





OWASP 2024
GLOBAL
AppSec

CONGRESS CENTRE
LISBON
JUNE 24-28

Assessing 3rd Party Libraries more easily with Security Scorecards

Niels Tanis
Sr. Principal Security Researcher

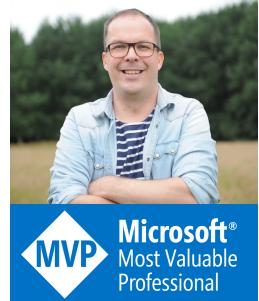
VERACODE



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

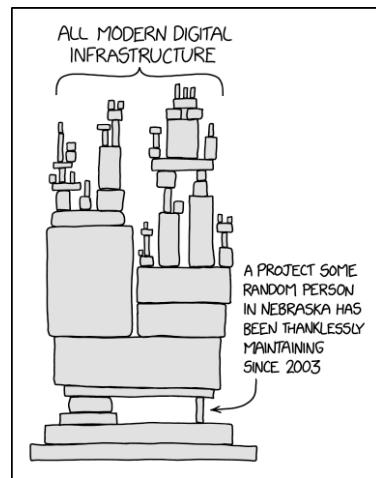
VERACODE



MVP Microsoft®
Most Valuable
Professional

 @nielstanis@infosec.exchange

Modern Application Architecture XKCD 2347



@nielstanis@infosec.exchange

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

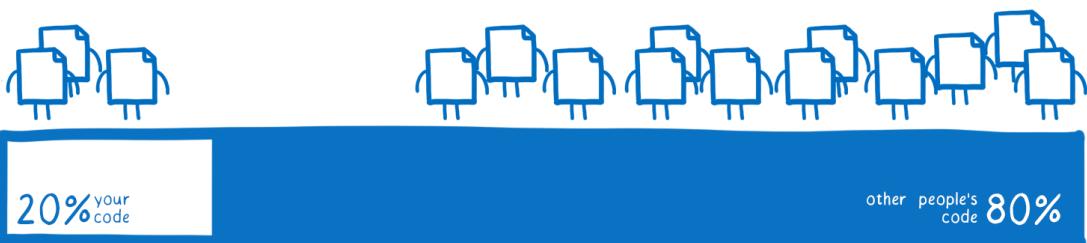
Agenda

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



@nielstanis@infosec.exchange

Average codebase composition



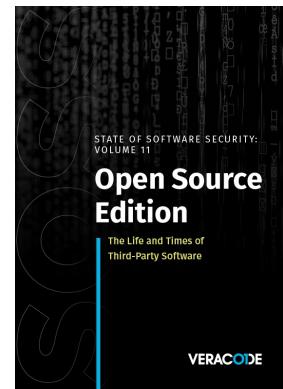
@nielstanis@infosec.exchange

<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



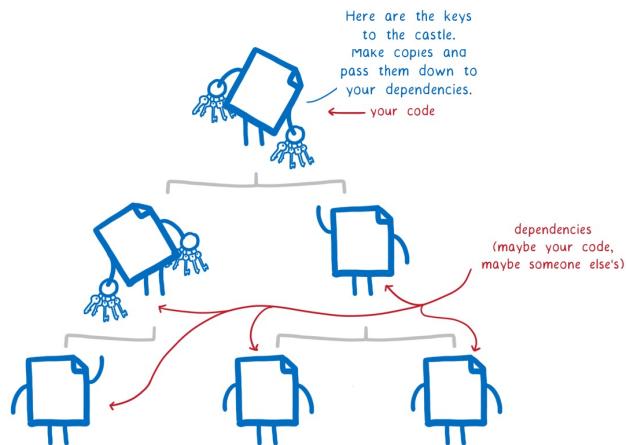
State of Log4j - 2 years later

- 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>

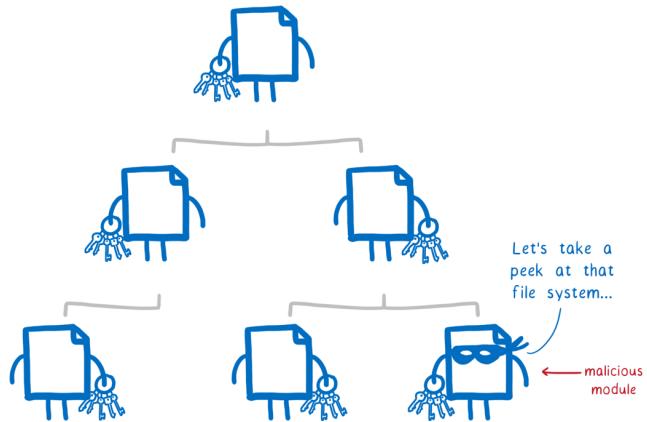
Average codebase composition



@nielstanis@infosec.exchange

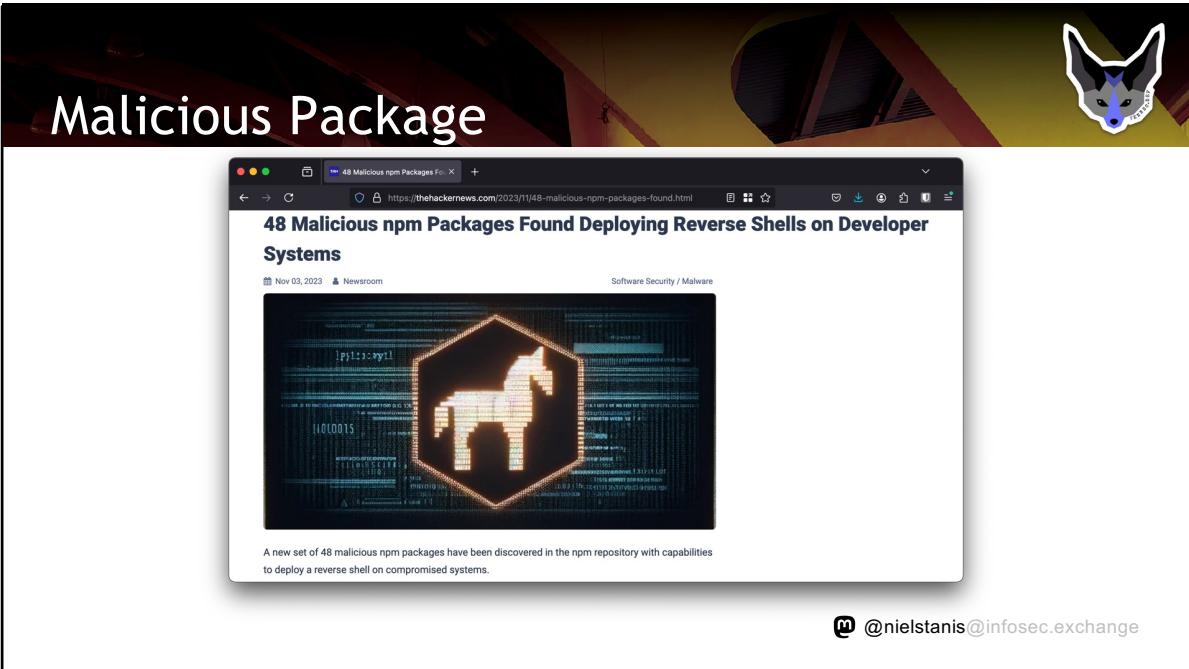
<https://hacks.mozilla.org/2019/11/announcing-the-bytecode-alliance/>

Malicious Package

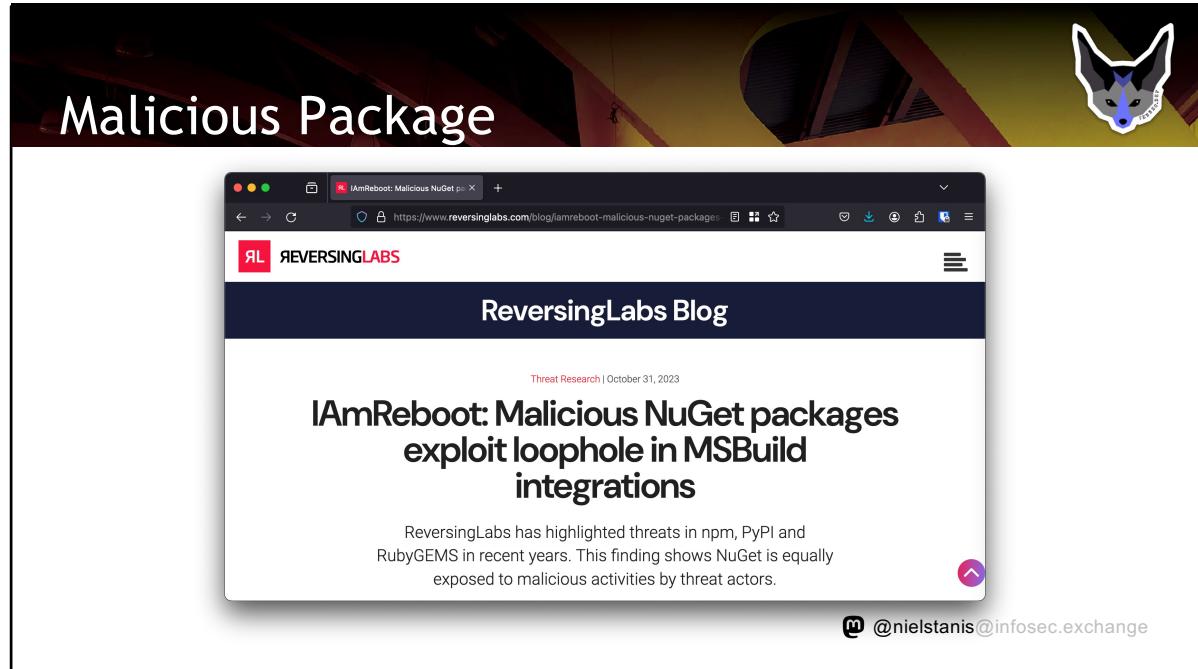


@nielstanis@infosec.exchange

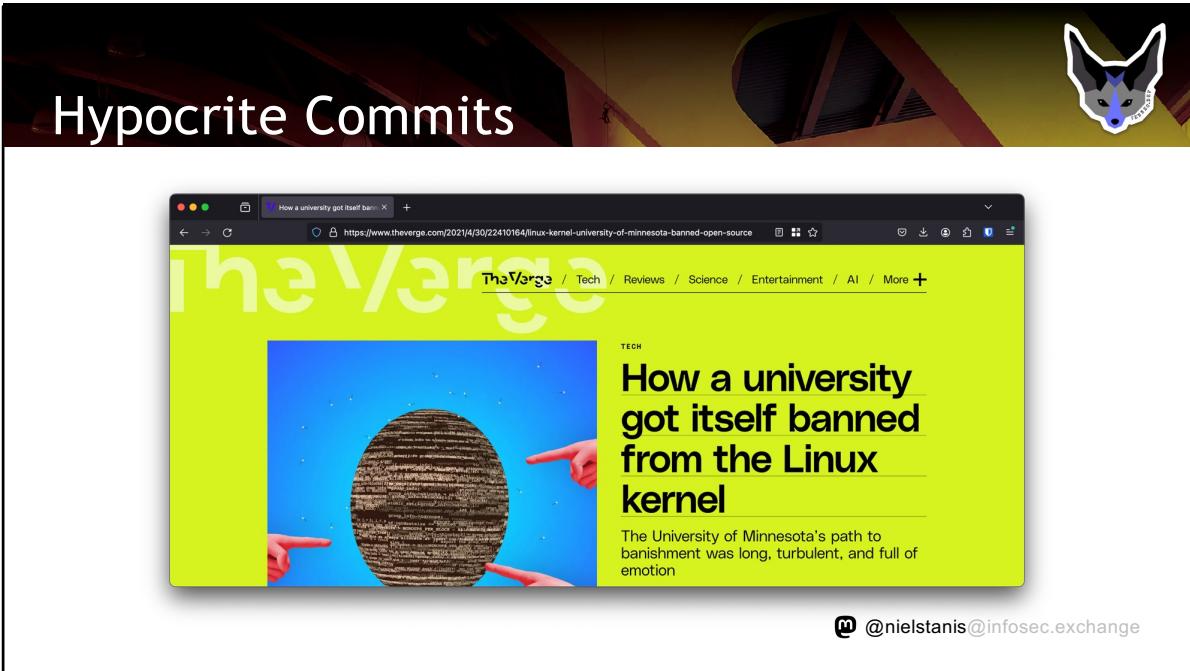
<https://hacks.mozilla.org/2019/11/announcing-the-bytecode-alliance/>



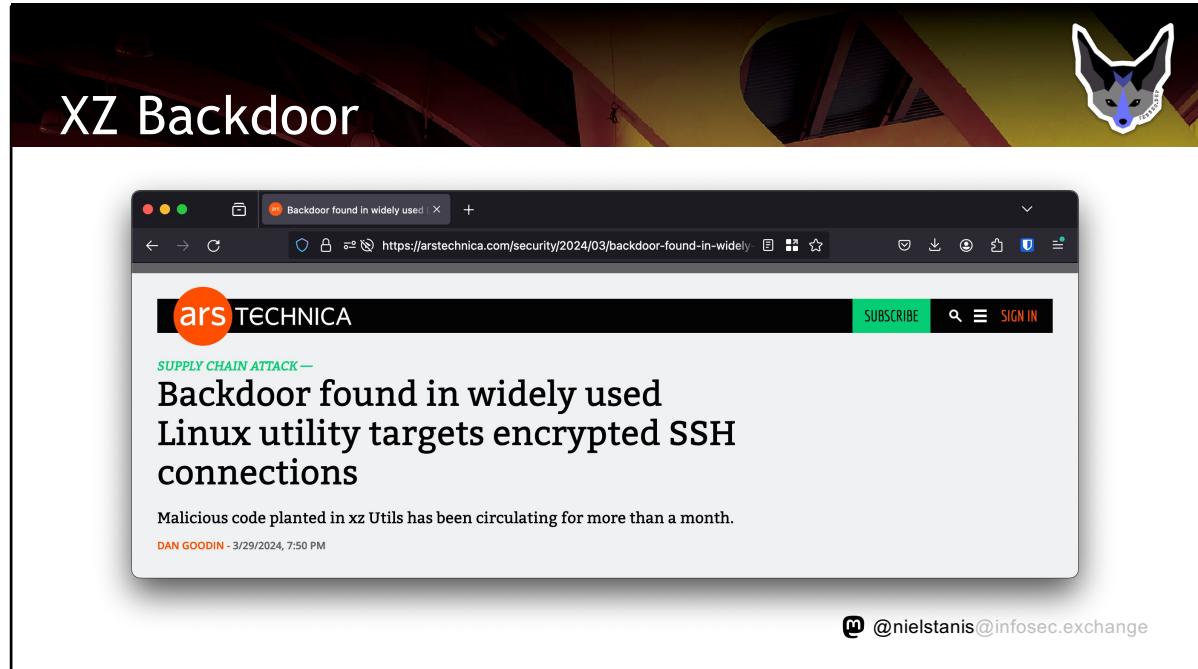
<https://thehackernews.com/2023/11/48-malicious-npm-packages-found.html>



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

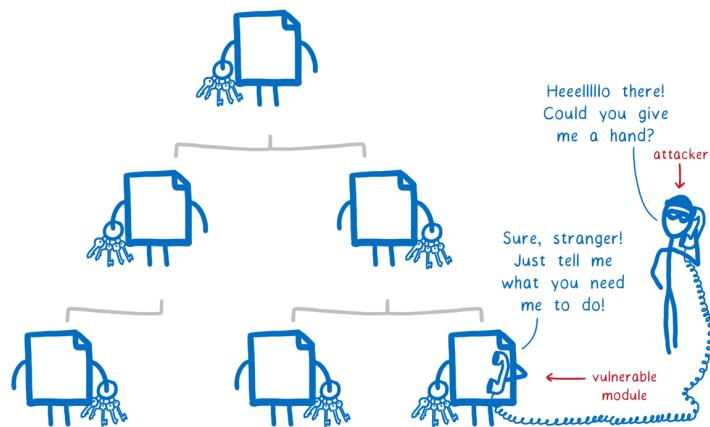


<https://www.theverge.com/2021/4/30/22410164/linux-kernel-university-of-minnesota-banned-open-source>



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

Vulnerable Package



@nielstanis@infosec.exchange

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

The screenshot shows a presentation slide with a dark background featuring a stylized fox logo in the top right corner. The main title 'Vulnerabilities in Libraries' is displayed prominently at the top left. Below the title is a screenshot of a GitHub issue page for Microsoft's .NET Security Advisory CVE-2023-36558. The GitHub interface includes a header with 'dotnet / announcements', a sidebar with 'Issues 283', and a central content area showing the advisory details. The advisory itself is titled 'Microsoft Security Advisory CVE-2023-36558: .NET Security Feature Bypass Vulnerability'. It includes sections for 'Executive summary' and 'Discussion', along with a sidebar for project management. At the bottom right of the slide, there is a small footer with a profile icon and the email address '@nielstanis@infosec.exchange'.

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

NPM Audit



```
==== npm audit security report ===
```

```
# Run npm install chokidar@2.0.3 to resolve 1 vulnerability
SEMVER WARNING: Recommended action is a potentially breaking change
```

Low	Prototype Pollution
Package	deep-extend
Dependency of	chokidar
Path	chokidar > fsevents > node-pre-gyp > rc > deep-extend
More info	https://nodesecurity.io/advisories/612

@nielstanis@infosec.exchange

DotNet CLI



```
nelson@ghost-m2 ~/research/consoleapp $ dotnet list package
Project 'consoleapp' has the following package references
[net8.0]:
Top-level Package      Requested    Resolved
> docgenerator          1.0.0        1.0.0

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

The following sources were used:
https://f.feedz.io/fennec/docgenerator/nuget/index.json
https://api.nuget.org/v3/index.json

The given project `consoleapp` has no vulnerable packages given the current sources.
nelson@ghost-m2 ~/research/consoleapp $
```

@nielstanis@infosec.exchange

DotNet CLI



```
nelson@ghost-m2 ~/research/consoleapp $ dotnet list package --include-transitive
Project 'consoleapp' has the following package references
[net8.0]:
Top-level Package      Requested   Resolved
> docgenerator        1.0.0       1.0.0

Transitive Package                               Resolved
> itext7                                         7.2.2
> itext7.common                                     7.2.2
> Microsoft.CSharp                                4.0.1
> Microsoft.DotNet.PlatformAbstractions           1.1.0
> Microsoft.Extensions.DependencyInjection             5.0.0
> Microsoft.Extensions.DependencyInjection.Abstractions 5.0.0
> Microsoft.Extensions.DependencyModel            1.1.0
> Microsoft.Extensions.Logging                     5.0.0
> Microsoft.Extensions.Logging.Abstractions         5.0.0
> Microsoft.Extensions.Options                   5.0.0
> Microsoft.Extensions.Primitives                5.0.0
```

@nielstanis@infosec.exchange

DotNet CLI

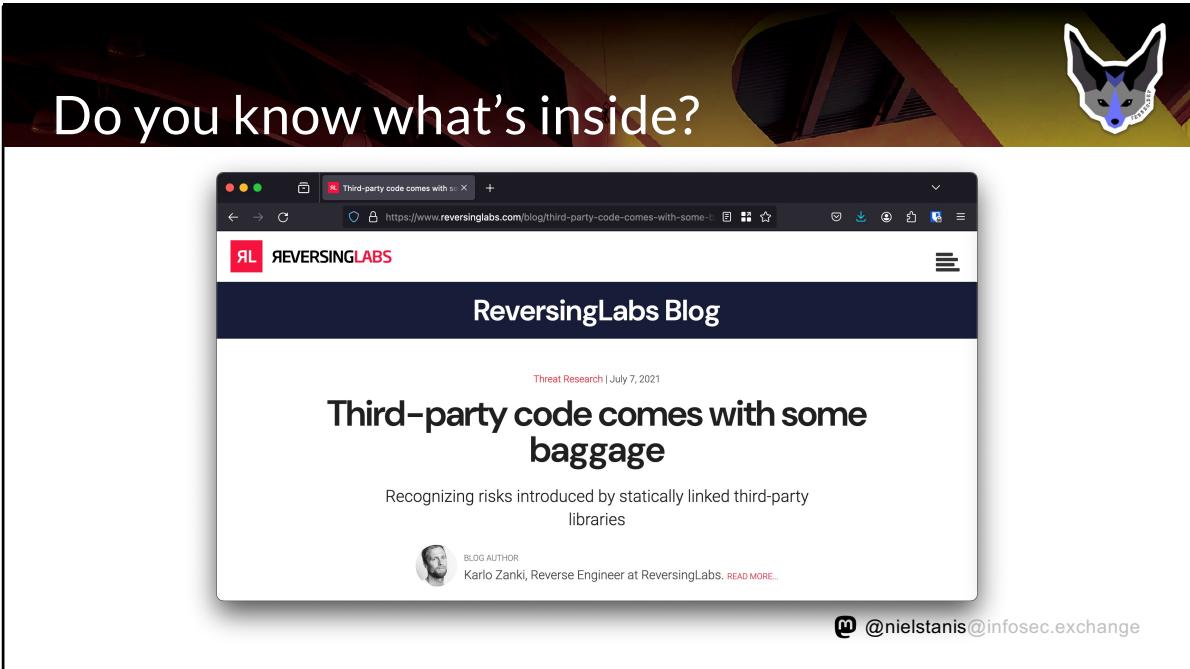


```
nelson@ghost-m2 ~/research/consoleapp $ dotnet list package --vulnerable --include-transitive
The following sources were used:
https://f.feedz.io/fennec/docgenerator/nuget/index.json
https://api.nuget.org/v3/index.json

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



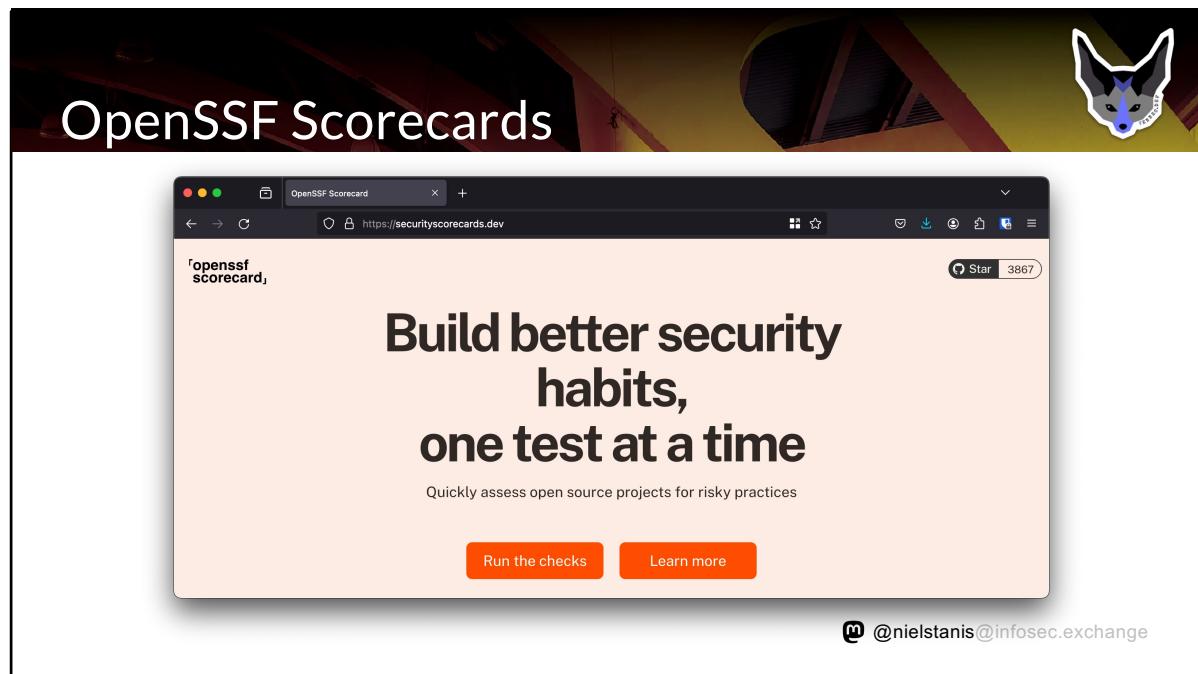
<https://www.reversinglabs.com/blog/third-party-code-comes-with-some-baggage>

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". 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 two columns. The left column contains text about the scorecard's purpose and creation. The right column contains text about its use and best practices. At the bottom of the content area are two small icons: a red circle with a white dot and a grid of squares.

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 for the OpenSSF Security Scorecards. The title bar says "OpenSSF Scorecard". The URL in the address bar is "https://securityscorecards.dev/#how-it-works". The main content area has a dark background with a stylized fox logo in the top right corner. The title "How it works" is centered at the top of the page. Below it, there are two sections: "Run the checks" and "Learn more".

Run the checks

- Using the GitHub Action
- Using the CLI

Learn more

- The problem
- What is OpenSSF Scorecard?
- How it works**
- The checks
- Use cases
- About the project name
- Part of the OSS community
- Get involved

The main text area explains the process: "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."

Below this, there are three horizontal bars representing risk levels:

- CRITICAL RISK: Score 10
- HIGH RISK: Score 7.5
- MEDIUM RISK: Score 5

At the bottom right of the page, there is a small footer with a profile icon and the email address "@nielstanis@infosec.exchange".

<https://securityscorecards.dev/>

OpenSSF Security Scorecards

The screenshot shows the homepage of the OpenSSF Security Scorecards. On the left, there's a sidebar with links for "Run the checks" (GitHub Action, CLI) and "Learn more" (problem, what it is, how it works, checks, use cases, project name, community, get involved). The main content area features a large diagram titled "HOLISTIC SECURITY PRACTICES" in red text, surrounded by five interconnected circles: "CODE VULNERABILITIES" at the top, "BUILD ASSESSMENT" on the left, "MAINTENANCE" on the right, "SOURCE RISK ASSESSMENT" at the bottom left, and "CONTINUOUS TESTING" at the bottom right. A small blue fox logo is in the top right corner of the slide.

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

HOLISTIC SECURITY PRACTICES

CODE VULNERABILITIES

BUILD ASSESSMENT

MAINTENANCE

SOURCE RISK ASSESSMENT

CONTINUOUS TESTING

@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 0.10.1 0.11.0 0.12.0 0.2.0 0.3.0 —	yesterday	Fix available
GHSA-m5cv-6rdh-3yj9	PIP:azure-identity npm/@azure/identity Maven/com.azure.azure-identity npm/@azure/msal-node NuGet/Microsoft.Identity.Client GoLang/com.azure.azure-sdk-for-go/sdk/azidentity Maven/com.microsoft.azure/msal4 NuGet/Azure.Identity	Azure Identity Libraries and Microsoft Authentication Library Elevation of Privilege Vulnerability	1.0.0 1.0.0b2 1.0.0b3 1.0.0b4 1.0.1 1.1.0 —	yesterday	Fix available
GHSA-rq9-xiam-wrfw	NuGet/Umbraco.Commerce	Umbraco Commerce vulnerable to Stored Cross-site Scripting on Print Functionality	12.0.0 12.1.0 12.1.0-re1 12.1.1	2 weeks ago	Fix available

@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.

 @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**)



- 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).

@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 managed languages based 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 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!**

 @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 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)

@nielstanis@infosec.exchange

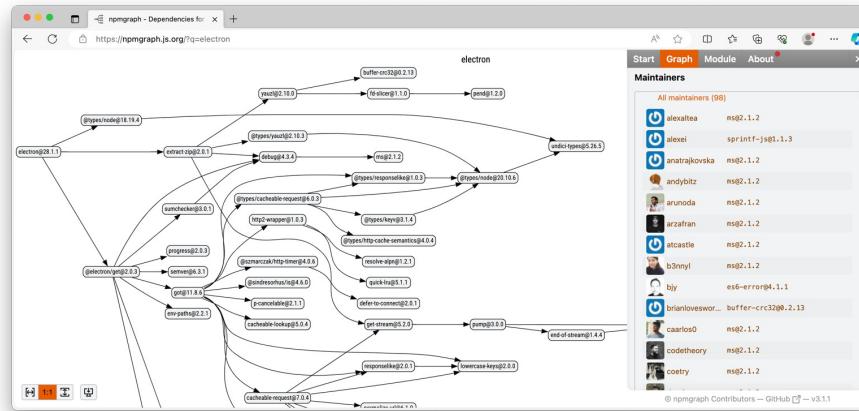
Source Risk Assessment Contributors (**Low**)



- 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?

@nielstanis@infosec.exchange

Source Risk Assessment Contributors (Low)



 @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?

@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

Measure?



@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

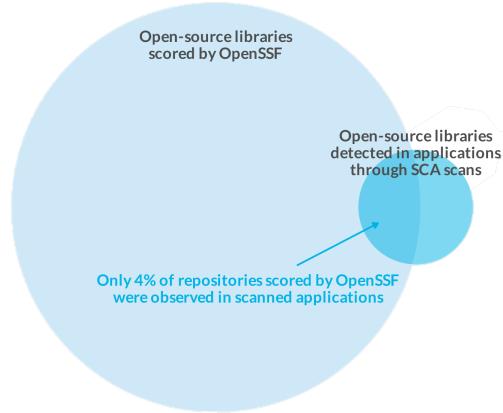
<https://openssf.org/download-the-2023-openssf-annual-report/>

SOSS & OpenSSF Scorecard



@nielstanis@infosec.exchange

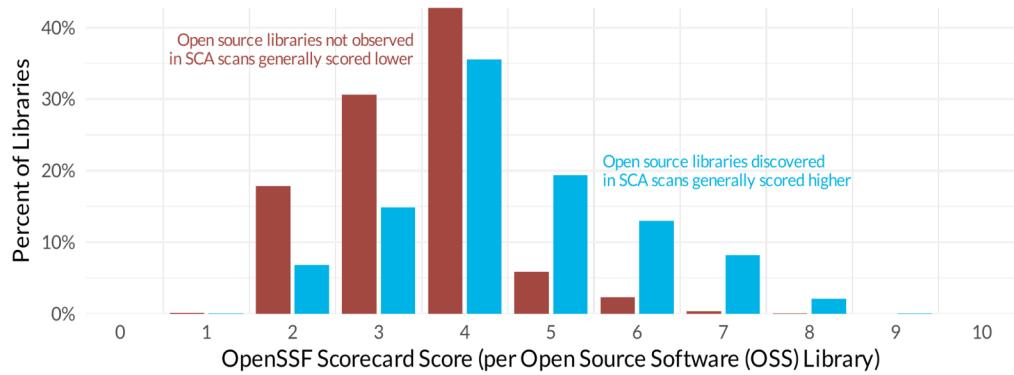
SOSS & OpenSSF Scorecard



@nielstanis@infosec.exchange

<https://www.rsaconference.com/Library/presentation/usa/2024/quantifying%20the%20probability%20of%20flaws%20in%20open%20source>

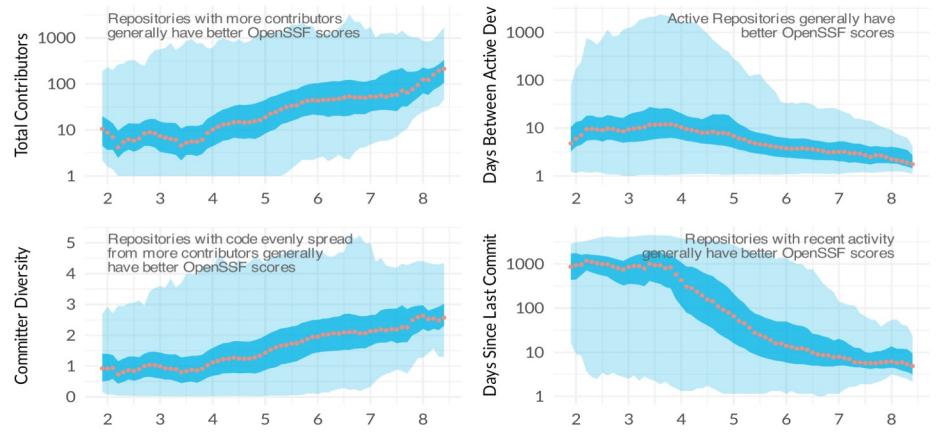
Correlation between SOSS



@nielstanis@infosec.exchange

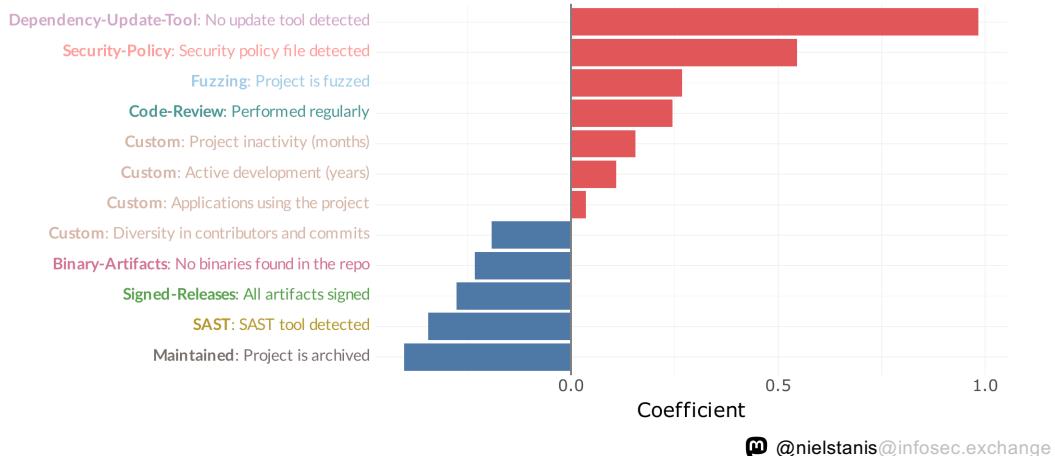


Github commits vs OpenSSF



@nielstanis@infosec.exchange

What really contributes to OSS Security?



What to improve?



 @nielstanis@infosec.exchange

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

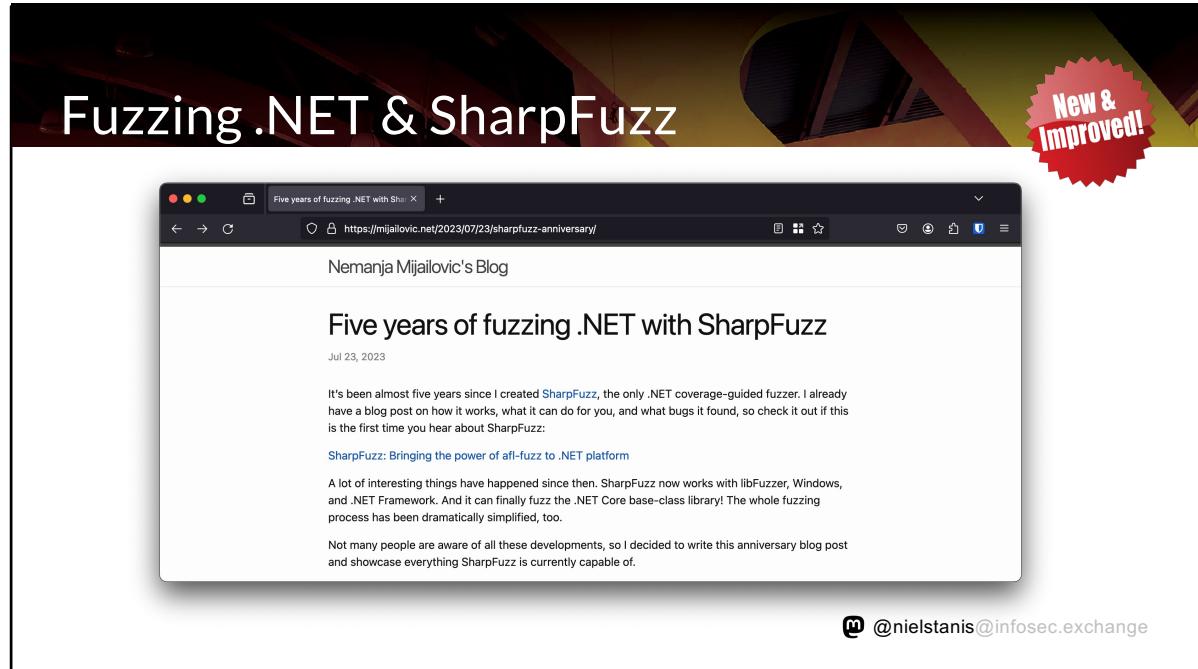
Fuzzing managed languages?



- 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 and/or Java?

 @nielstanis@infosec.exchange

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



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

The screenshot shows a web browser window with the title bar "Five years of fuzzing .NET with SharpFuzz". The main content area displays a blog post titled "Trophies". The post discusses the growth of bugs found by SharpFuzz, mentioning over 80 entries, and highlights three specific bugs: BigInteger.TryParse out-of-bounds access, Double.Parse throwing AccessViolationException on .NET Core 3.0, and G17 format specifier not always round-trip double values. It also notes that SharpFuzz finds correctness bugs in addition to crashes. Two CVEs are mentioned: CVE-2019-0980 (.NET Framework and .NET Core Denial of Service Vulnerability) and CVE-2019-0981 (.NET Framework and .NET Core Denial of Service Vulnerability). A red starburst badge in the top right corner says "New & Improved!". Below the browser window, there is a social media handle "@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!

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.

@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>

Package Reproducibility

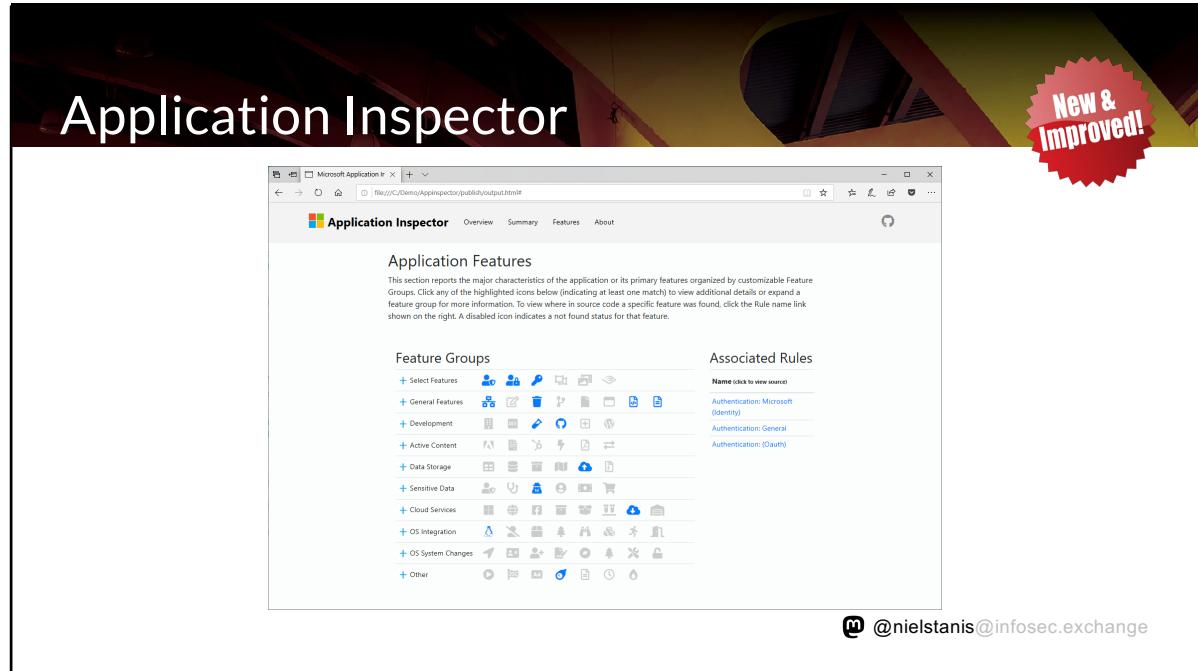


- A build is **reproducible** if given the same source code, build environment and build instructions, any party can recreate bit-by-bit identical copies of all specified artifacts.
 - .NET Roslyn is deterministic →
.NET8 Console App Demo in repo

 @nielstanis@infosec.exchange

The screenshot shows a web browser window displaying the Reproducible-Builds.org documentation. The title bar reads "Definitions -- reproducible-builds.org". The main content area is titled "Definitions" and contains sections on what makes a build reproducible, explanations, and achieving deterministic builds. A sidebar on the left lists navigation links like Home, News, Documentation, Tools, Who is involved?, Talks, Events, Continuous tests, Contribute, and Sponsors. The right sidebar includes links to the documentation home, introduction, and various technical topics such as "Definitions", "Builds", "Variations in the build environment", and "Deterministic build systems". A footer at the bottom right of the page includes a Twitter icon and the email address "@nielstanis@infosec.exchange".

Reproducible-Builds.org



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

The screenshot shows the Microsoft Application Inspector interface. At the top, there's a banner with the text "Application Inspector" and a red starburst badge that says "New & Improved!". Below the banner is a table with the following data:

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 left, there's a link "Select Features". On the right side, there's a Twitter handle "@nielstanis@infosec.exchange".

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

Community Review

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

New & Improved!

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

@nielstanis@infosec.exchange

Conclusion & QA

- Scorecard helps security reviewing a 3rd Party 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!
- Look into metrics that matter for you!



 @nielstanis@infosec.exchange

Conclusion QA

- <https://github.com/nielstanis/appseceu2024>
 - NPM rocket project
 - NuGet Package Scoring (NET Score)
 - Fennec CLI & Generic tool



 @nielstanis@infosec.exchange

Questions?



 @nielstanis@infosec.exchange



OWASP 2024
GLOBAL
AppSec

CONGRESS CENTRE
LISBON
JUNE 24-28



THANK YOU