution over five years, mentioning its coverage-guided nature, support for libFuzzer, Windows, and .NET Framework, and its ability to fuzz the .NET Core base-class library. It also notes that many people are unaware of these developments and aims to showcase SharpFuzz's capabilities.

## 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:

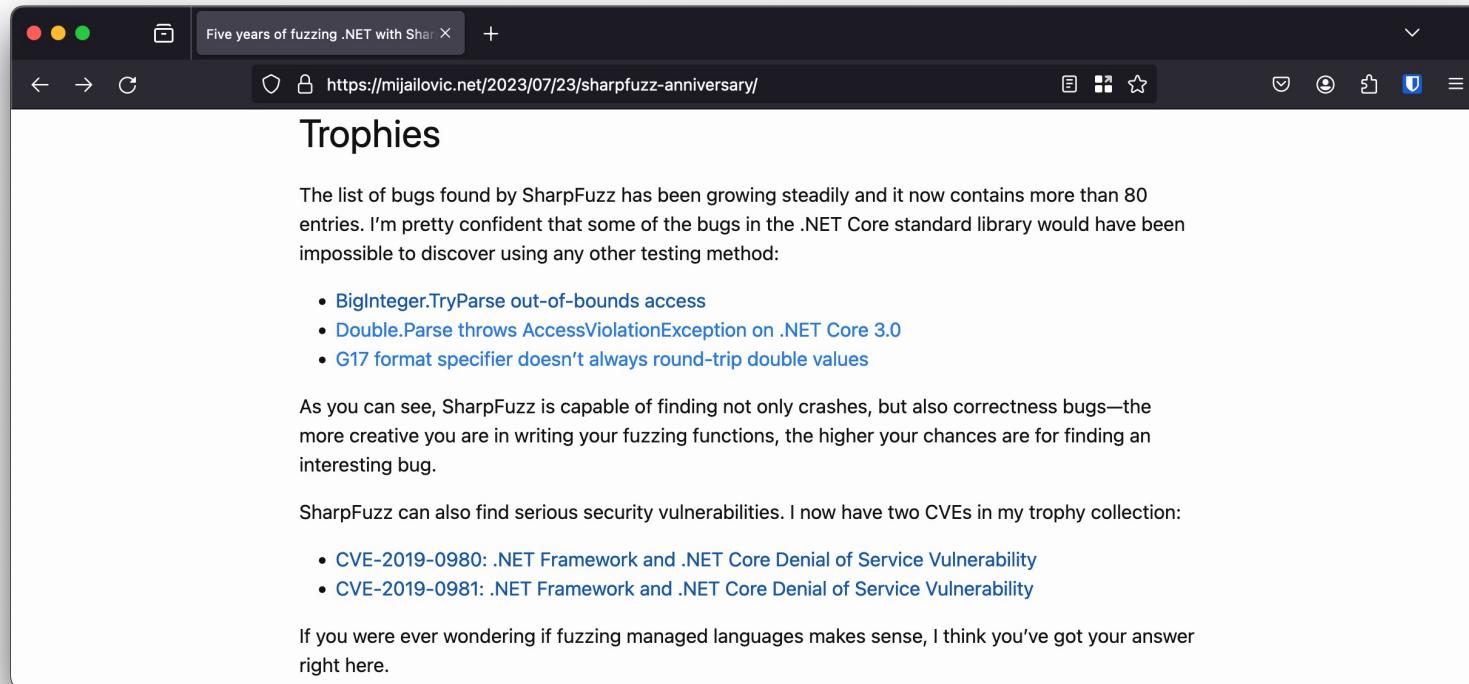
[SharpFuzz: Bringing the power of afl-fuzz to .NET platform](#)

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.



# Fuzzing .NET & SharpFuzz



The screenshot shows a web browser window with a dark theme. The title bar reads "Five years of fuzzing .NET with SharpFuzz". The address bar shows the URL <https://mijailovic.net/2023/07/23/sharpfuzz-anniversary/>. The main content area has a white background and features the following text:

## 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) { }
    });
}
```

NDC { Manchester }



@niels.fennec.dev



@nielstanis@infosec.exchange

# Fuzzomatic: Using AI to Fuzz Rust

The screenshot shows a web browser window with a dark theme. The title bar reads "Introducing Fuzzomatic: Using AI to automatically fuzz Rust projects from scratch". The main content area has a white background and features the following text:

**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.

At the bottom right of the content area, there is a footer with social media links: "Comment", "Reblog", "Subscribe", and "...".



# Future Idea – Dashboard & Feedback

- Local dashboard for project
- Remote dashboard for community
- Ability to do review and share security related data of project
- MCP server with embedded prompt resources to help security review
- Roslyn analyzer that can help guide the developer



# Conclusion

- Doing software security is hard and Software Security Professionals don't scale!
- Package eco-systems have been wild west lately
- Goal is to have Fennec Labs help with reviewing and sharing security details for all audiences
- Allowing everyone to make better informed decisions and can act if needed.



# Thank you!

- ntanis at Veracode.com
- <https://github.com/niestanis/ndcmanchester2025>
- Questions?

