VERACODE

Reviewing NuGet Packages security easily using OpenSSF Scorecard

Niels Tanis
Sr. Principal Security Researcher

NDC { Security }

Who am I?

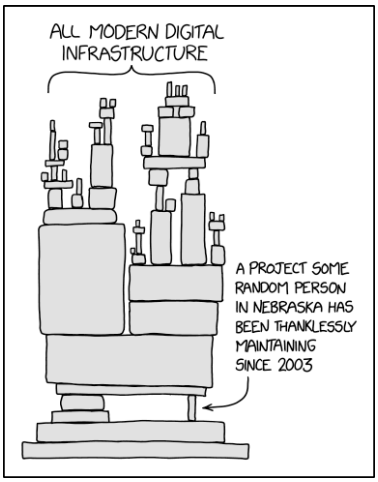
- Niels Tanis
- Sr. Principal Security Researcher
  - Background .NET Development, Pentesting/ethical hacking, and software security consultancy
  - Research on static analysis for .NET apps
  - Enjoying Rust!
- Microsoft MVP – Developer Technologies

VERACODE

MVP Microsoft® Most Valuable Professional

@nielstanis@infosec.exchange

Modern Application Architecture
XKCD 2347



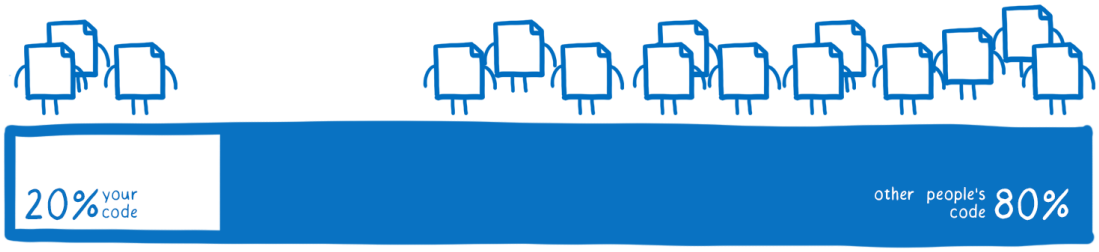
@nielstanis@infosec.exchange

https://xkcd.com/2347/

Agenda

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

Average codebase composition

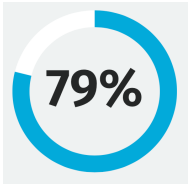


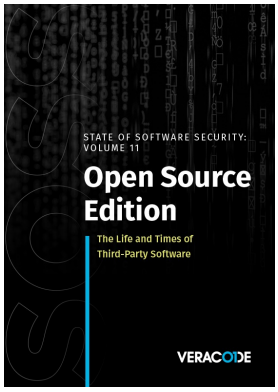
@nielstanis@infosec.exchange

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.”



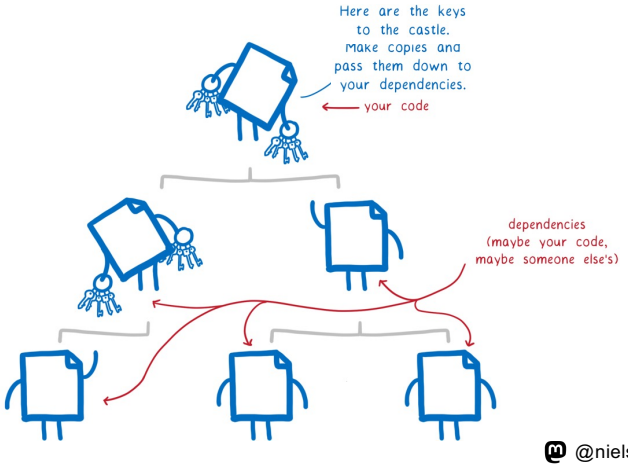


State of Log4j – 2 years later

- 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.

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

Average codebase composition

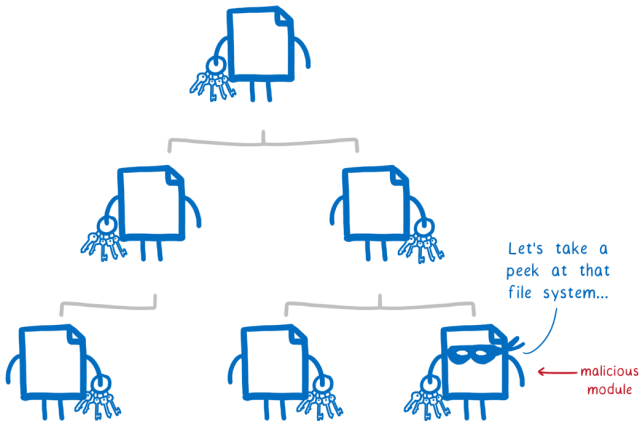


NDC { Security }

@nielstanis@infosec.exchange

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

Malicious Assemby

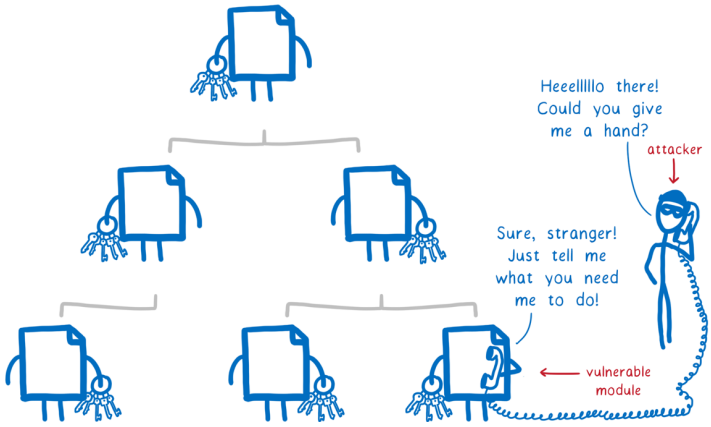


NDC { Security }

@nielstanis@infosec.exchange

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

Vulnerable Assemby

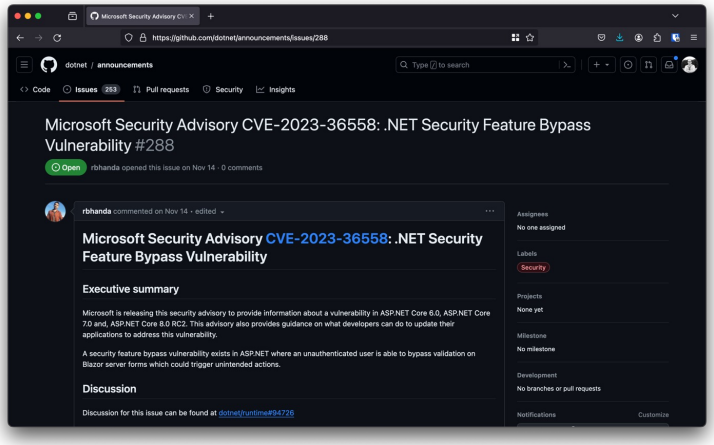


NDC { Security }

@nielstanis@infosec.exchange

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

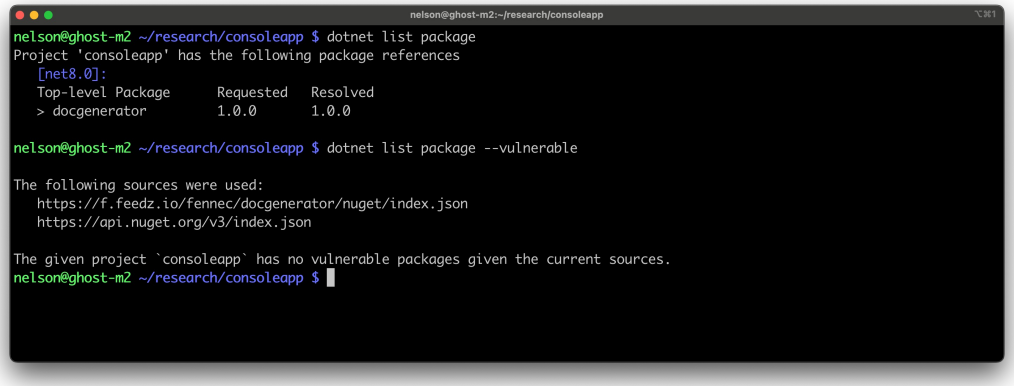
Vulnerabilities in Libraries



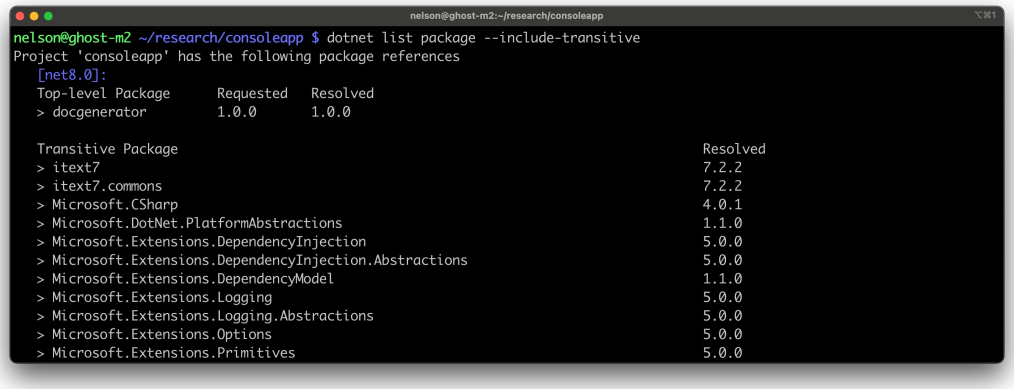
@nielstanis@infosec.exchange

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

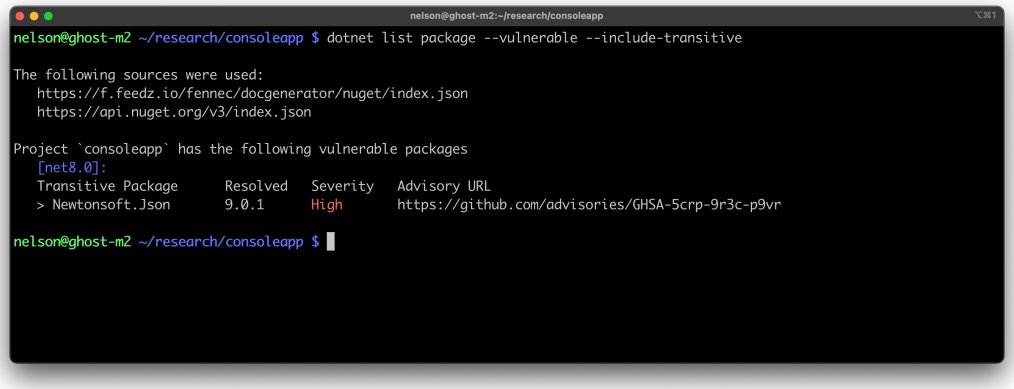
DotNet CLI


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 $

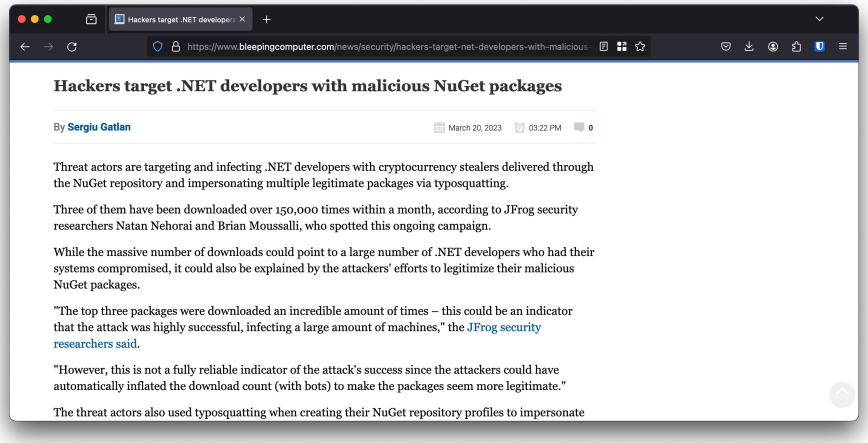
DotNet CLI


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 $

Malicious Package



Hackers target .NET developers with malicious NuGet packages

By Sergiu Gatlan — March 20, 2023 03:22 PM 0

Threat actors are targeting and infecting .NET developers with cryptocurrency stealers delivered through the NuGet repository and impersonating multiple legitimate packages via typosquatting.

Three of them have been downloaded over 150,000 times within a month, according to JFrog security researchers Natan Nehorai and Brian Moussalli, who spotted this ongoing campaign.

While the massive number of downloads could point to a large number of .NET developers who had their systems compromised, it could also be explained by the attackers' efforts to legitimize their malicious NuGet packages.

"The top three packages were downloaded an incredible amount of times – this could be an indicator that the attack was highly successful, infecting a large amount of machines," the JFrog security researchers said.

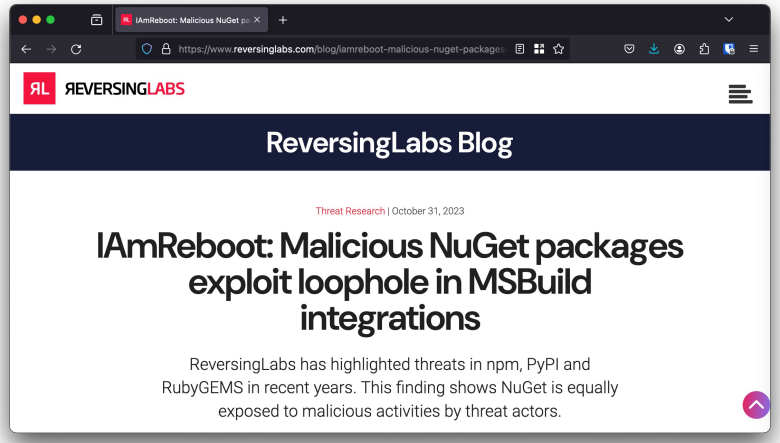
"However, this is not a fully reliable indicator of the attack's success since the attackers could have automatically inflated the download count (with bots) to make the packages seem more legitimate."

The threat actors also used typosquatting when creating their NuGet repository profiles to impersonate

@nielstanis@infosec.exchange

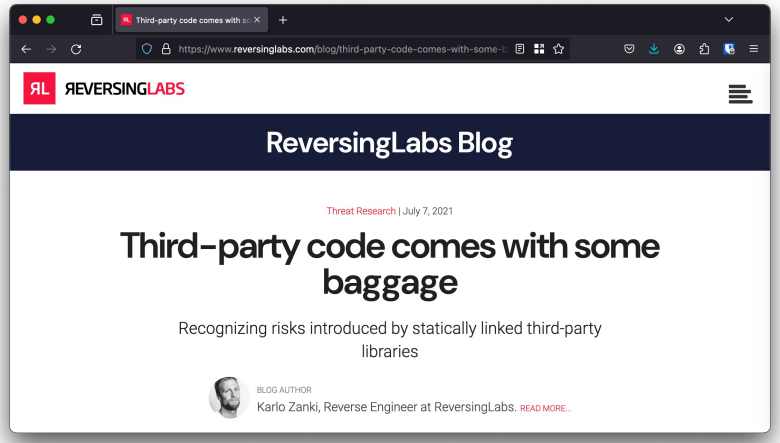
https://www.bleepingcomputer.com/news/security/hackers-target-net-developers-with-malicious-nuget-packages//

Malicious Package



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

Do you know what's inside?



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

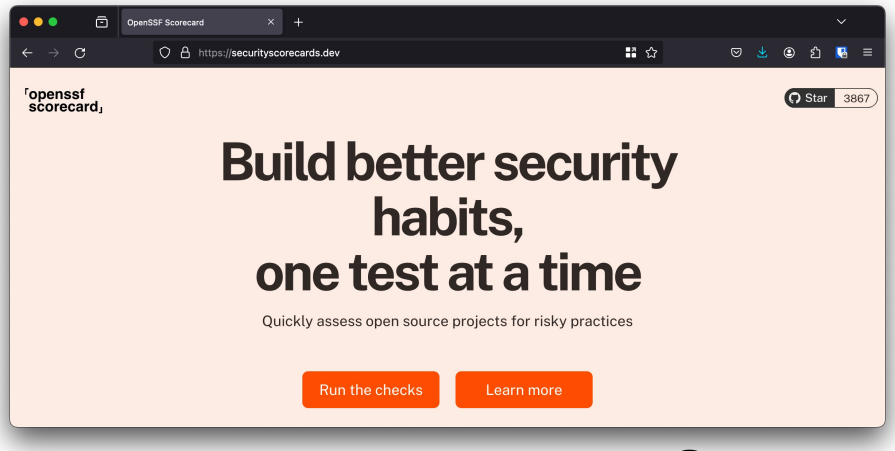
Nutrion Label for Software?

NDC { Security }

@nielstanis@infosec.exchange

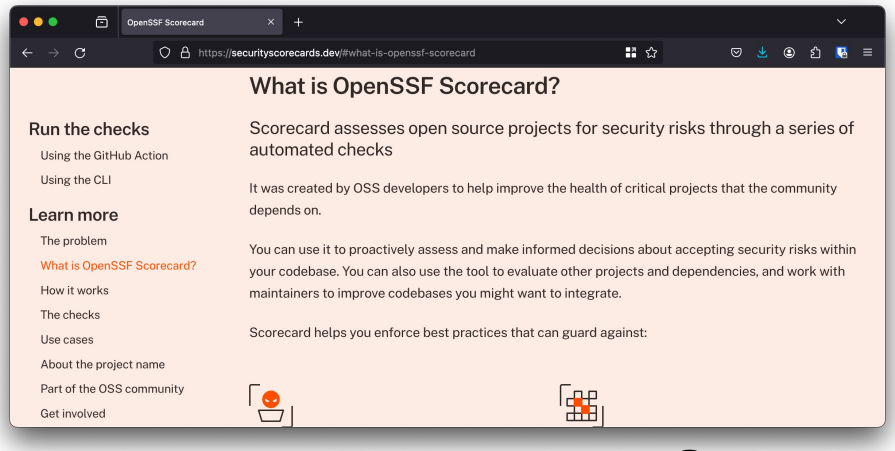
https://securityscorecards.dev/

OpenSSF Scorecards



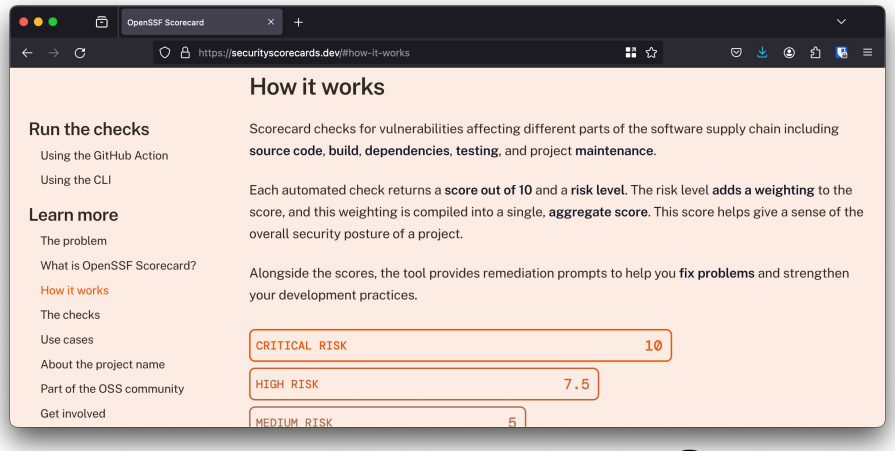
https://securityscorecards.dev/

OpenSSF Security Scorecards



https://securityscorecards.dev/

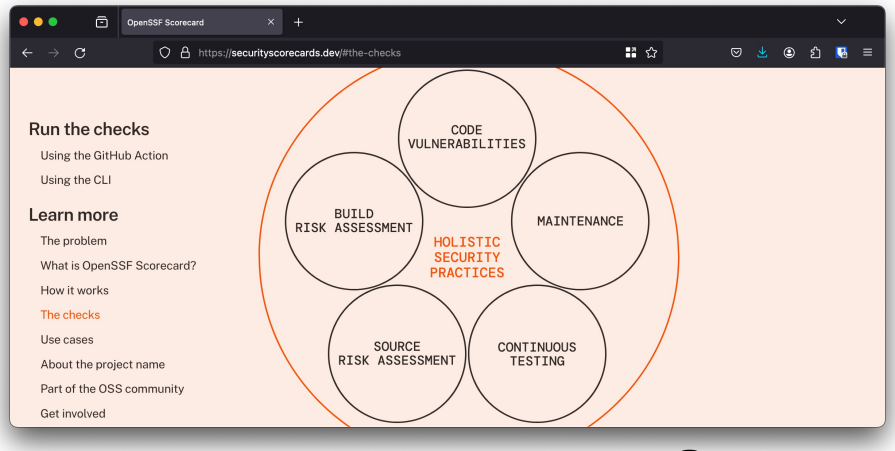
OpenSSF Security Scorecards



@nielstanis@infosec.exchange

https://securityscorecards.dev/

OpenSSF Security Scorecards

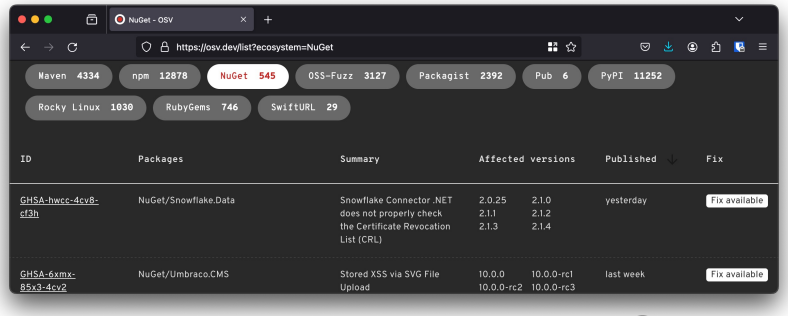


@nielstanis@infosec.exchange

https://securityscorecards.dev/

Code Vulnerabilities (High)

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



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.

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.

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).

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?

@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!

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/

@nielstanis@infosec.exchange

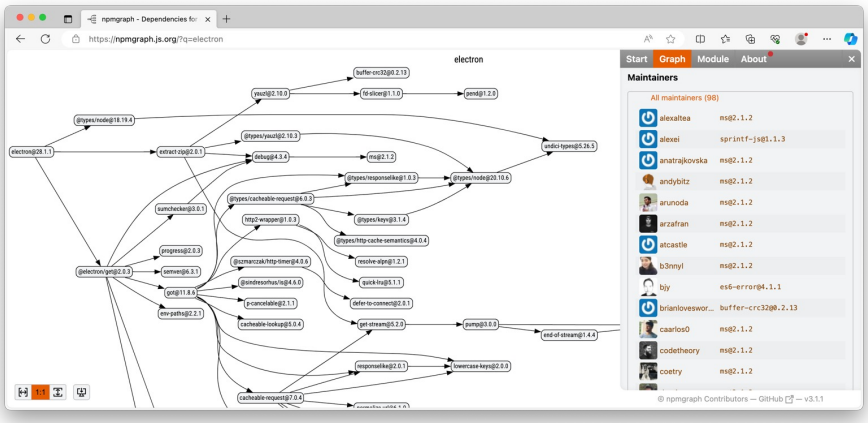
Source Risk Assesement
Code Review (Low)

- 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)

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
- But is a large list of contributors good?

Source Risk Assesement Contributors (Low)



@nielstanis@infosec.exchange

Build Risk Assesement
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

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.

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

Build Risk Assesement Packaging (Medium)

- This check tries to determine if the project is published as a package.
- Packages give users of a project an easy way to download, install, update, and uninstall the software by a package manager.

Build Risk Assessment
Signed Releases (High)

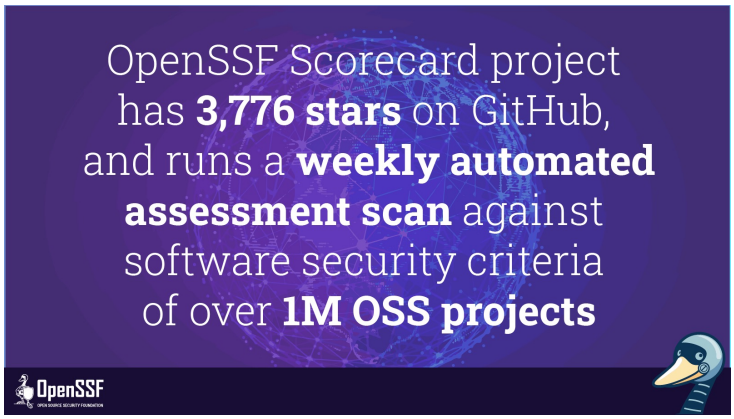
- This check tries to determine if the project cryptographically signs release artifacts.
  - Signed release packages
  - Signed build provenance

Demo OpenSSF Scorecard


```
Running checks
```


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

OpenSSF Annual Report 2023



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

What can we improve?

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

Fuzzing .NET

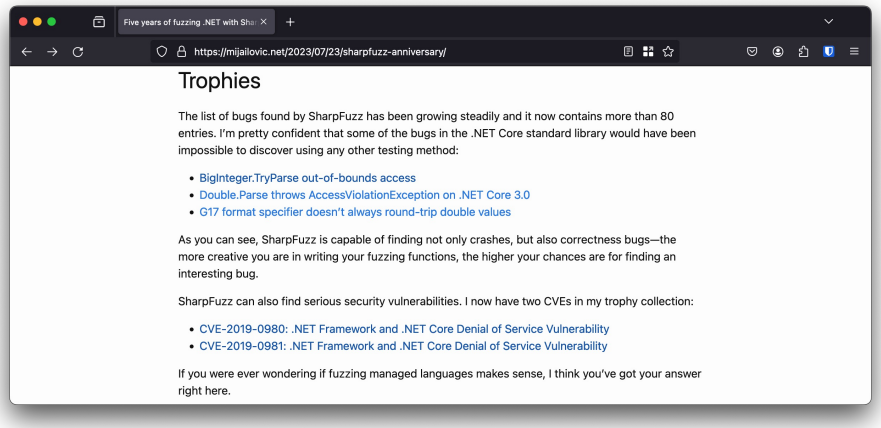
- Fuzzing, or fuzz testing, is defined as an automated software testing method that uses a wide range of ***invalid*** and unexpected data as input to find flaws in the software undergoing the test.
- Used a lot for finding C/C++ memory issues
- Can it be of any value with managed languages like .NET?

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

Fuzzing .NET & SharpFuzz

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

Fuzzing .NET & SharpFuzz



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

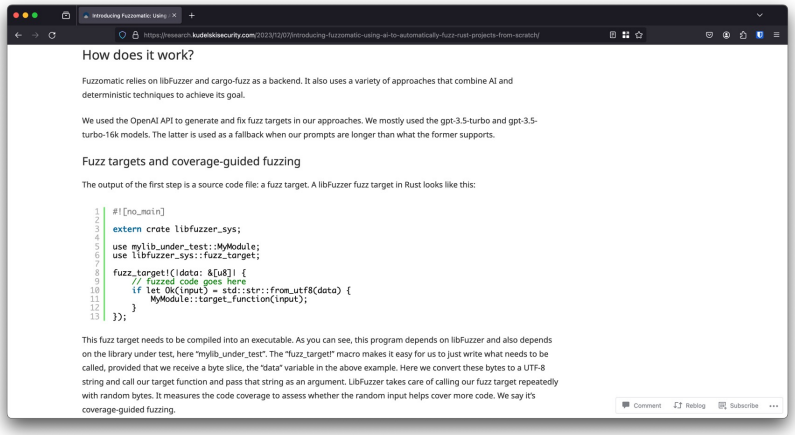
Fuzzing .NET – Jil JSON Serializer


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


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:

```
#![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.

@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);
}
```


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");
    }
}
```

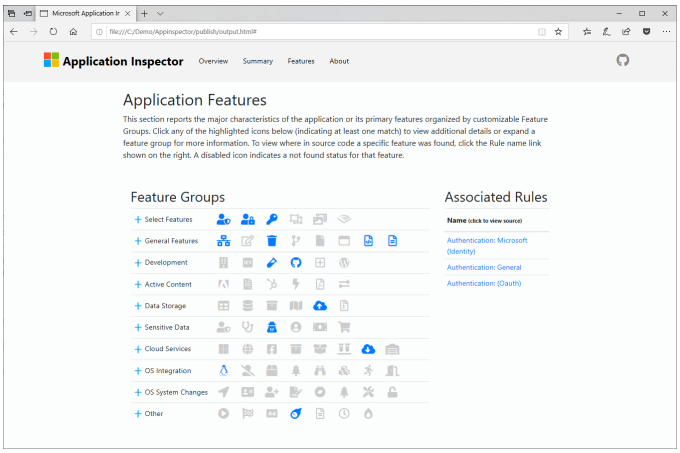

@nielstanis@infosec.exchange

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

.NET Reproducibility

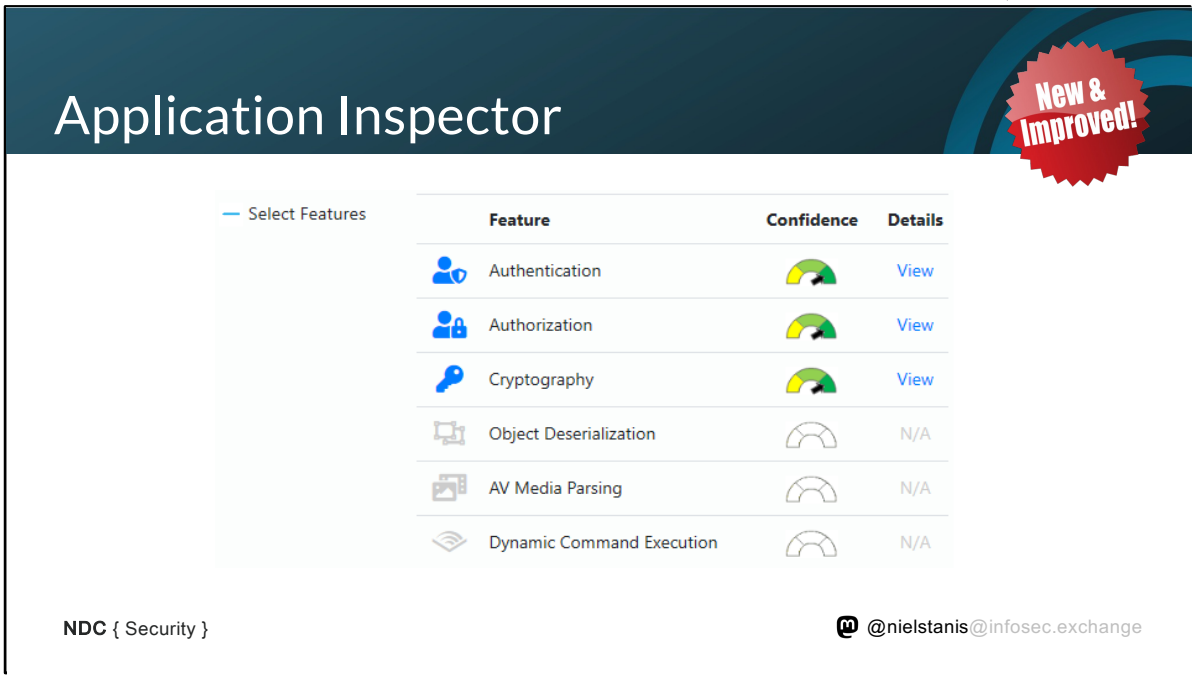
- Reproducible builds are a set of software development practices that create an independently-verifiable path from source to binary code.
- .NET Roslyn Deterministic Inputs
- How reproducible is a simple console app?

Application Inspector



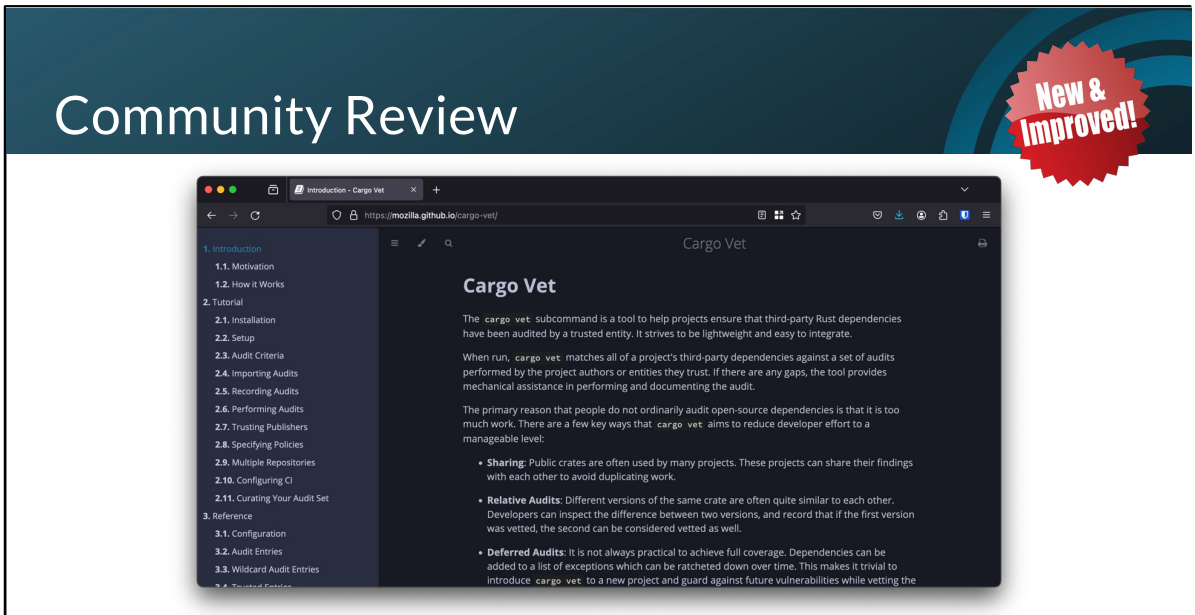
https://github.com/microsoft/ApplicationInspector

Application Inspector



https://github.com/microsoft/ApplicationInspector

Community Review



@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

Questions?

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