VERACODE

Reviewing NuGet Packages security easily using OpenSSF Scorecard

Niels Tanis
Sr. Principal Security Researcher

devday
GET INSPIRED

PLATINIUM

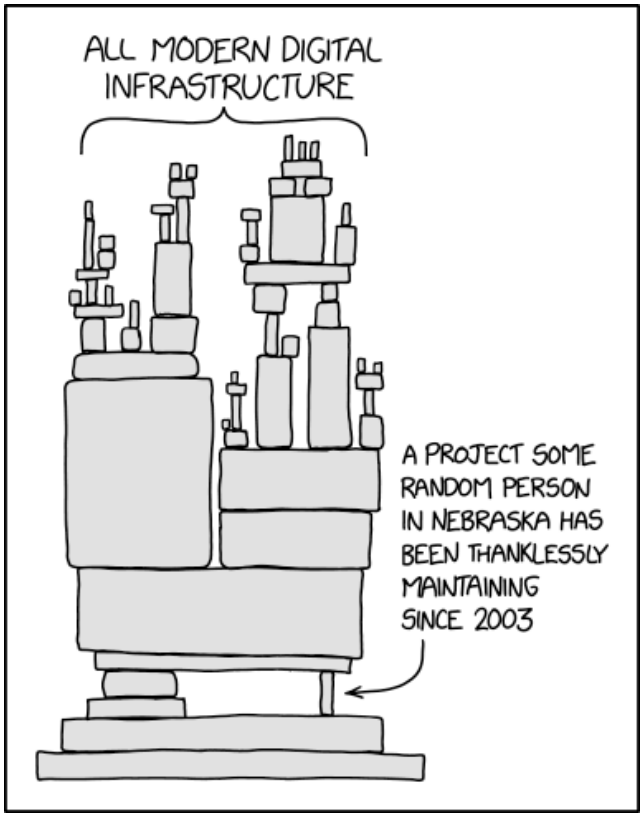
GOLD

PARTNER

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

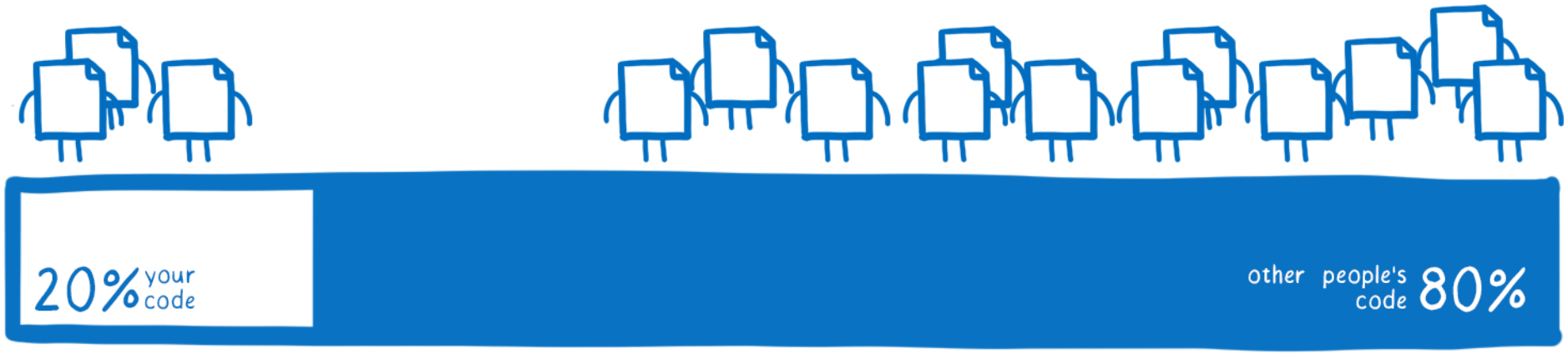
Modern Application Architecture XKCD 2347



Agenda

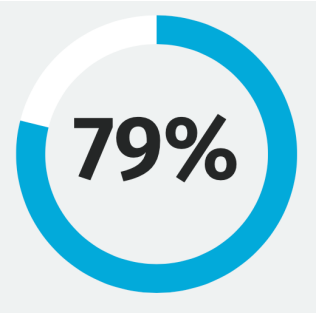
- Risks in 3rd party NuGet Packages
- OpenSFF Scorecard
- Measure, New & Improved
- Conclusion - Q&A

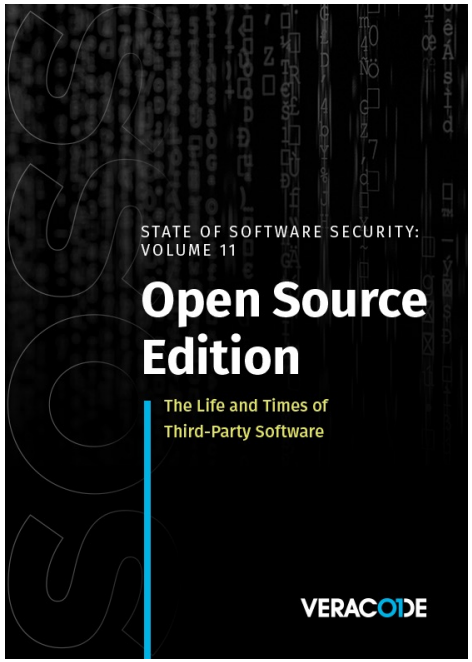
Average codebase composition



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.”

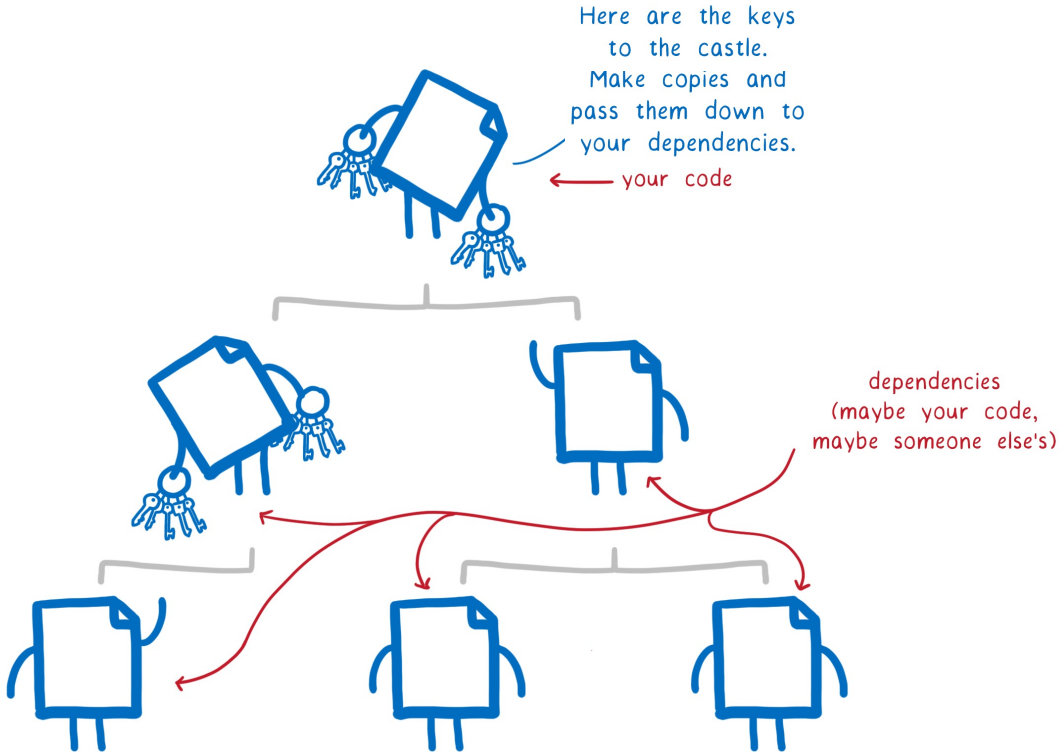




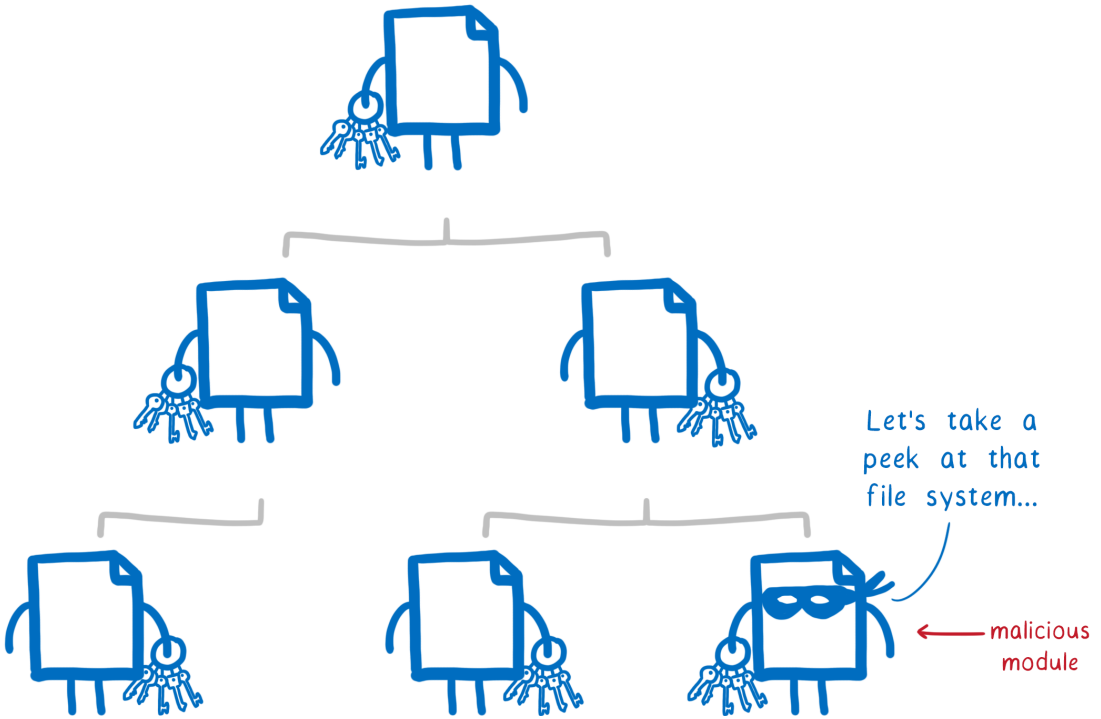
State of Log4j – 2 years later

- 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.

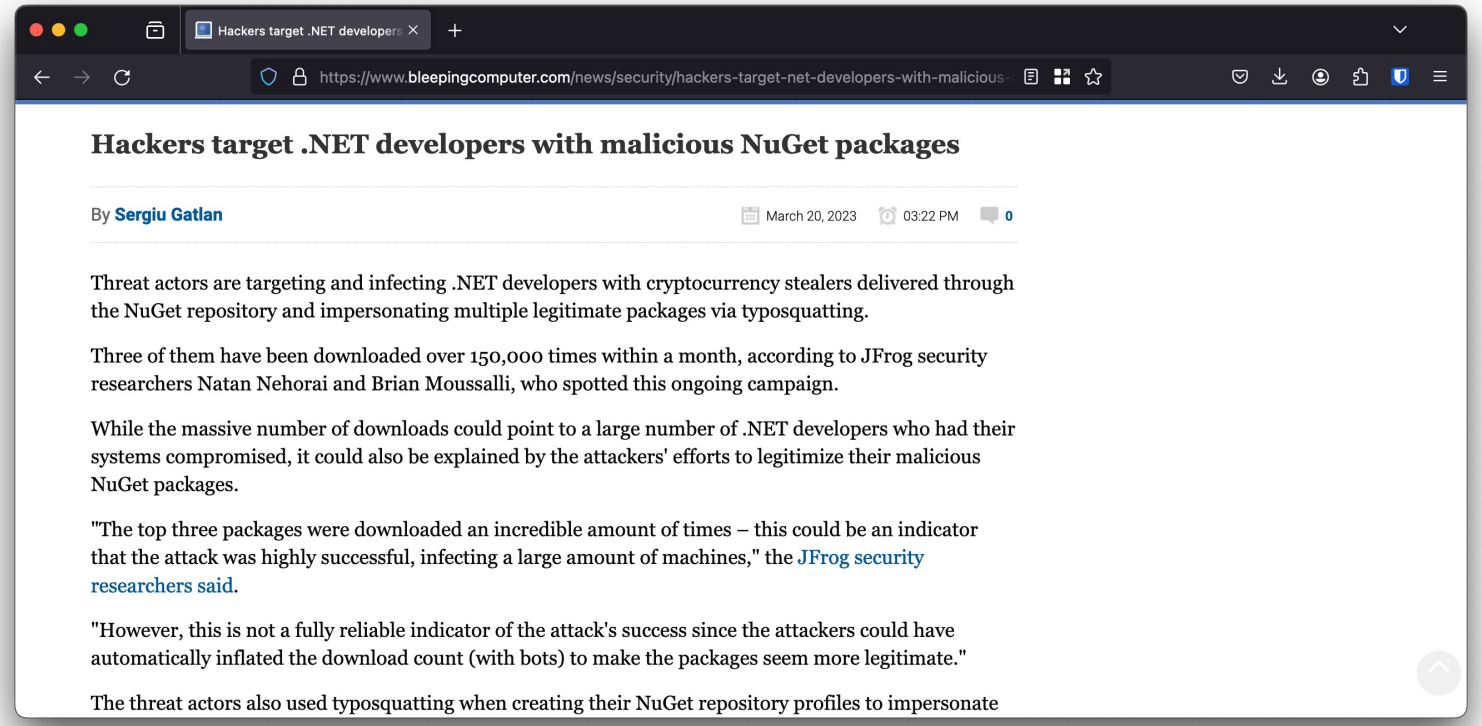
Average codebase composition



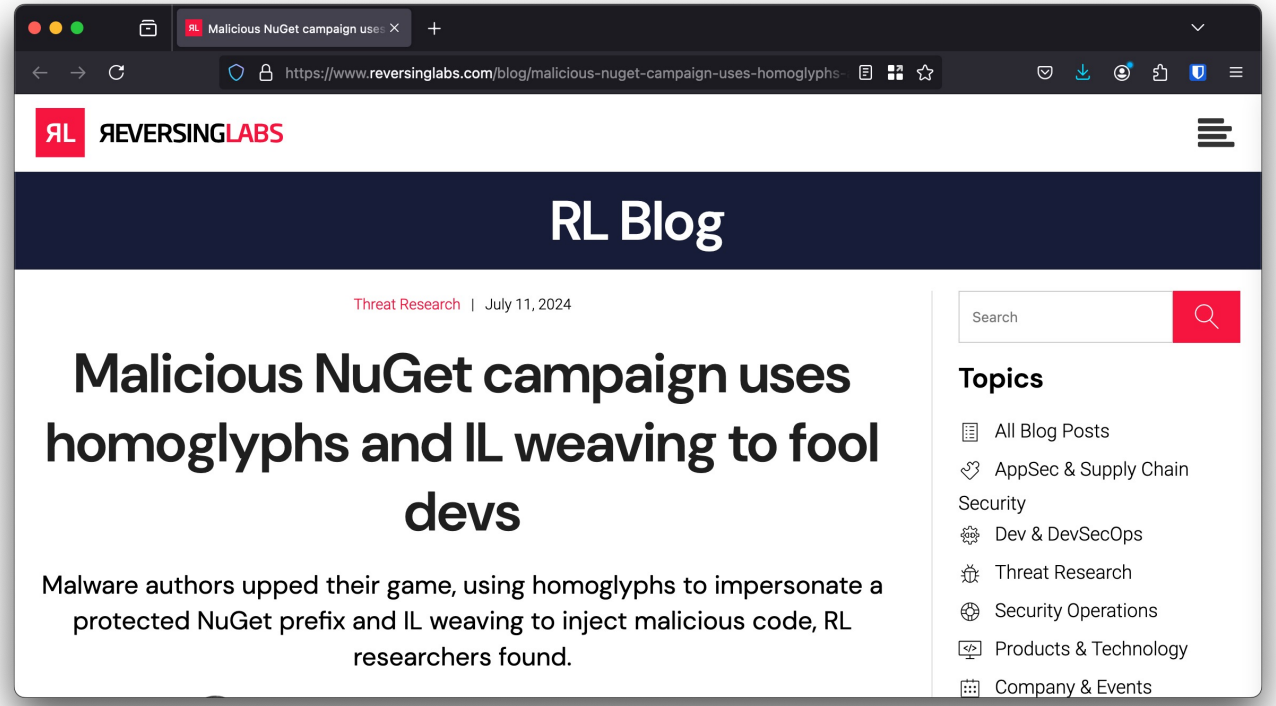
Malicious Assembly



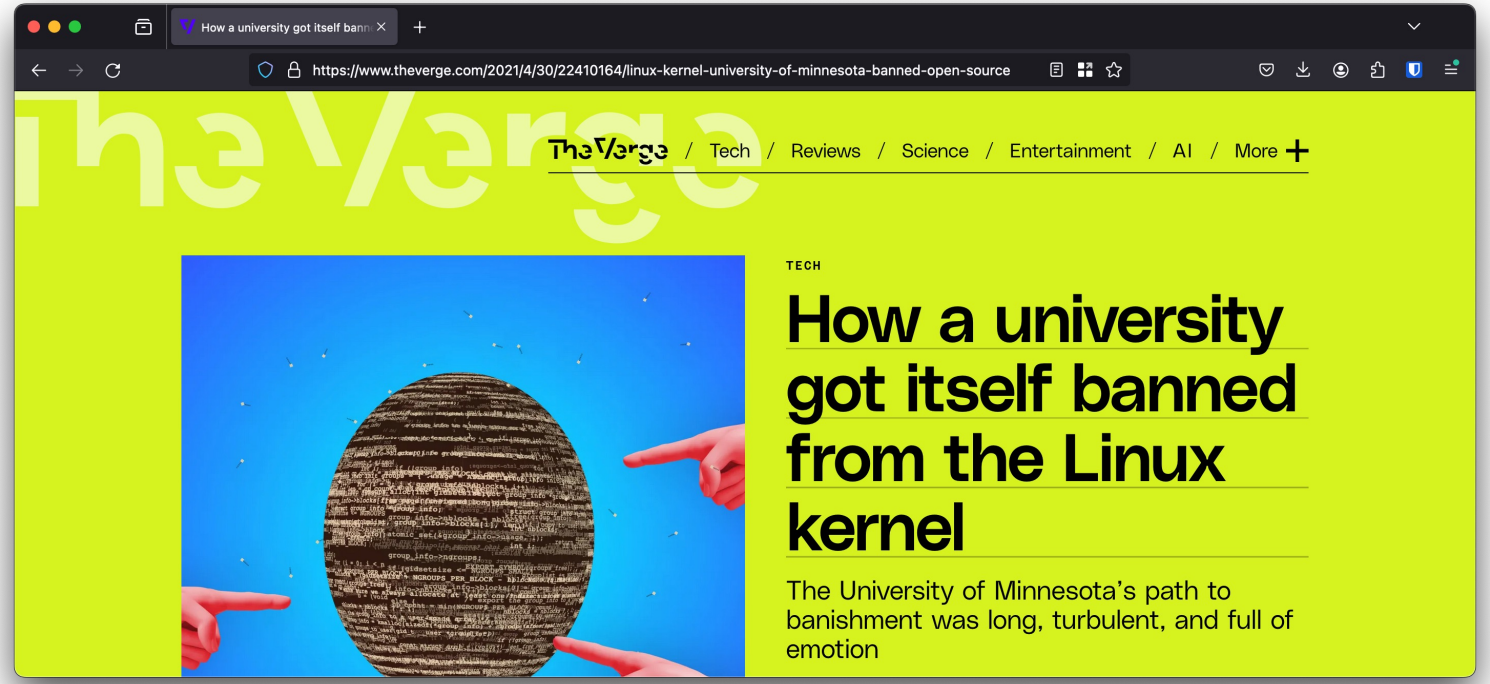
Malicious Package



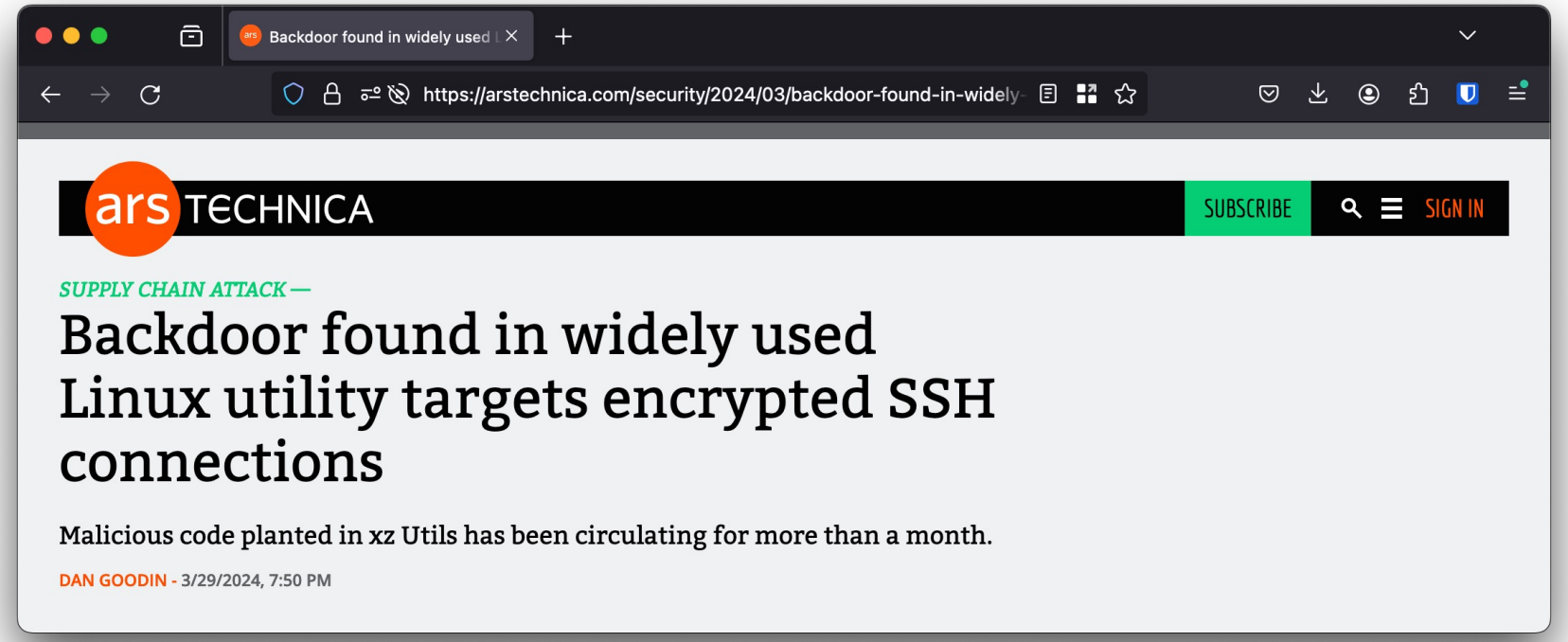
Malicious Package



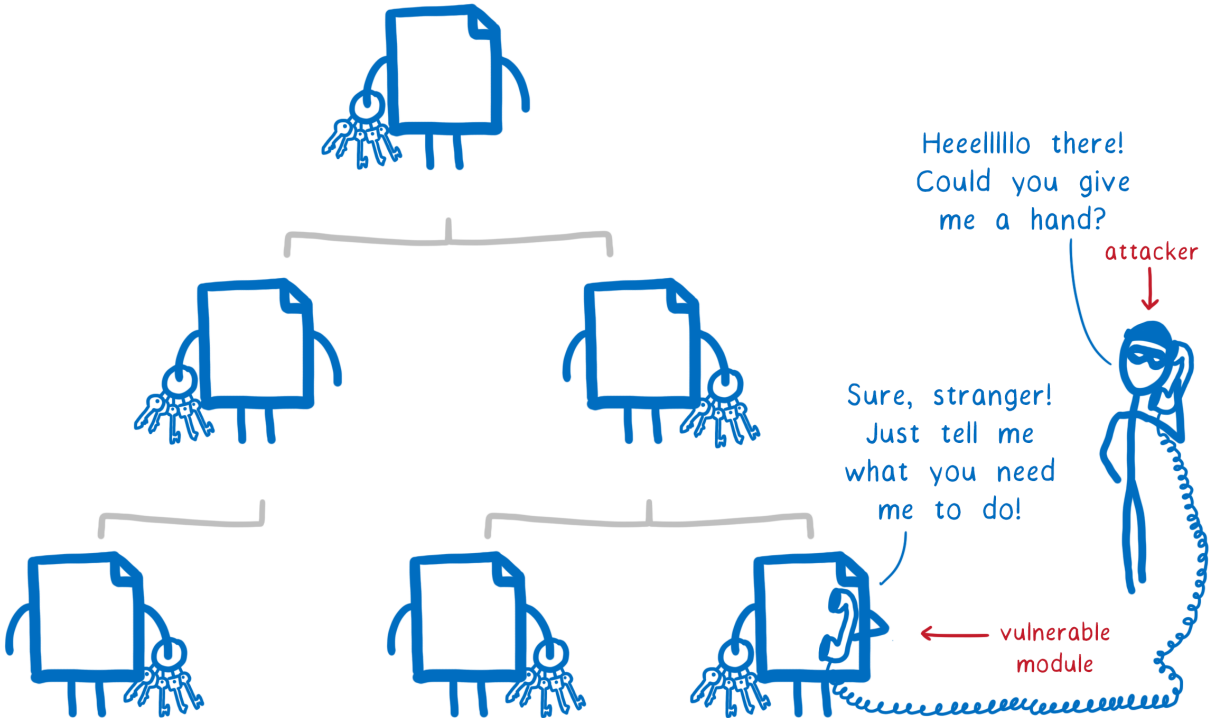
Hypocrite Commits



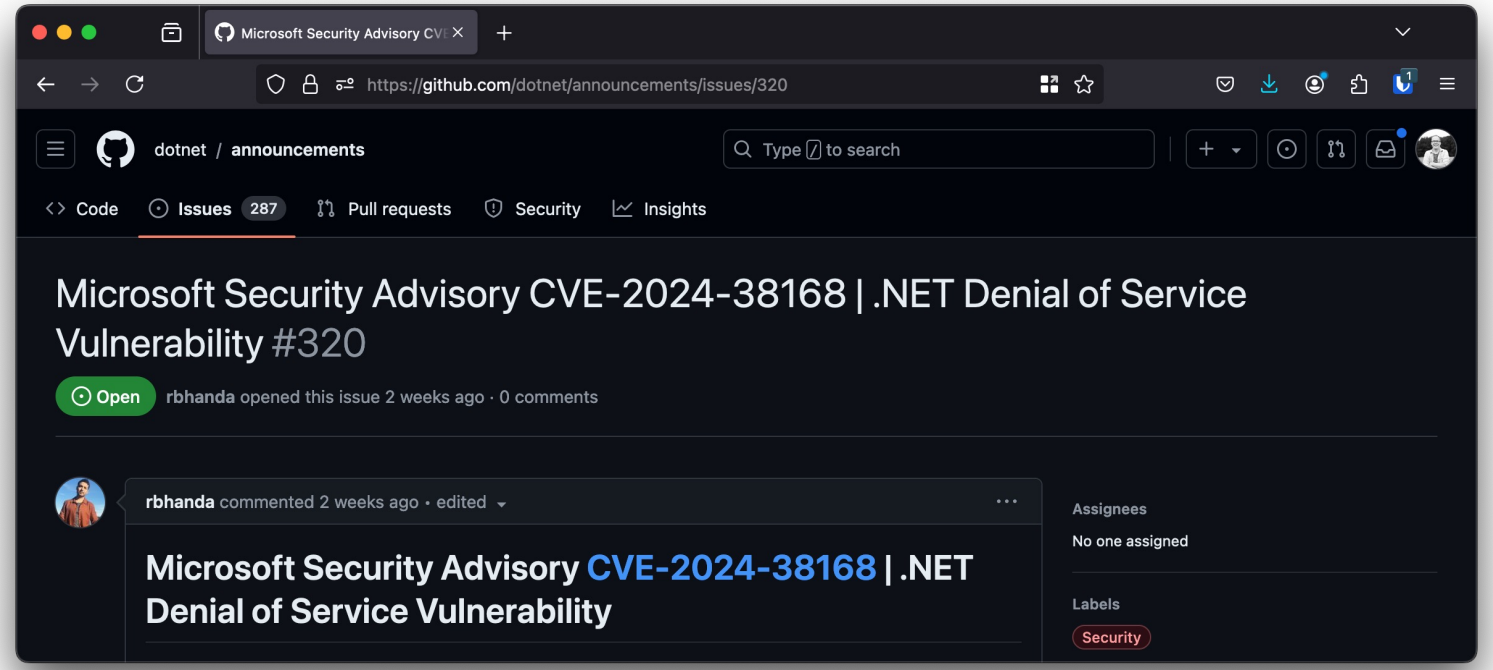
XZ Backdoor


Backdoor found in widely used
https://arstechnica.com/security/2024/03/backdoor-found-in-widely-
ars TECHNICA
SUBSCRIBE
SIGN IN
SUPPLY CHAIN ATTACK —
Backdoor found in widely used Linux utility targets encrypted SSH connections
Malicious code planted in xz Utils has been circulating for more than a month.
DAN GOODIN - 3/29/2024, 7:50 PM

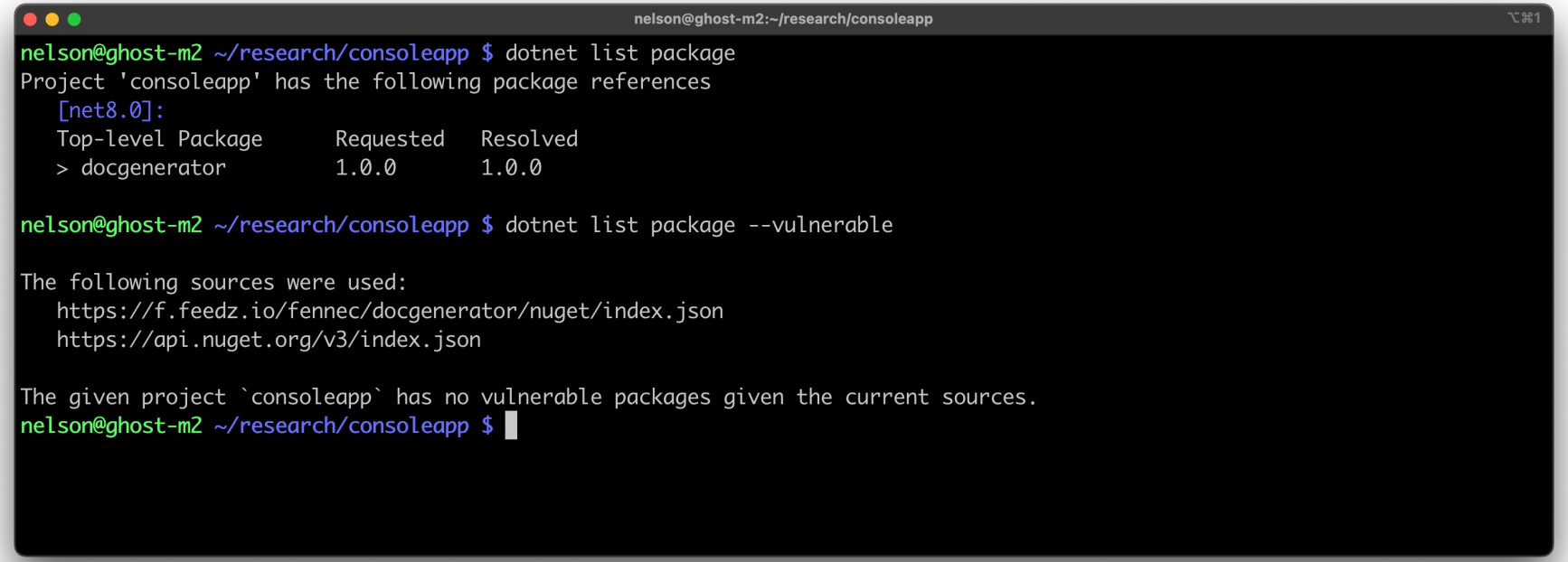
Vulnerable Assembly



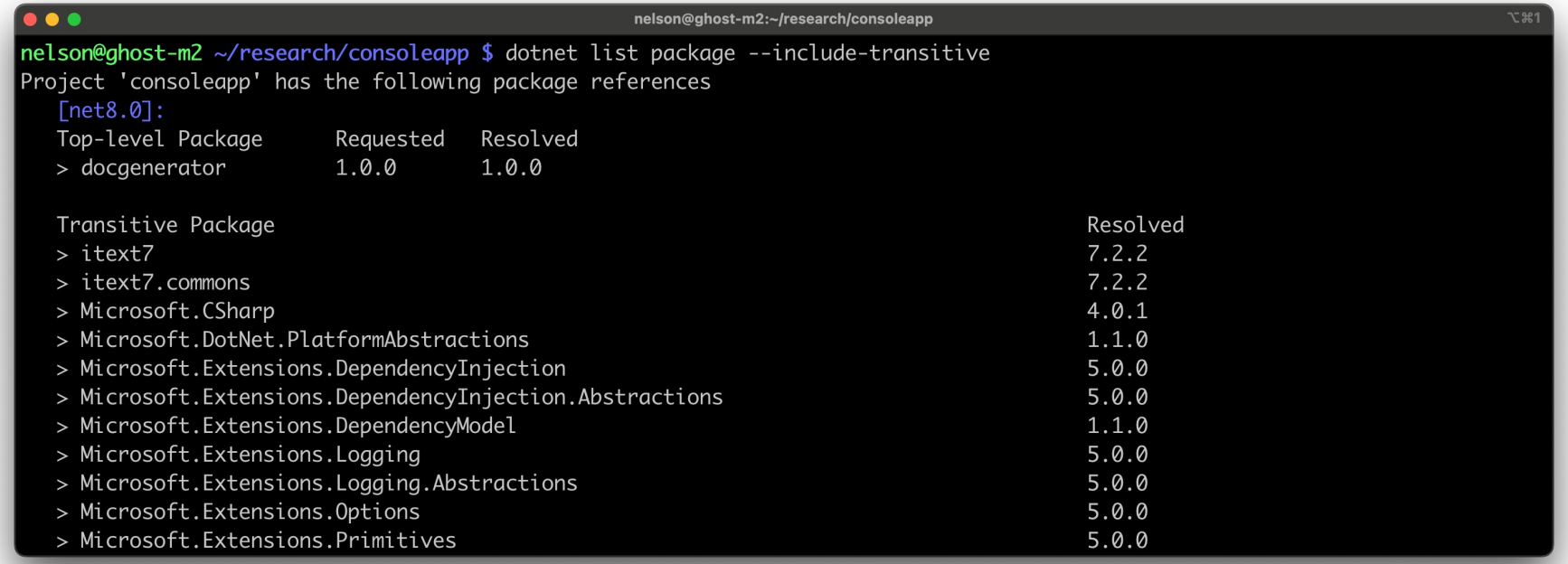
Vulnerabilities in Libraries



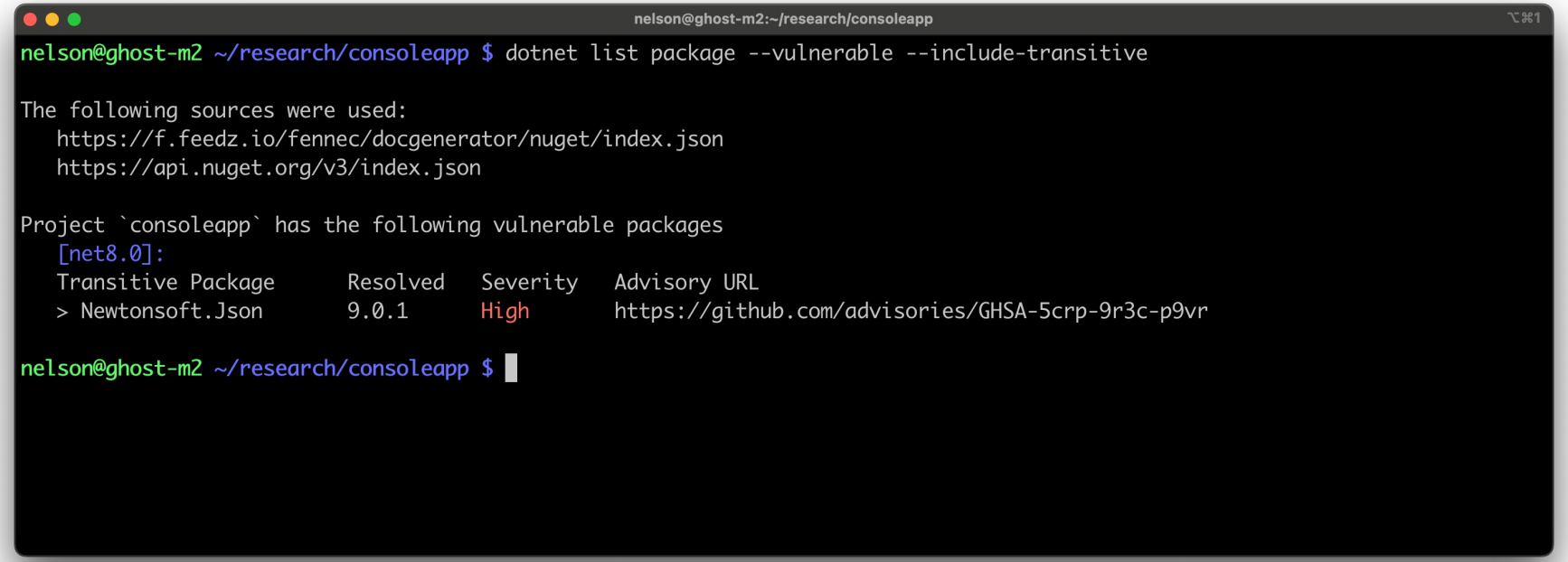
DotNet CLI


nelson@ghost-m2:~/research/consoleapp
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 $

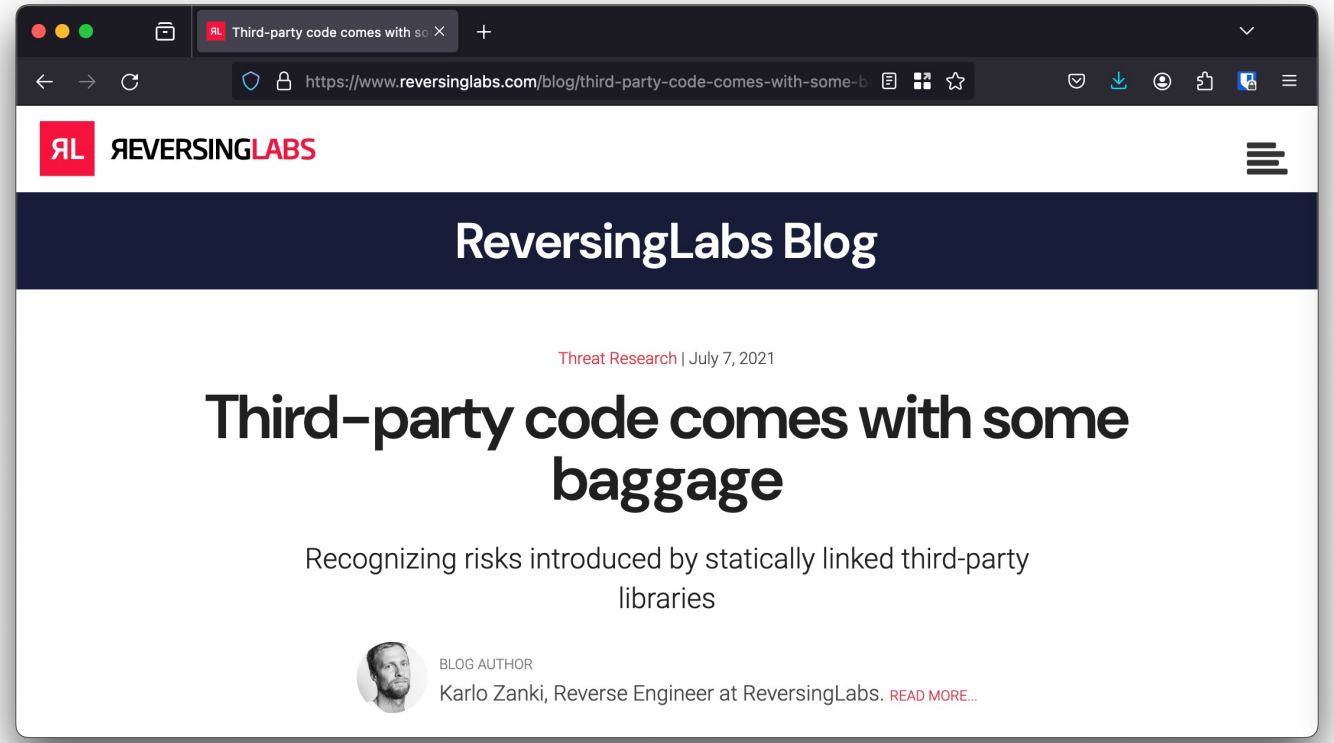
DotNet CLI


nelson@ghost-m2:~/research/consoleapp
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.commons 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

DotNet CLI


nelson@ghost-m2:~/research/consoleapp
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 $

Do you know what's inside?

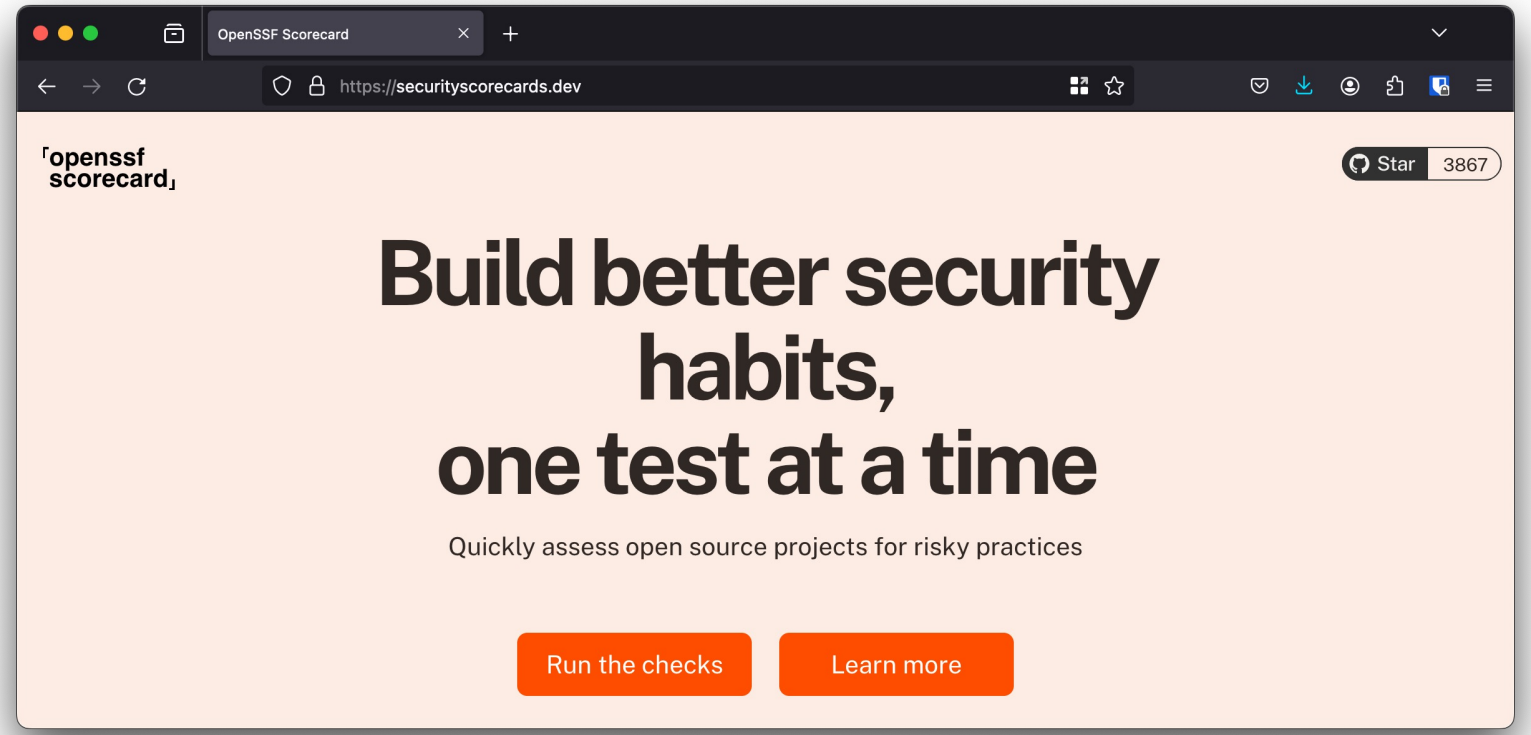


devday

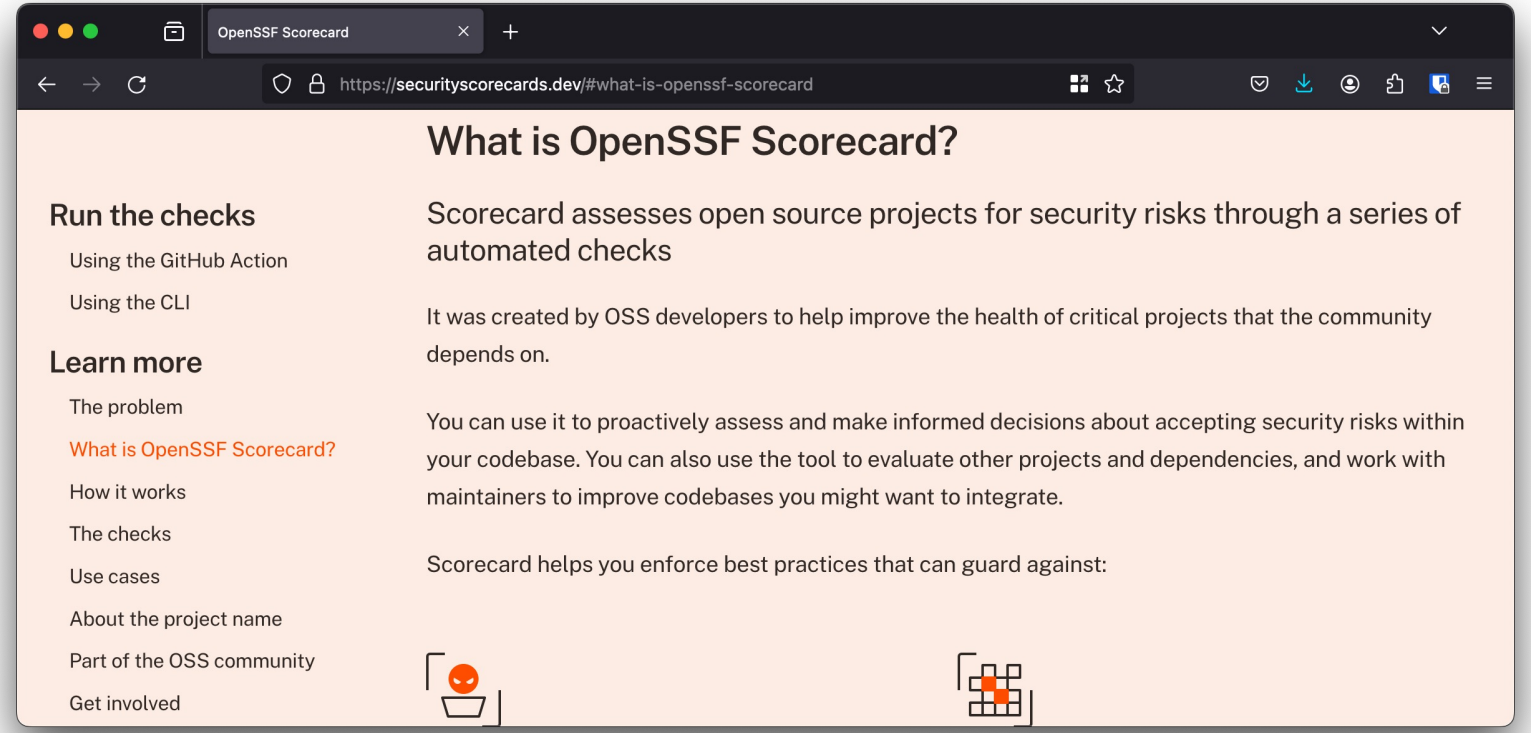
@nielstanis@infosec.exchange

Nutrion Label for Software?

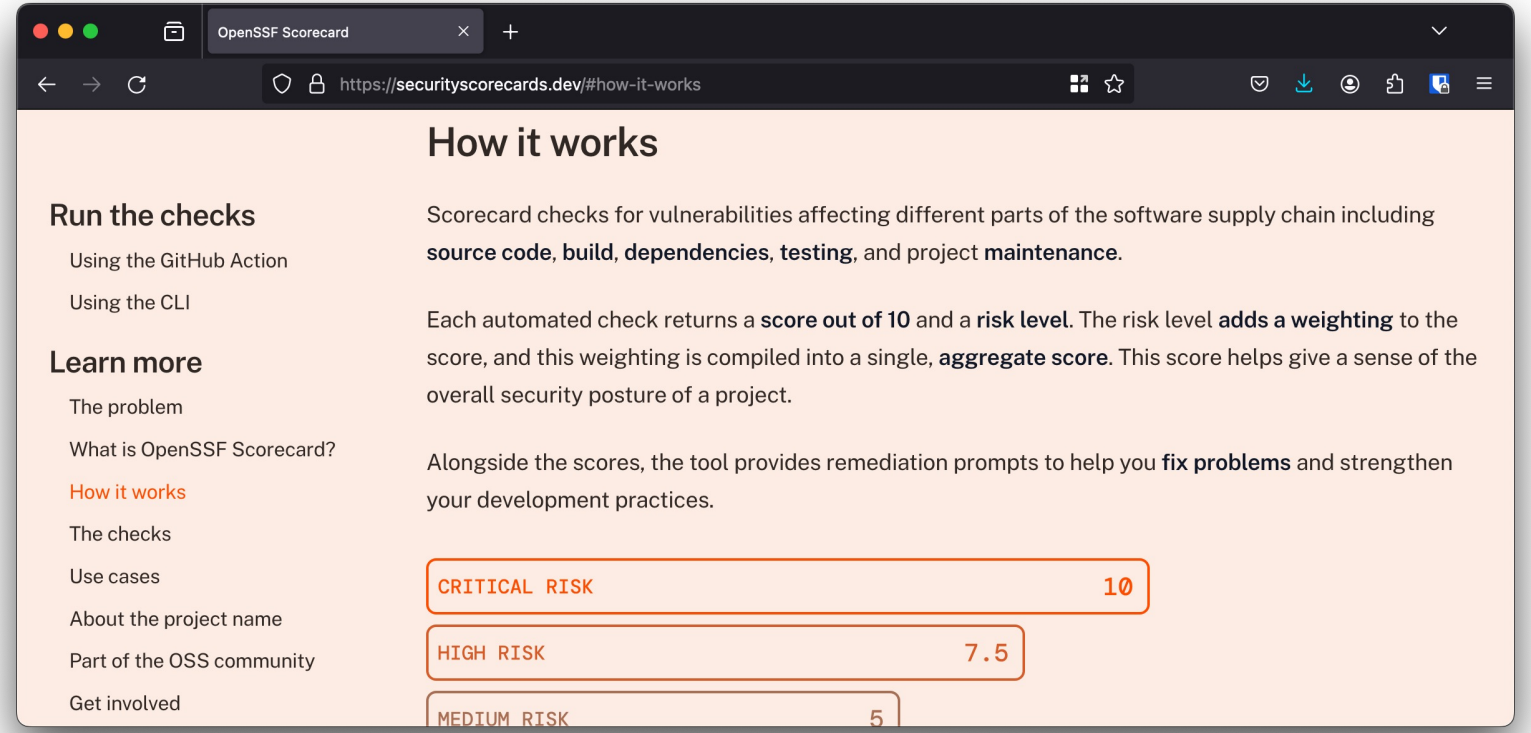
OpenSSF Scorecards



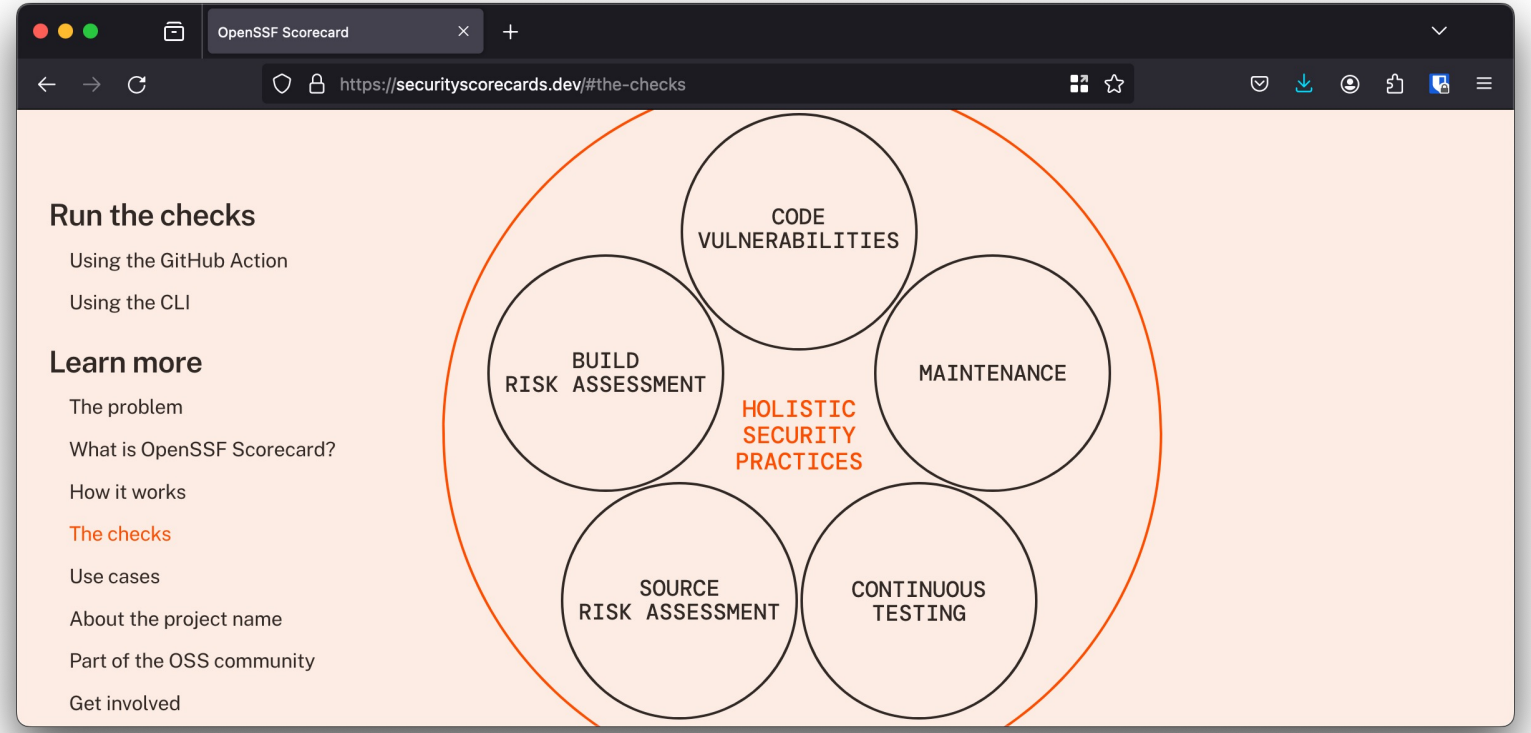
OpenSSF Security Scorecards



OpenSSF Security Scorecards

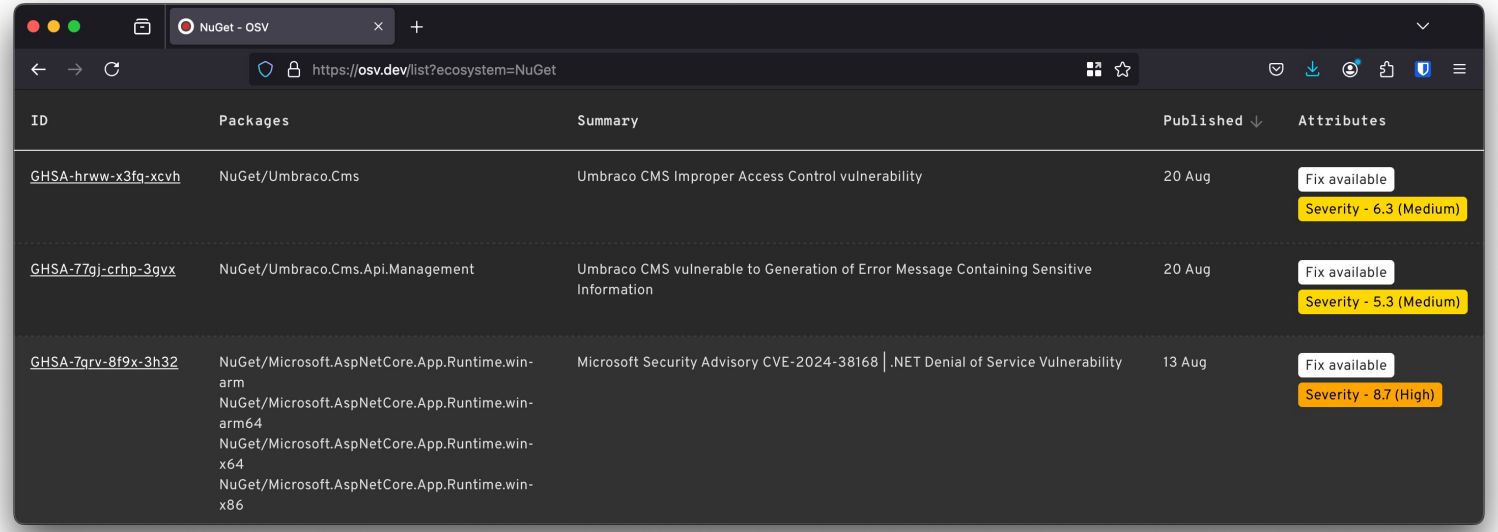


OpenSSF Security Scorecards



Code Vulnerabilities (High)

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



ID	Packages	Summary	Published	Attributes
GHSA-hrww-x3fq-xcvh	NuGet/Umbraco.Cms	Umbraco CMS Improper Access Control vulnerability	20 Aug	Fix available Severity - 6.3 (Medium)
GHSA-77gj-crhp-3gvx	NuGet/Umbraco.Cms.Api.Management	Umbraco CMS vulnerable to Generation of Error Message Containing Sensitive Information	20 Aug	Fix available Severity - 5.3 (Medium)
GHSA-7qrv-8f9x-3h32	NuGet/Microsoft.AspNetCore.App.Runtime.win-arm NuGet/Microsoft.AspNetCore.App.Runtime.win-arm64 NuGet/Microsoft.AspNetCore.App.Runtime.win-x64 NuGet/Microsoft.AspNetCore.App.Runtime.win-x86	Microsoft Security Advisory CVE-2024-38168 | .NET Denial of Service Vulnerability	13 Aug	Fix available Severity - 8.7 (High)

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.

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.

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.

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

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).

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?

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!

Source Risk Assesement
Binary Artifacts (High)

- This check determines whether the project has generated executable (binary) artifacts in the source repository.
- Binary artifacts cannot be reviewed, allowing possible obsolete or maliciously subverted executables.
- There is need for **reproducible** builds!

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!

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/

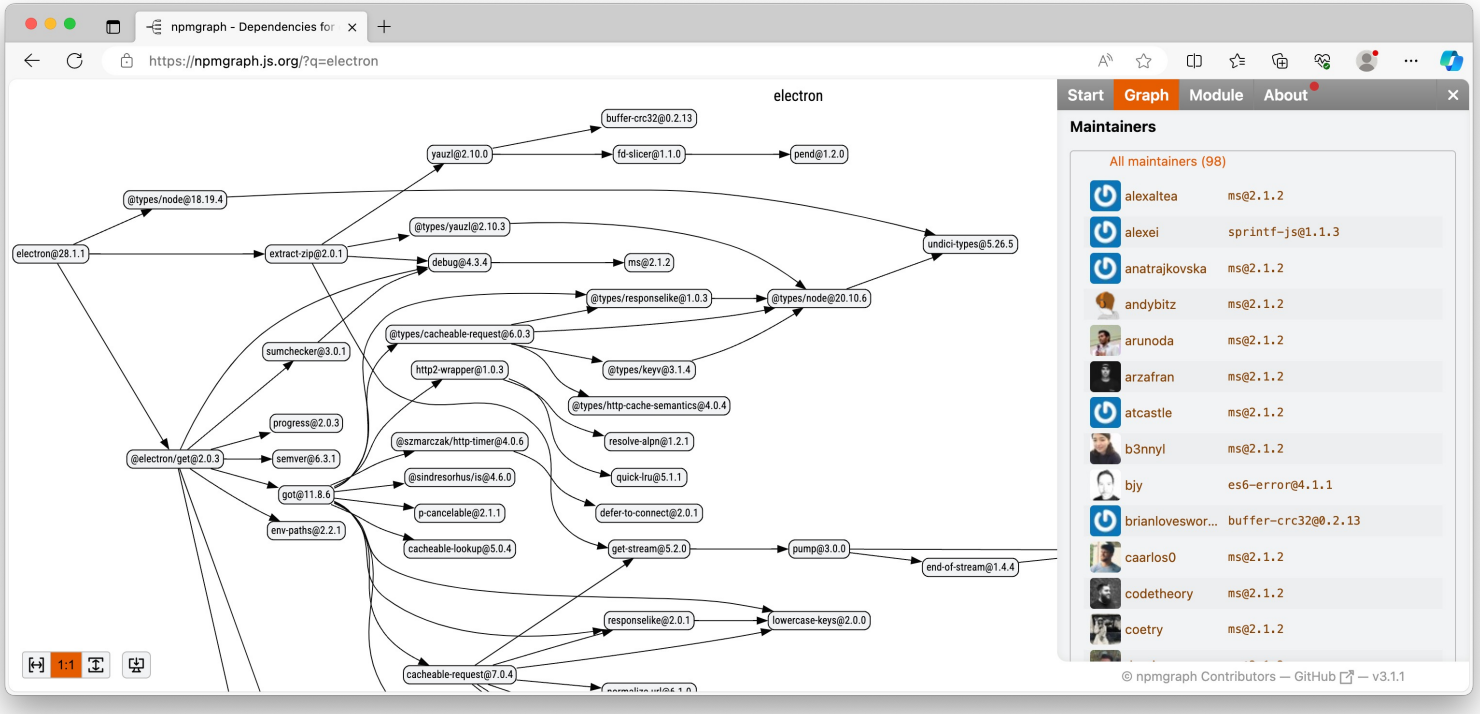
Source Risk Assesement
Code Review (Low)

- 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)

Source Risk Assesement 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?

Source Risk Assesement Contributors (Low)



devday GET INSPIRED

@nielstanis@infosec.exchange

Build Risk Assesement
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?

Build Risk Assesement
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.

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.

Build Risk Assessment
Signed Releases (High)

- This check tries to determine if the project cryptographically signs release artifacts.
  - Signed release packages
  - Signed build provenance

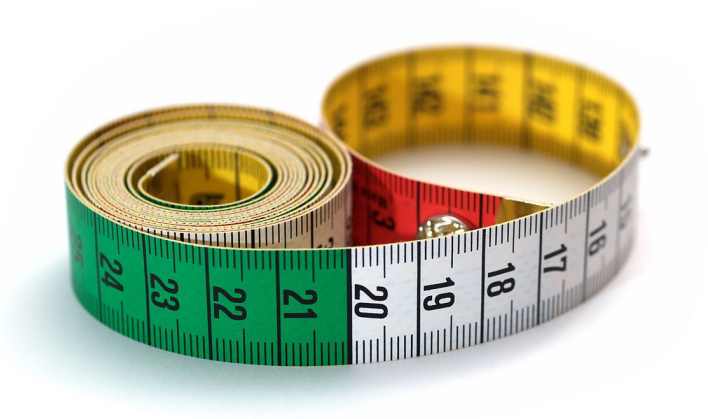
Demo OpenSSF Scorecard
Fennec CLI

```
Running checks
```

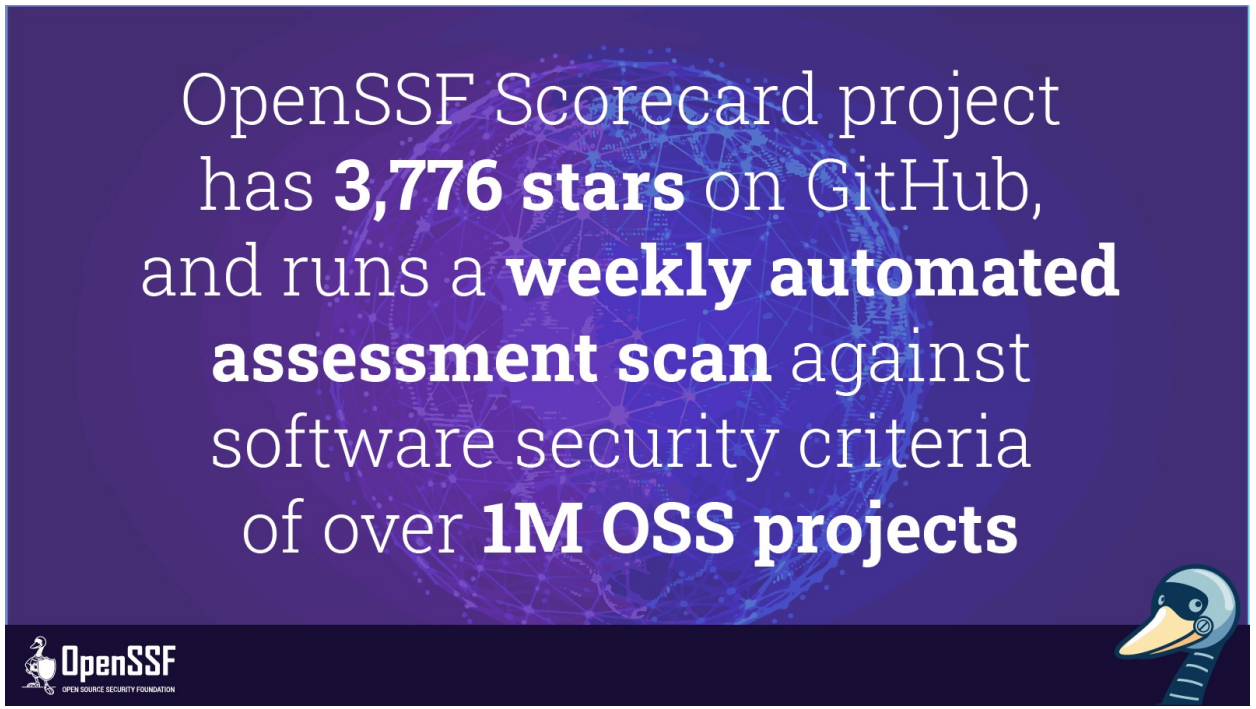
devday
GET INSPIRED

@nielstanis@infosec.exchange

Measure?

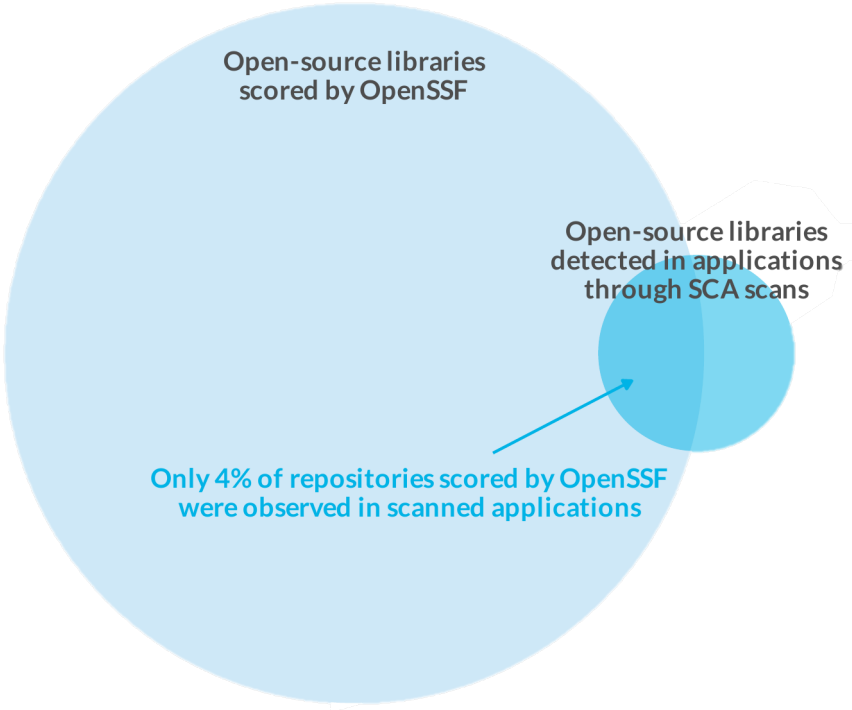


OpenSSF Annual Report 2023

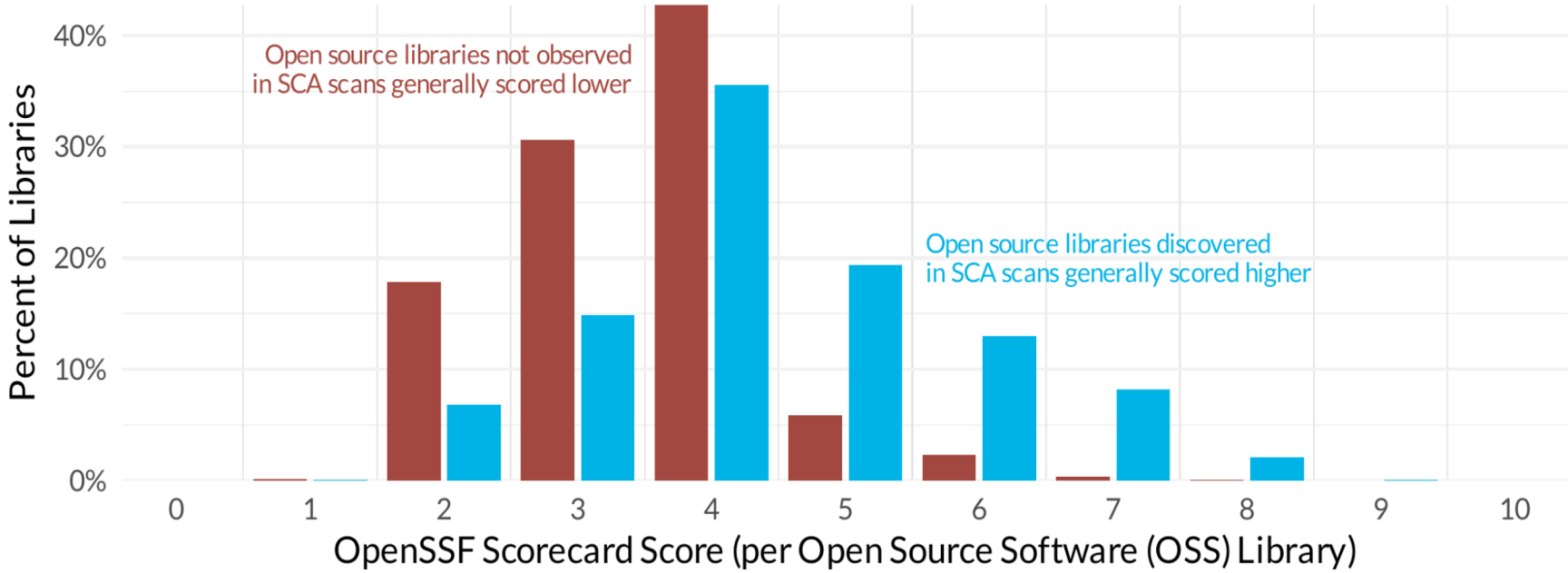


SOSS & OpenSSF Scorecard

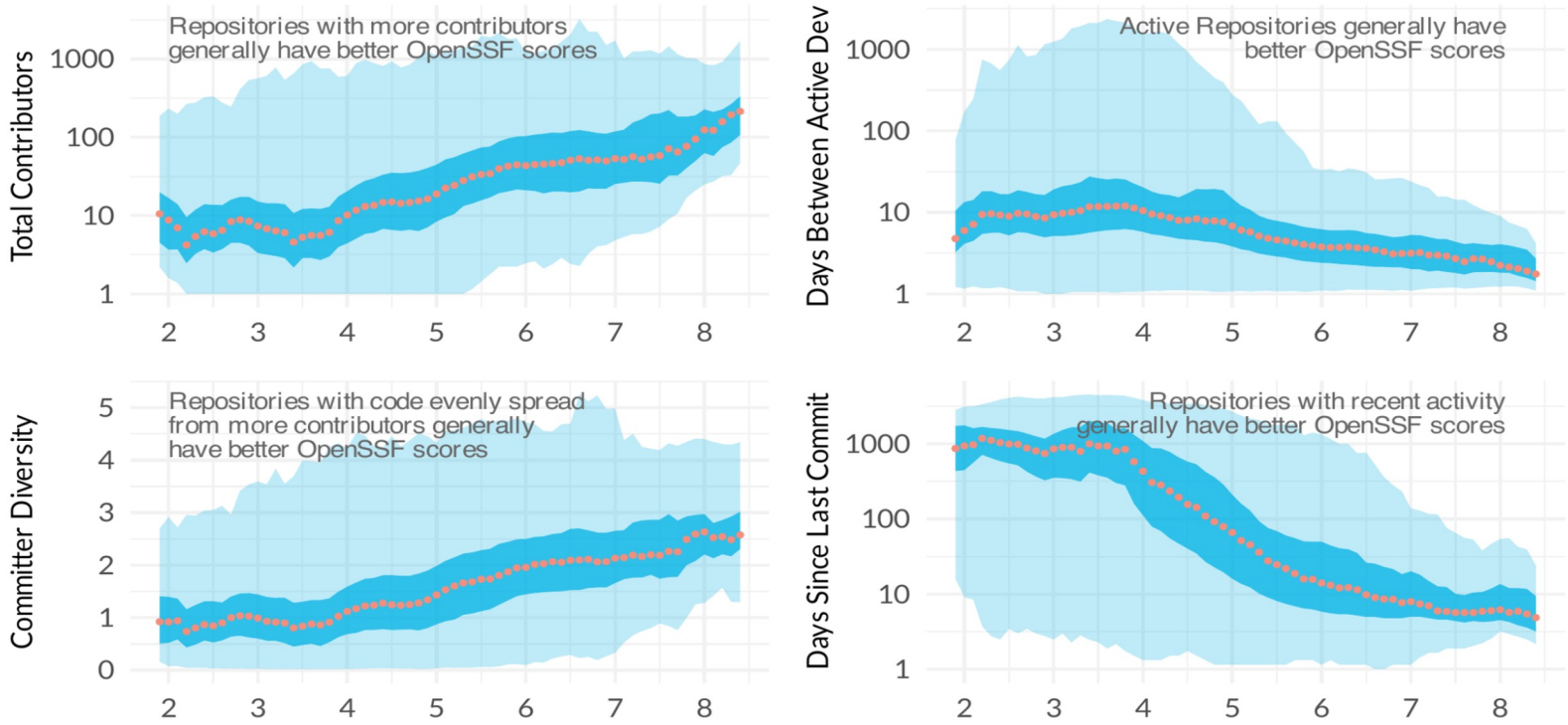
SOSS & OpenSSF Scorecard



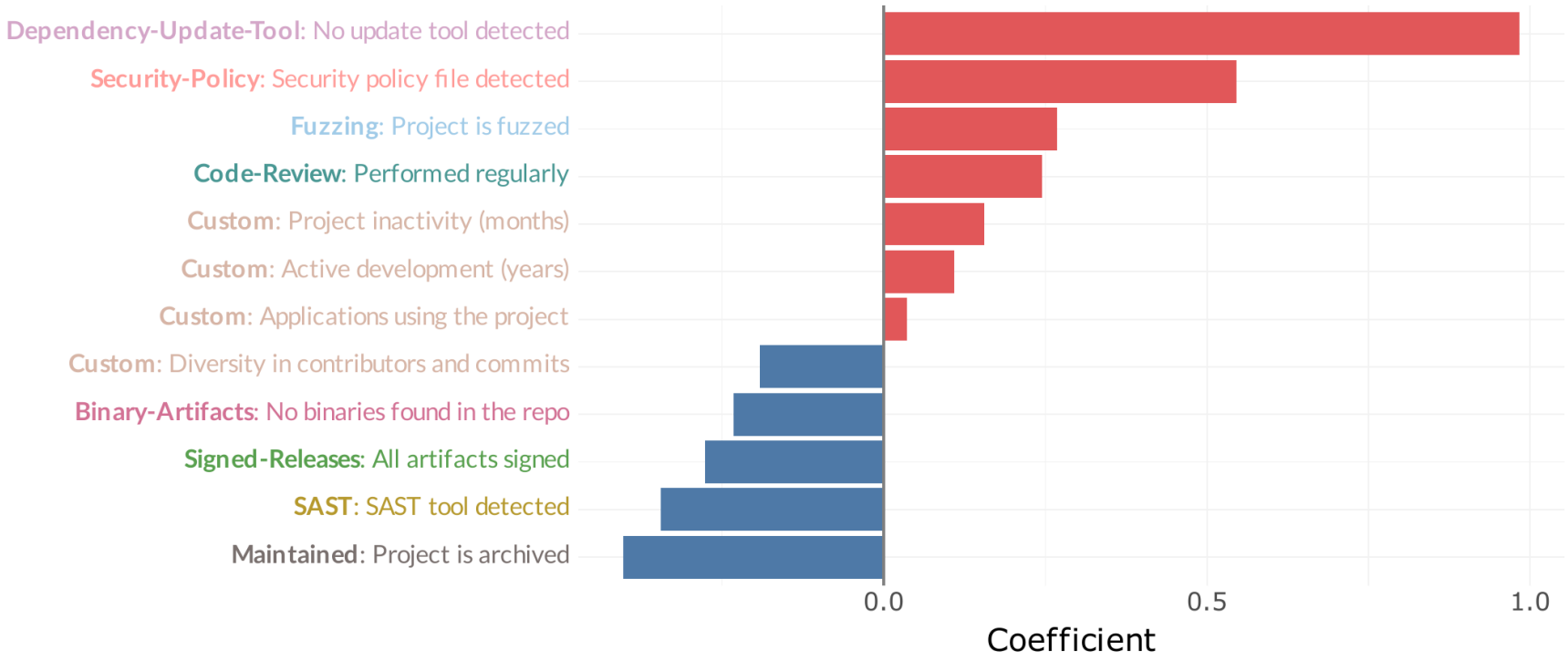
Correlation between SOSS



Github commits vs OpenSSF



What really contributes to OSS Security?



devday

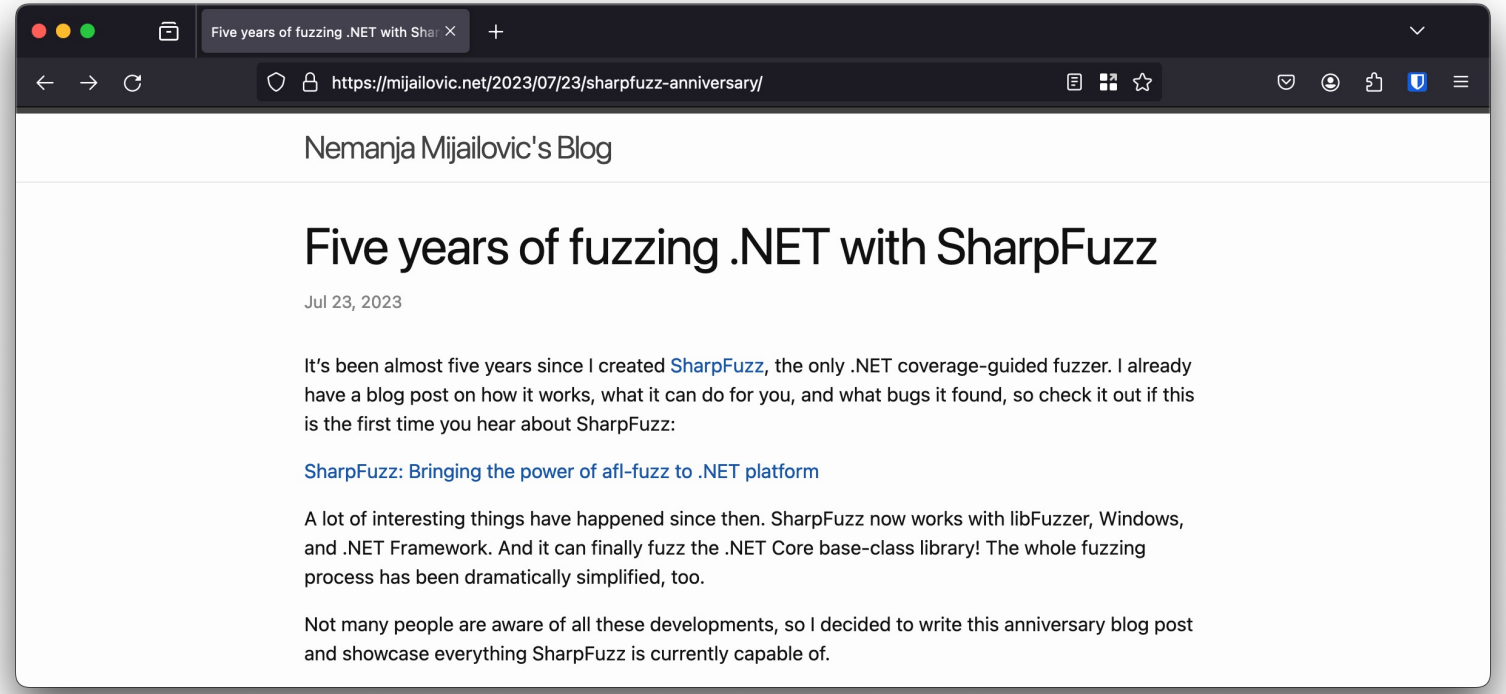
@nielstanis@infosec.exchange

What can we improve?

Fuzzing .NET

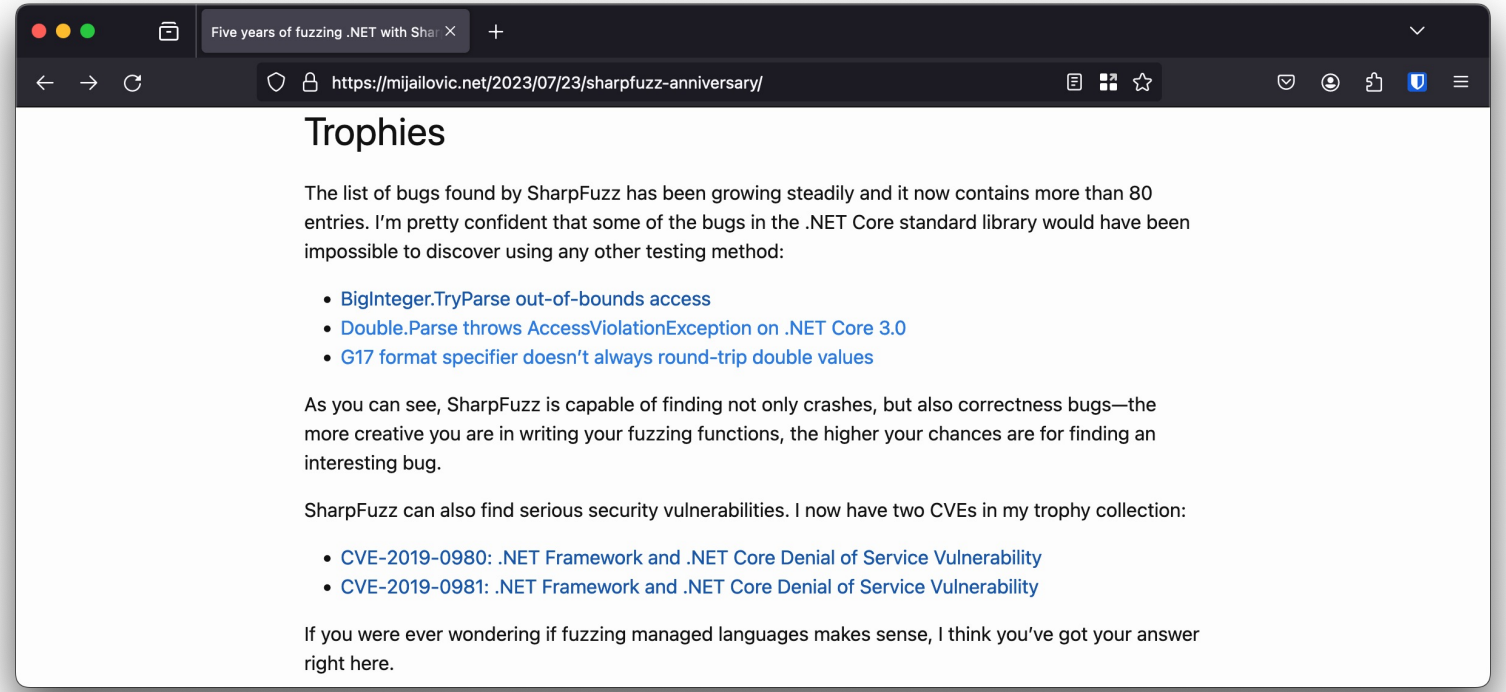
- Fuzzing, or fuzz testing
  - Automated software testing method that uses a wide range of ***invalid*** and unexpected data as input to find flaws
- Definitely good for finding C/C++ memory issues
- Can it be of any value with managed languages like .NET?

Fuzzing .NET & SharpFuzz



New & Improved!

Fuzzing .NET & SharpFuzz



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:

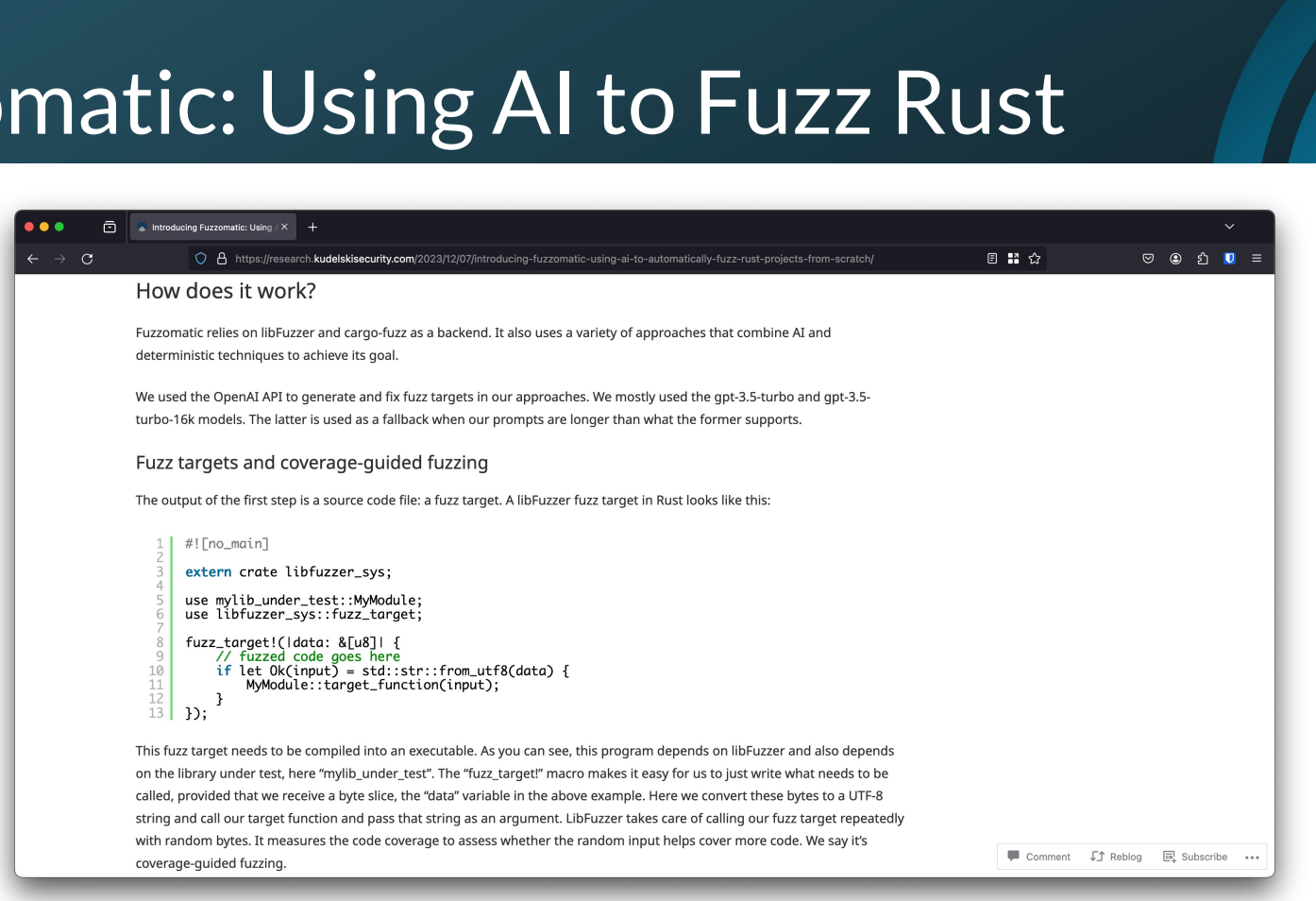
- CVE-2019-0980: .NET Framework and .NET Core Denial of Service Vulnerability
- CVE-2019-0981: .NET Framework and .NET Core Denial of Service Vulnerability

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) { }
        });
}
```

Fuzzomatic: Using AI to Fuzz Rust



Introducing Fuzzomatic: Using

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

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:

```
#![no_main]

extern crate libfuzzer_sys;

use mylib_under_test::MyModule;
use libfuzzer_sys::fuzz_target;

fuzz_target!(|data: &[u8]| {
    // fuzzed code goes here
    if let Ok(input) = std::str::from_utf8(data) {
        MyModule::target_function(input);
    }
});
```

This fuzz target needs to be compiled into an executable. As you can see, this program depends on libFuzzer and also depends on the library under test, here “mylib_under_test”. The “fuzz_target!” macro makes it easy for us to just write what needs to be called, provided that we receive a byte slice, the “data” variable in the above example. Here we convert these bytes to a UTF-8 string and call our target function and pass that string as an argument. LibFuzzer takes care of calling our fuzz target repeatedly with random bytes. It measures the code coverage to assess whether the random input helps cover more code. We say it’s coverage-guided fuzzing.

Comment Reblog Subscribe

devday GET INSPIRED

@nielstanis@infosec.exchange

Static Code Analysis (SAST)


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

Static Code Analysis (SAST)

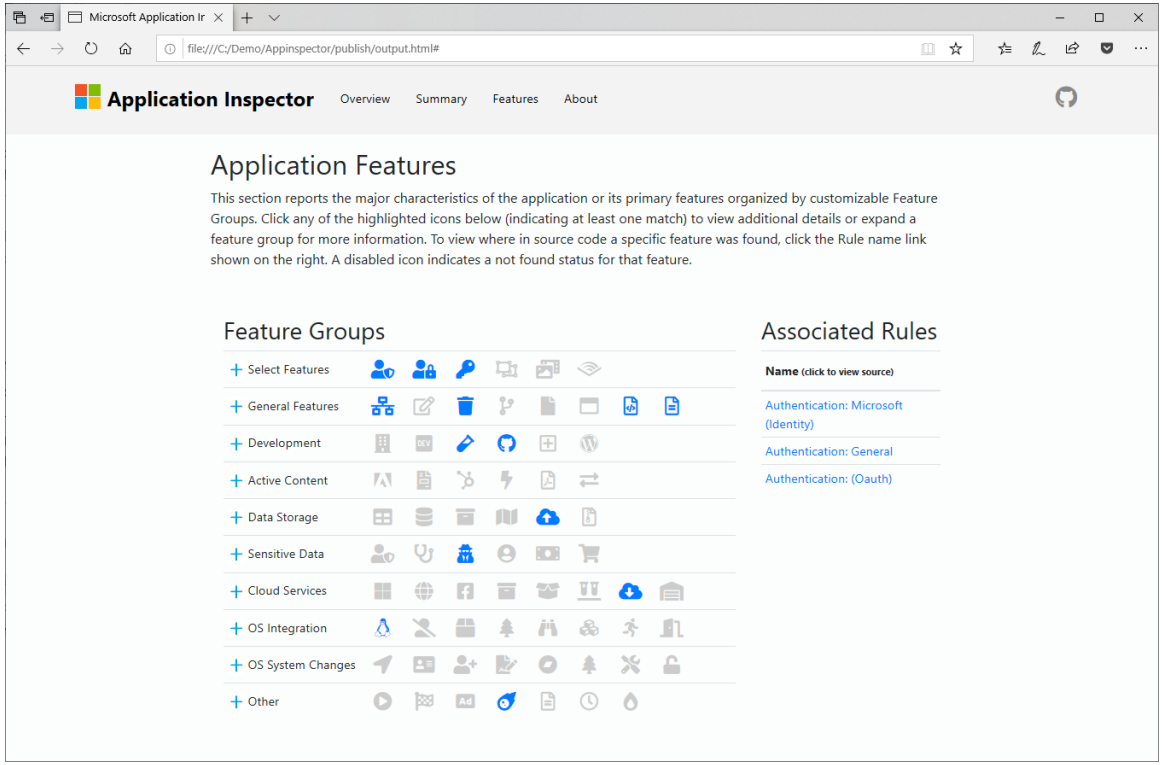

```
public class CustomerController : Controller
{
    public IActionResult GenerateCustomerReport(string customerID)
    {
        var data = Reporting.GenerateCustomerReportOverview(customerID)
        return View(data);
    }
}
```

```
public static class Reporting
{
    public static byte[] GenerateCustomerReportOverview(string ID)
    {
        return System.IO.File.ReadAllBytes($"./data/{ID}.pdf");
    }
}
```

.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)

Application Inspector



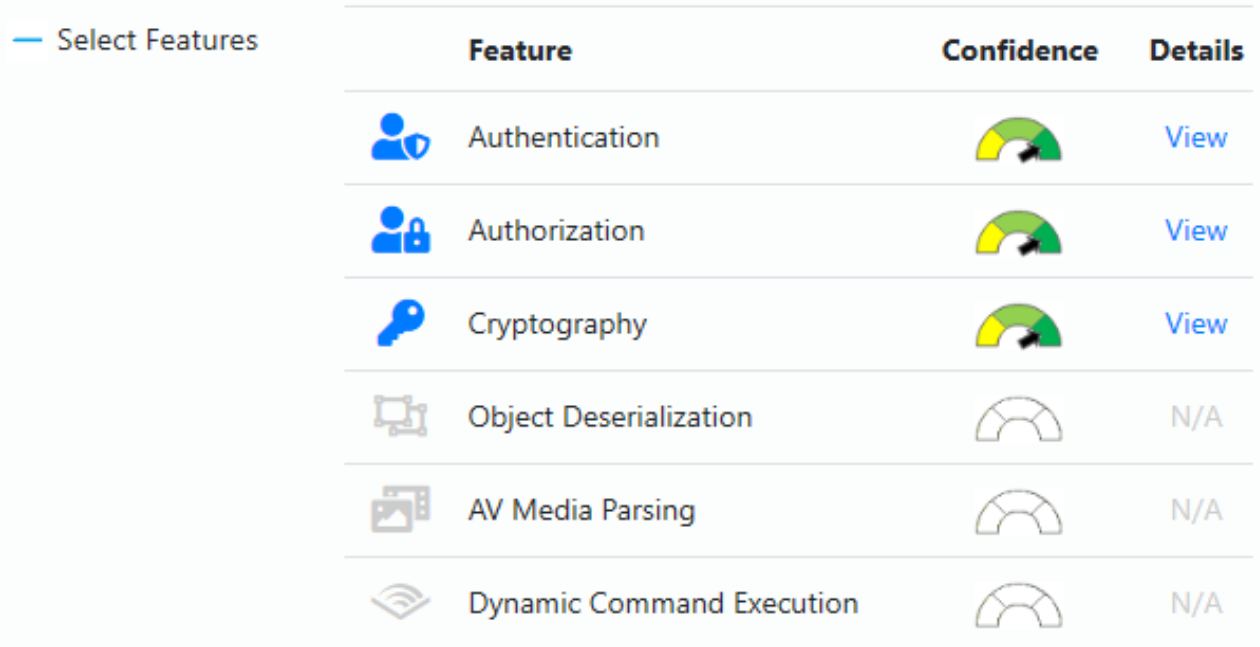
New & Improved!

devday GET INSPIRED

@nielstanis@infosec.exchange

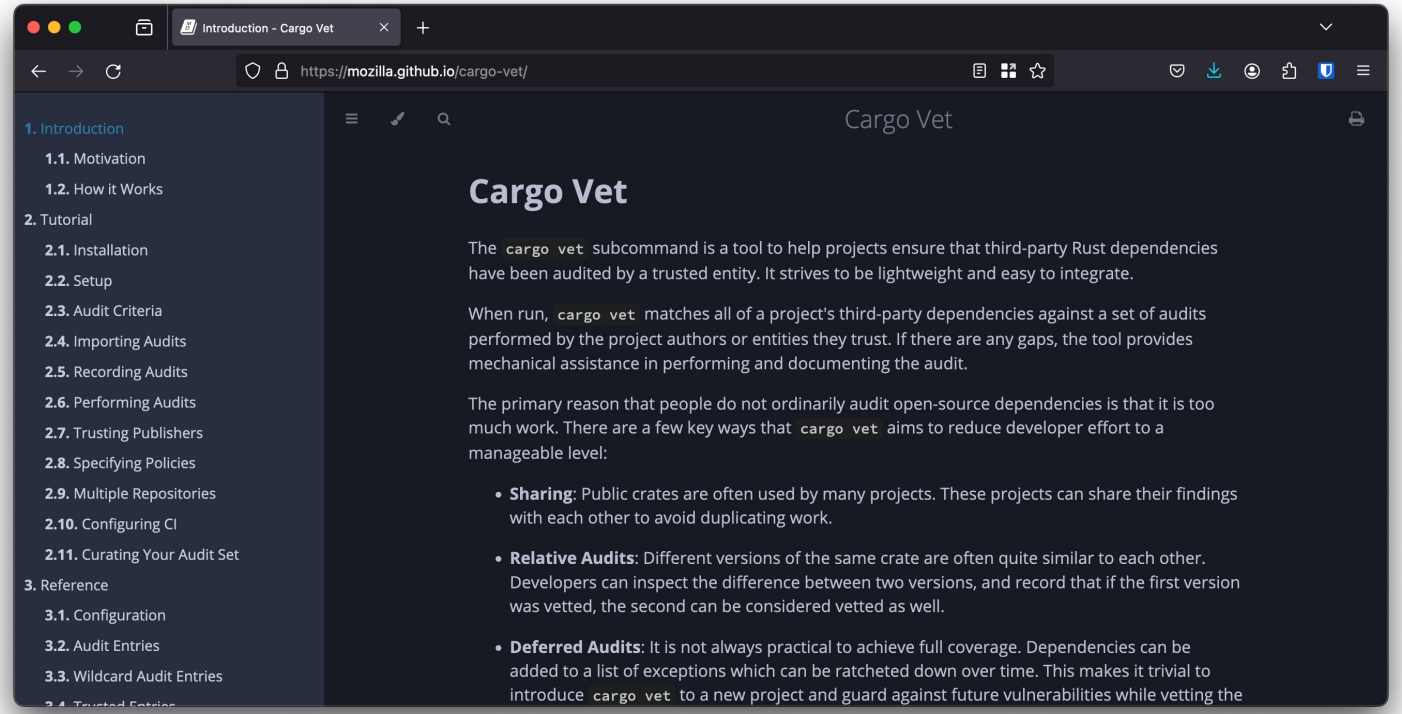
Application Inspector

New & Improved!

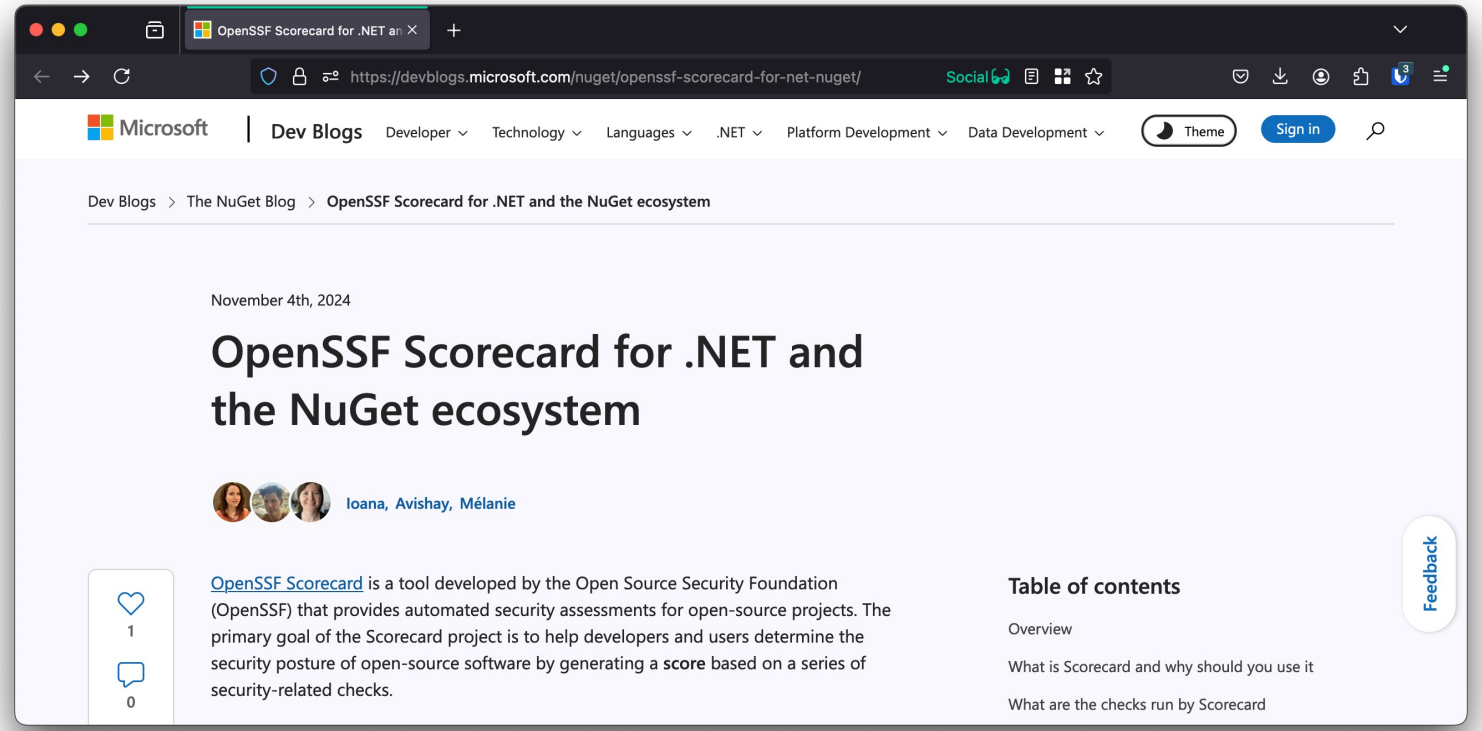


Community Review

New & Improved!



NuGet Blog


OpenSSF Scorecard for .NET ar
https://devblogs.microsoft.com/nuget/openssf-scorecard-for-net-nuget/
Social
Microsoft
Dev Blogs
Developer
Technology
Languages
.NET
Platform Development
Data Development
Theme
Sign in
Dev Blogs > The NuGet Blog > OpenSSF Scorecard for .NET and the NuGet ecosystem
November 4th, 2024
OpenSSF Scorecard for .NET and the NuGet ecosystem
Ioana, Avishay, Mélanie
OpenSSF Scorecard is a tool developed by the Open Source Security Foundation (OpenSSF) that provides automated security assessments for open-source projects. The primary goal of the Scorecard project is to help developers and users determine the security posture of open-source software by generating a score based on a series of security-related checks.
1
0
Table of contents
Overview
What is Scorecard and why should you use it
What are the checks run by Scorecard
Feedback

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!

Conclusion

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


```
dotnet tool install -g fennec
```

Questions?

Links & Feedback

- https://github.com/nielstanis/devday2024
- ntanis at Veracode.com
- @nielstanis@infosec.exchange
- https://www.fennec.dev
  https://blog.fennec.dev