

VERACODE

Reviewing NuGet Packages security easily using OpenSSF Scorecard

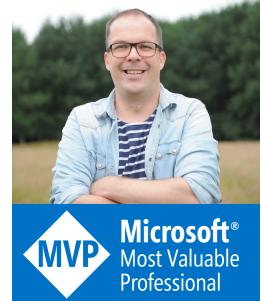
Niels Tanis
Sr. Principal Security Researcher



Who am I?



VERACODE



MVP Microsoft®
Most Valuable
Professional

- 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

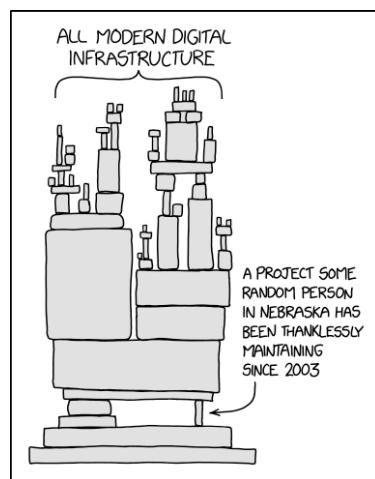
[bitbash]

@nielstanis@infosec.exchange

Modern Application Architecture XKCD 2347



[bitbash]



@nielstanis@infosec.exchange

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

Agenda

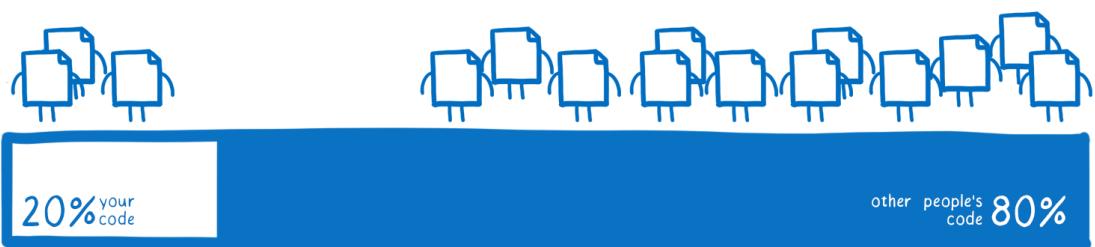
- Risks in 3rd NuGet Package
- OpenSFF Scorecard
- New & Improved
- Conclusion - Q&A



@nielstanis@infosec.exchange

[bitbash]

Average codebase composition



[bitbash]

@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.

[bitbash]

@nielstanis@infosec.exchange

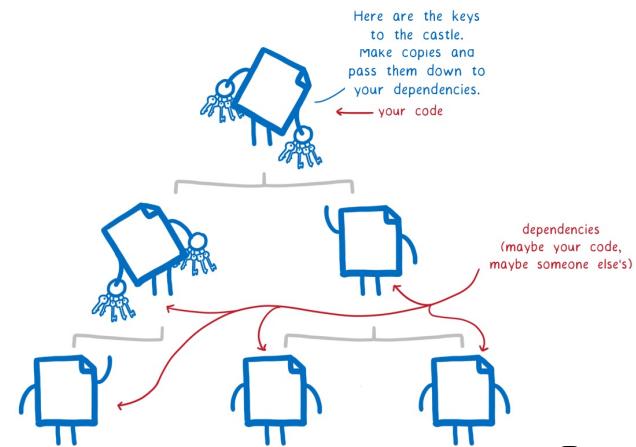
<https://www.veracode.com/blog/research/state-log4j-vulnerabilities-how-much-did-log4shell-change>

Average codebase composition



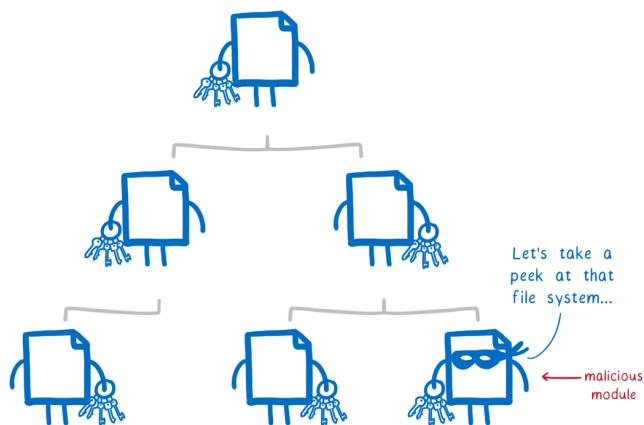
[bitbash]

@nielstanis@infosec.exchange



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

Malicious Assembly



bitbash

@nielstanis@infosec.exchange

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

The screenshot shows a news article titled "Hackers target .NET developers with malicious NuGet packages" from BleepingComputer.com. The article discusses threat actors targeting .NET developers by impersonating legitimate packages via typosquatting. It quotes researchers from JFrog security and provides context on the attack's scale and methods. The page includes a sidebar with a "REBELS" logo and a "bitbash" watermark.

Malicious Package

Hackers target .NET developers with malicious NuGet packages

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

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.

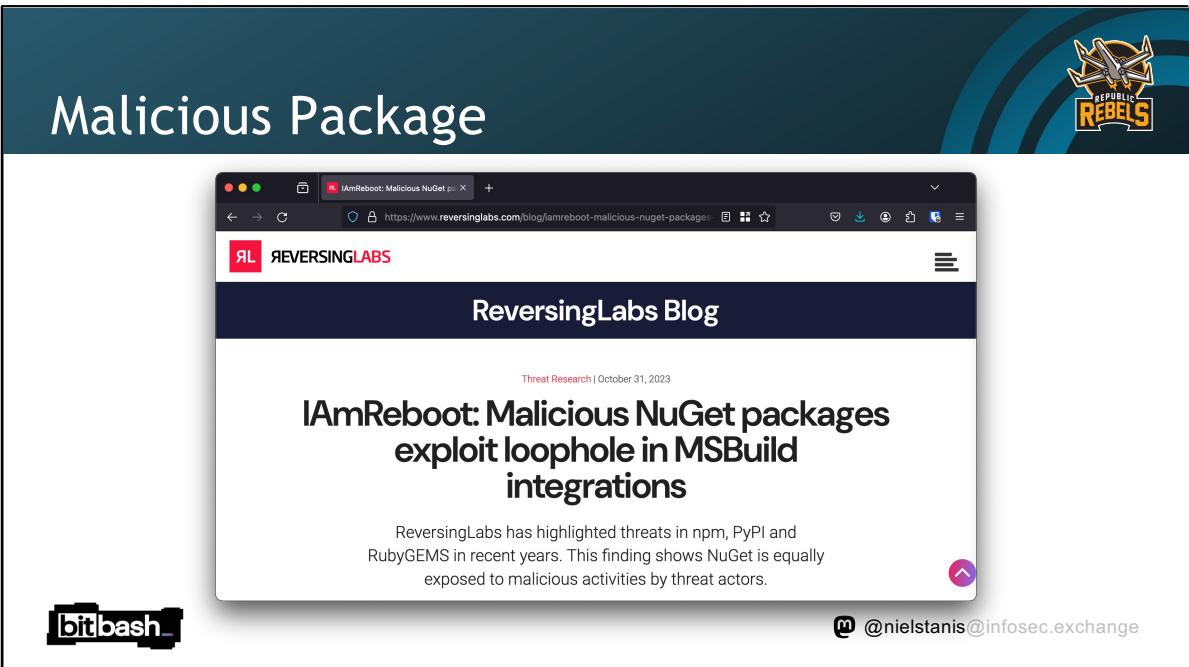
"However, this is not a fully reliable indicator of the attack's success since the attackers could have automatically inflated the download count (with bots) to make the packages seem more legitimate."

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

[bitbash]

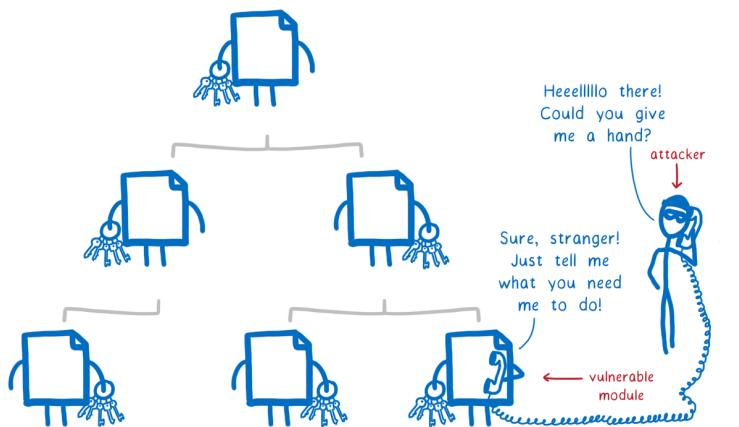
@nielstanis@infosec.exchange

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



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

Vulnerable Assembly



@nielstanis@infosec.exchange

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

Vulnerabilities in Libraries

The screenshot shows a GitHub issue page for Microsoft Security Advisory CVE-2023-36558: .NET Security Feature Bypass Vulnerability. The page includes a header with the title, a sidebar with repository navigation, and a main content area with the advisory details. The advisory summary states: "Microsoft is releasing this security advisory to provide information about a vulnerability in ASP.NET Core 6.0, ASP.NET Core 7.0 and, ASP.NET Core 8.0 RC2. This advisory also provides guidance on what developers can do to update their applications to address this vulnerability. A security feature bypass vulnerability exists in ASP.NET where an unauthenticated user is able to bypass validation on Blazor server forms which could trigger unintended actions." The issue has 0 comments and 0 pull requests. The sidebar shows the repository has 283 issues and 0 pull requests. The footer includes a watermark for "bitbash" and a Twitter handle "@nielstanis@infosec.exchange".

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

DotNet CLI



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

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

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

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

[bitbash]

@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



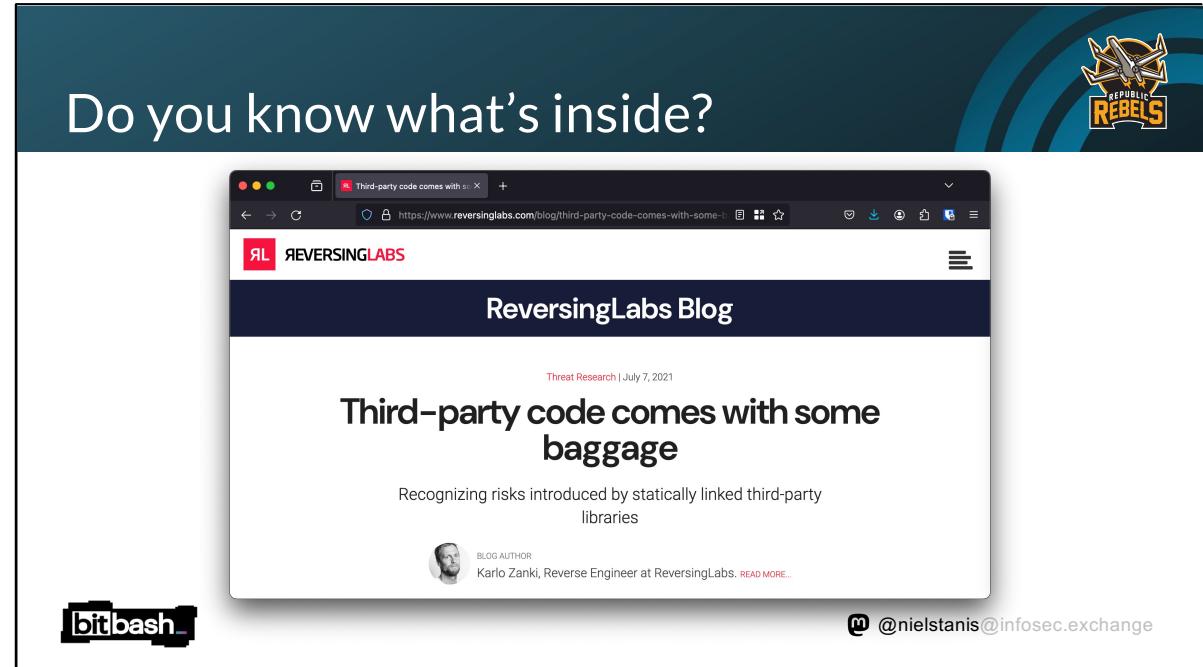
```
nelson@ghost-m2 ~/research/consoleapp $ dotnet list package --vulnerable --include-transitive
The following sources were used:
https://f.feedz.io/fennec/docgenerator/nuget/index.json
https://api.nuget.org/v3/index.json

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 $
```

[bitbash]

@nielstanis@infosec.exchange



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

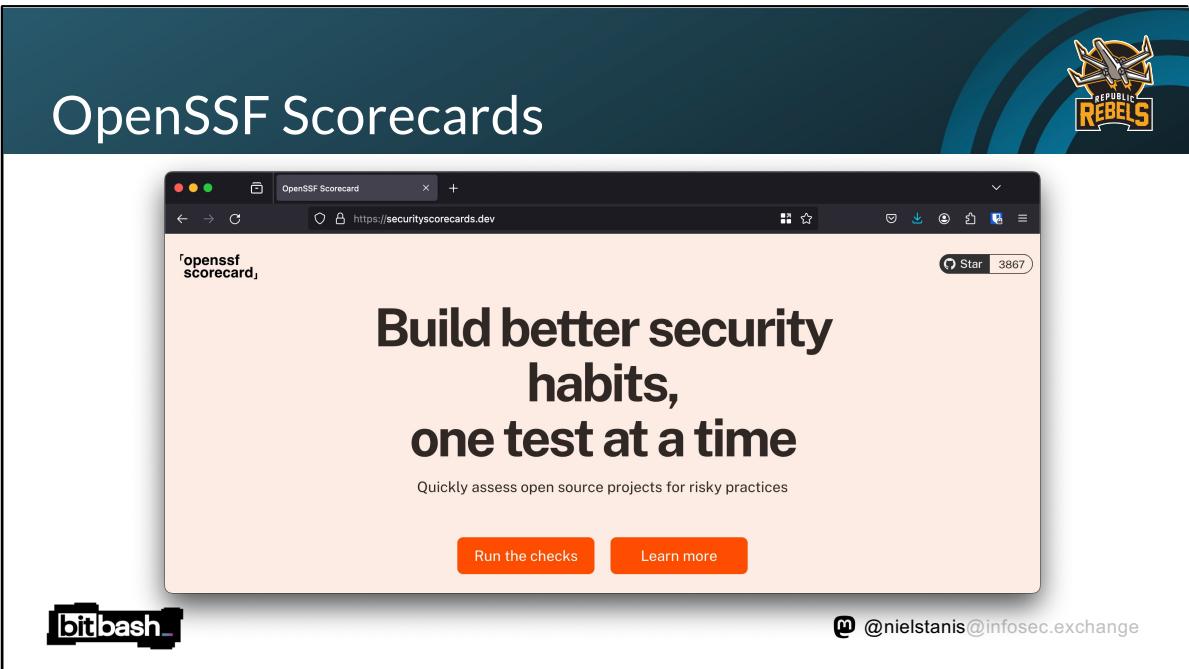


Nutrition Label for Software?



@nielstanis@infosec.exchange

<https://securityscorecards.dev/>



<https://securityscorecards.dev/>

The screenshot shows a web browser window with the title "OpenSSF Security Scorecards". In the top right corner, there is a logo for "REPUBLIC REBELS" featuring a stylized eagle. The main content area has a light orange background. On the left, there is a sidebar with two sections: "Run the checks" and "Learn more". The "Run the checks" section contains links for "Using the GitHub Action" and "Using the CLI". The "Learn more" section contains links for "The problem", "What is OpenSSF Scorecard?", "How it works", "The checks", "Use cases", "About the project name", "Part of the OSS community", and "Get involved". The central content area is titled "What is OpenSSF Scorecard?". It includes a paragraph about the scorecard assessing projects through automated checks, its creation by OSS developers to improve critical project health, and its use for proactive assessment and best practices enforcement. Below this text are two small icons: one of a person in a bowl-like shape and another of a grid with orange squares.

bitbash [REDACTED]

[@nielstanis@infosec.exchange](https://securityscorecards.dev/#what-is-openssf-scorecard)

<https://securityscorecards.dev/>

The screenshot shows a web browser window with the title "OpenSSF Security Scorecard" and the URL "https://securityscorecards.dev/#how-it-works". The page has a dark blue header with the "REBELS" logo on the right. The main content area has a light orange background. It features a sidebar with links like "Run the checks", "Learn more", and "Get involved". The main content area is titled "How it works" and contains text about scorecard checks, risk levels, and remediation prompts. At the bottom, there are three horizontal bars representing risk levels: "CRITICAL RISK" (10), "HIGH RISK" (7.5), and "MEDIUM RISK" (5). The footer includes the "bitbash" logo and a Twitter handle "@nielstanis@infosec.exchange".

<https://securityscorecards.dev/>

OpenSSF Security Scorecards

The screenshot shows the OpenSSF Security Scorecards website. 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 with a central orange circle labeled "HOLISTIC SECURITY PRACTICES". Five smaller white circles are arranged around it, connected by lines: "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. At the bottom of the page, there's a "bitbash" logo and a Twitter handle "@nielstanis@infosec.exchange".

<https://securityscorecards.dev/>

Code Vulnerabilities (High)



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

The screenshot shows a web browser window titled "NuGet - OSV" displaying the OSV service interface. The URL is https://osv.dev/list?ecosystem=NuGet. The page header includes statistics for various ecosystems: Maven (4334), npm (12878), NuGet (545), OSS-Fuzz (3127), Packagist (2392), Pub (6), PyPI (11252), Rocky Linux (1030), RubyGems (746), and SwiftURL (29). Below this is a table listing two vulnerabilities:

ID	Packages	Summary	Affected versions	Published	Fix
GHSA-hwcc-4cy8-cf3h	NuGet/Snowflake.Data	Snowflake Connector .NET does not properly check the Certificate Revocation List (CRL)	2.0.25 2.1.1 2.1.3	2.1.0 2.1.2 2.1.4	yesterday
GHSAA-6xmx-85x3-4cy2	NuGet/Umbraco.CMS	Stored XSS via SVG File Upload	10.0.0 10.0.0-rc2	10.0.0-rc1 10.0.0-rc3	last week

@nielstanis@infosec.exchange

Maintenance Dependency-Update-Tool (**High**)



- This check tries to determine if the project uses a dependency update tool, use of: Dependabot, Renovate bot
- Out-of-date dependencies make a project vulnerable to known flaws and prone to attacks.

[bitbash]

@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.

[bitbash]

[@nielstanis@infosec.exchange](mailto: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.

[bitbash]

@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)



- 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).

bitbash

@nielstanis@infosec.exchange

Continuous testing Fuzzing (Medium)



- This check tries to determine if the project uses fuzzing by checking:
 - Added to [OSS-Fuzz](#) project.
 - If [ClusterFuzzLite](#) is deployed in the repository;
 - If there are user-defined language-specified fuzzing functions in the repository.
- Does it make sense to do fuzzing on .NET projects?

bitbash

@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!

[bitbash]

@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!

[bitbash]

[@nielstanis@infosec.exchange](mailto: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!

[bitbash]

@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/>

[bitbash]

@nielstanis@infosec.exchange

Source Risk Assessment Code Review (Low)



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

[bitbash]

@nielstanis@infosec.exchange

Source Risk Assessment Contributors (Low)

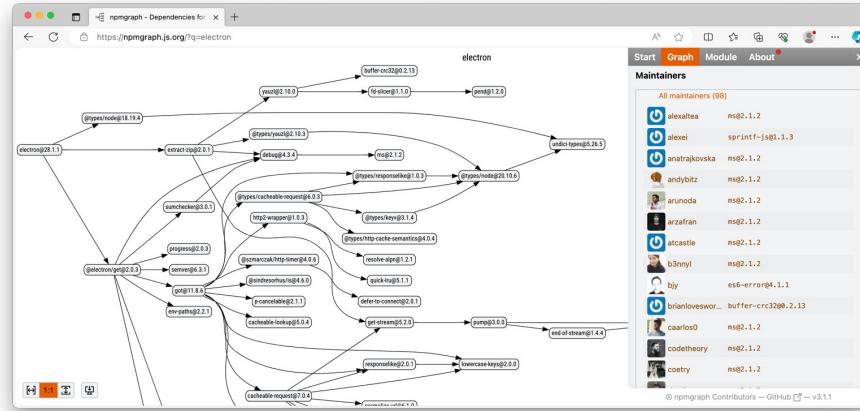


- This check tries to determine if the project has recent contributors from multiple organizations (e.g., companies).
- Relying on single contributor is a risk
- But is a large list of contributors good?

[bitbash]

@nielstanis@infosec.exchange

Source Risk Assessment Contributors (Low)



 @nielstanis@infosec.exchange

Build Risk Assessment Pinned Dependencies (**High**)



- This check tries to determine if the project pins dependencies used during its build and release process.
- **RestorePackagesWithLockFile** in MSBuild results in packages.lock.json file containing versioned dependency tree with hashes

[bitbash]

@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.

[bitbash]

@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.

[bitbash]

@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

[bitbash]

@nielstanis@infosec.exchange

Demo OpenSSF Scorecard Fennec CLI



Running checks



@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**



[bitbash]

@nielstanis@infosec.exchange

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

What can we improve?



[bitbash]

@nielstanis@infosec.exchange

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

Fuzzing .NET

New & Improved!



- 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?



 @nielstanis@infosec.exchange

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

Fuzzing .NET & SharpFuzz

Nemanja Mijailovic's Blog

Five years of fuzzing .NET with SharpFuzz

Jul 23, 2023

It's been almost five years since I created [SharpFuzz](#), the only .NET coverage-guided fuzzer. I already have a blog post on how it works, what it can do for you, and what bugs it found, so check it out if this is the first time you hear about SharpFuzz:

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

A lot of interesting things have happened since then. SharpFuzz now works with libFuzzer, Windows, and .NET Framework. And it can finally fuzz the .NET Core base-class library! The whole fuzzing process has been dramatically simplified, too.

Not many people are aware of all these developments, so I decided to write this anniversary blog post and showcase everything SharpFuzz is currently capable of.

  @nielstanis@infosec.exchange

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

The screenshot shows a blog post titled "Fuzzing .NET & SharpFuzz". In the top right corner, there is a "New & Improved!" badge with a yellow and red design. Below the title, there is a section titled "Trophies" which lists several bugs found by SharpFuzz. The post also mentions two CVEs: CVE-2019-0980 and CVE-2019-0981. At the bottom of the post, there is a link to bitbash.

<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) { }
    });
}
```

[bitbash]

@nielstanis@infosec.exchange

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

Fuzzomatic: Using AI to Fuzz Rust

New & Improved!

REPUBLIC REBELS

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 !data.ok() {
11 |     std::str::from_utf8(data).unwrap();
12 |     MyModule::target.function(data);
13 |   }
14 |});
```

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 ...

@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);
}
```

[bitbash]

@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>

.NET Reproducibility

New & Improved!



- 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?



[@nielstanis@infosec.exchange](mailto:nielstanis@infosec.exchange)

The screenshot shows the Microsoft Application Inspector application window. At the top, there's a banner with the text "New & Improved!" and a logo for "REPUBLIC REBELS". The main title "Application Inspector" is displayed prominently. Below the title, the application features a navigation bar with tabs: Overview, Summary, Features, and About. The "Features" tab is selected. The main content area is titled "Application Features" and contains a detailed description of the feature groups. On the left, a sidebar lists "Feature Groups" with icons: Select Features, General Features, Development, Active Content, Data Storage, Sensitive Data, Cloud Services, OS Integration, OS System Changes, and Other. To the right, under "Associated Rules", there is a table with one row: Name (click to view source) - Authentication: Microsoft (Identity). The bottom right corner of the window has a watermark with the Microsoft logo and the text "@nielstanis@infosec.exchange".

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

The screenshot shows the Microsoft Application Inspector interface. At the top, there's a dark header with the title "Application Inspector". In the top right corner, there's a red circular badge with the text "New & Improved!" and the "REBELS" logo. Below the header is a table with the following columns: "Feature", "Confidence", and "Details". The table lists six features: Authentication, Authorization, Cryptography, Object Deserialization, AV Media Parsing, and Dynamic Command Execution. Each feature has a corresponding icon and a confidence meter. The "Details" column contains links labeled "View" or "N/A".

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

@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

bitbash

@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 its own!
- NuGet Package Scoring (NET Score)
- Room for .NET specific improvements with Fennec CLI & contributions to OpenSSF Scorecard project



bitbash

@nielstanis@infosec.exchange

Questions?



@nielstanis@infosec.exchange

bitbash

Thanks!



- <https://github.com/nielstanis/bitbash2024/>
- ntanis at Veracode.com
- @nielstanis@infosec.exchange
- <https://www.fennec.dev> & <https://blog.fennec.dev>
- Bedankt! Thank you!



[@nielstanis@infosec.exchange](mailto:nielstanis@infosec.exchange)