

# Sandboxing .NET Assemblies for fun, profit and, of course Security!

Niels Tanis

NDC { Porto }

VERACODE



## Who am I?



- Niels Tanis
- Principal Security Researcher @ Veracode
  - Background .NET Development, Pentesting/ethical hacking, and software security consultancy
  - ISC<sup>2</sup> CSSLP
  - Research on static analysis for .NET apps



NDC { Porto }

@nielstanis

# Agenda



- Introduction
- The security risks of third party libraries
- Sandboxing techniques
- Let's create a sandbox!
- Conclusion
- QA

NDC { Porto }

@nielstanis

## Third Party Libraries

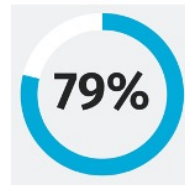


- Big chunk (80%+) of our apps consists of 3rd party libraries
- Efficient in time, why reinvent the wheel?
- How actively is it maintained?
- What do they do for security?

# State Of Software Security v11 2021



*"Despite this dynamic landscape, 79 percent of the time, developers never update third-party libraries after including them in a codebase."*



NDC { Porto }

@nielstanis

<https://info.veracode.com/fy22-state-of-software-security-v11-open-source-edition.html>

# Vulnerabilities in libraries



The screenshot shows a GitHub issue page for the repository 'dotnet/announcements'. The issue title is 'Microsoft Security Advisory CVE-2022-24512 | .NET Remote Code Execution Vulnerability #213'. It is marked as 'Open' and was opened by 'dcwhittaker' on 8 Mar. The issue has 0 comments. A comment from 'dcwhittaker' is visible, dated 8 Mar, with the text: 'Microsoft Security Advisory CVE-2022-24512 | .NET Remote Code Execution Vulnerability'. The comment includes an 'Executive summary' section stating: 'Microsoft is releasing this security advisory to provide information about a vulnerability in .NET 6.0, .NET 5.0, and .NET Core 3.1. This advisory also provides guidance on what developers can do to update their applications to remove this vulnerability. A Remote Code Execution vulnerability exists in .NET 6.0, .NET 5.0, and .NET Core 3.1 where a stack buffer overrun occurs in .NET Double Parse routine.' The 'Discussion' section mentions: 'Discussion for this issue can be found at dotnet/runtime#68346'. On the right side, the 'Assignees' section shows 'No one assigned'. The 'Labels' section includes 'Monthly-Update', '.NET Core 3.1', '.NET 5.0', '.NET 6.0', and 'Patch-Tuesday'. The 'Projects' section shows 'None yet'. The 'Milestone' section shows 'No milestone'.

NDC { Porto }

@nielstanis

<https://github.com/dotnet/announcements/issues/213>

# Vulnerabilities in libraries



**CYBERSECURITY  
& INFRASTRUCTURE  
SECURITY AGENCY**

Alerts and Tips   Resources   Industrial Control Systems

[National Cyber Awareness System](#) > [Current Activity](#) > [Malware Discovered in Popular NPM Package, ua-parser-js](#)

## Malware Discovered in Popular NPM Package, ua-parser-js

Original release date: October 22, 2021

[Print](#)   [Twitter](#)   [Sound](#)   [Share](#)

Versions of a popular NPM package named `ua-parser-js` it was found to contain malicious code<sup>1</sup>. `ua-parser-js` is used in apps and websites to discover the type of device or browser a person is using from User-Agent data. A computer or device with the affected software installed or running could allow a remote attacker to obtain sensitive information or take control of the system.

CISA urges users and administrators using compromised ua-parser-js versions 0.7.29, 0.8.0, and 1.0.0 to update to the respective patched versions: 0.7.30, 0.8.1, 1.0.1

For more information, see [Embedded malware in ua-parser-js](#)<sup>2</sup>.

**NDC { Porto }**@nielstanis

<https://us-cert.cisa.gov/ncas/current-activity/2021/10/22/malware-discovered-popular-npm-package-ua-parser-js>  
<https://portswigger.net/daily-swig/popular-npm-package-ua-parser-js-poisoned-with-cryptomining-password-stealing-malware>

# Vulnerabilities in libraries



November 15, 2021 — Open Source, Security

## GitHub's commitment to npm ecosystem security



Mike Hanley

The npm registry is central to all JavaScript development, and, as stewards of the registry, ensuring its security is a responsibility GitHub takes seriously. Transparency is key in maintaining the trust of our community. Today, we are sharing details of recent incidents on the npm registry, the details of our investigations, and how we're continuing to invest in the security of npm. These investments include the requirement of two-factor authentication (2FA) during authentication for maintainers and admins of popular packages on npm, starting with a cohort of top packages in the first quarter of 2022. Read on to learn more.

Share

Twitter

Facebook

LinkedIn

### Automated malware detection and requiring 2FA to combat maintainer account takeovers

We periodically see incidents on the registry where npm accounts are compromised by malicious actors and then used to insert malicious code into popular packages to which these accounts have access. Examples include the recent takeovers of the `ua-parser-js`, `coa`, and `rc` packages. At times, these account takeover (ATO) events have involved npm accounts where two-factor authentication was not enabled.

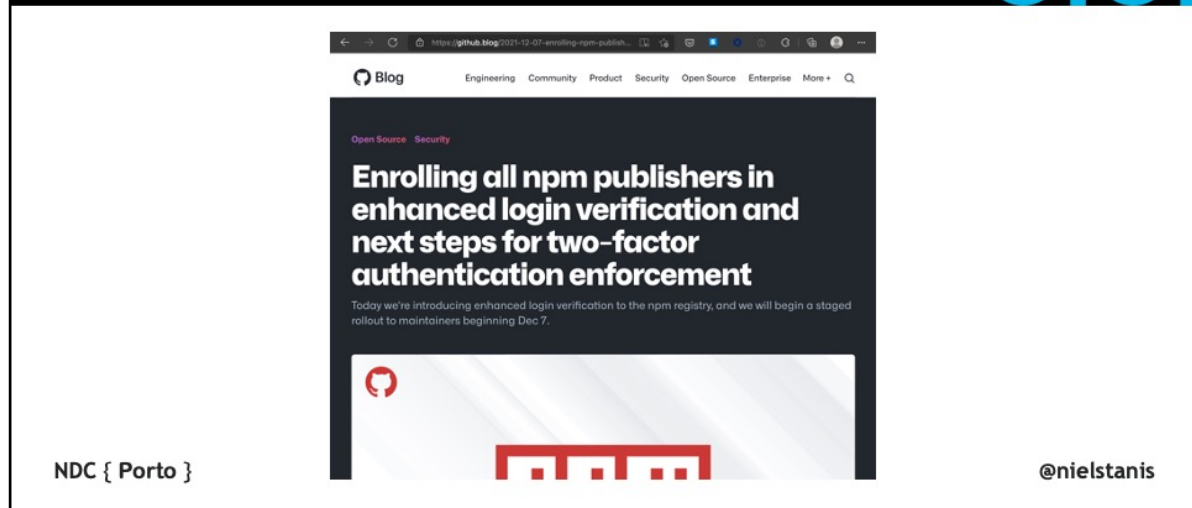
NDC { Porto }

@nielstanis

<https://github.blog/2021-11-15-githubs-commitment-to-npm-ecosystem-security/>



# Vulnerabilities in libraries



<https://github.blog/2021-12-07-enrolling-npm-publishers-enhanced-login-verification-two-factor-authentication-enforcement/>

# Vulnerabilities in libraries



## Third-party code comes with some baggage



### Recognizing risks introduced by statically linked third-party libraries

#### Introduction

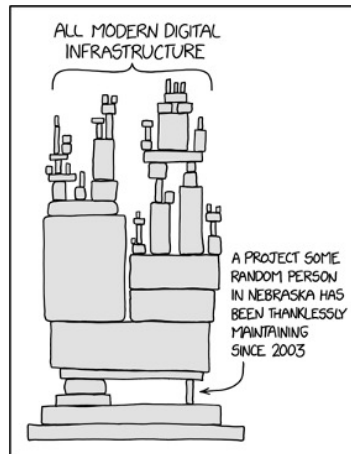
Developing software solutions is a complex task requiring a lot of time and resources. In order to accelerate time to market and reduce the cost, software developers create smaller pieces of functional code which can be reused across

NDC { Porto }

@nielstanis

<https://blog.secure.software/third-party-code-comes-with-some-baggage>

## XKDC - Dependency



NDC { Porto }

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

## Sandboxing .NET Assemblies



- Is there a way we can do a better job?
- A way for us to reduce the security risks?
- Keep in mind it's not a matter of how it's more when!

## Sandboxing .NET Assemblies

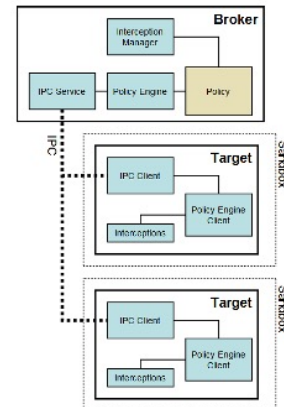


- We want to use the library without modification
- Can we maybe create a controlled (restricted) sandbox?
- A sandbox with limited capabilities?

# Browser Sandbox



- Chromium Sandbox
- No direct system access
- Each OS related call is done via IPC
- Firefox Sandbox
  - Containers & Site Isolation



NDC { Porto }

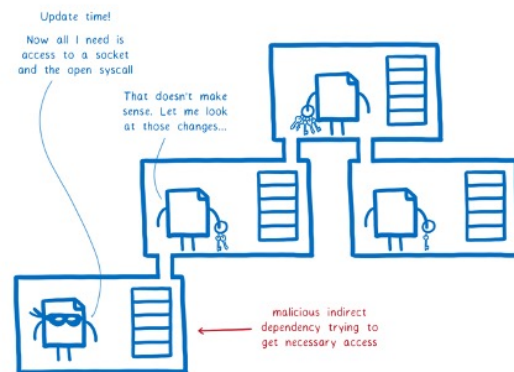
@nielstanis

<https://chromium.googlesource.com/chromium/src/+refs/heads/main/docs/design/sandbox.md>  
<https://hacks.mozilla.org/2021/05/introducing-firefox-new-site-isolation-security-architecture/>

# WebAssembly Nanoprocess



- Linear memory model
- WASM module isolation
- Declarative permissions
- Interface types
- WASI for BCL calls



NDC { Porto }

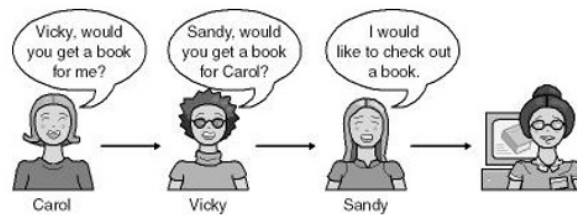
@nielstanis

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

## Code Access Security



- Evidence based model
- Code from different origins have different sets of rights
- Stack-walks that protect against luring attacks



NDC { Porto }

@nielstanis

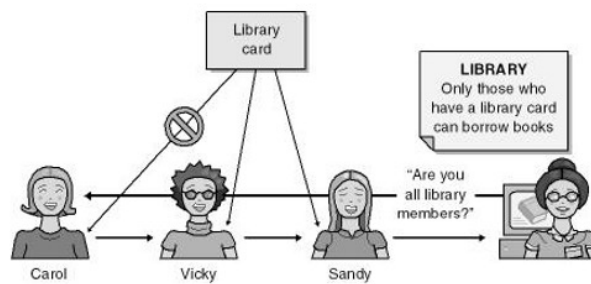
Figure 18-1; Writing Secure Code 2nd Edition



## Code Access Security



- Evidence library card
- Policy → Librarian only allows members



NDC { Porto }

@nielstanis

Figure 18-2; Writing Secure Code 2nd Edition

# Code Access Security



## •Stack walk

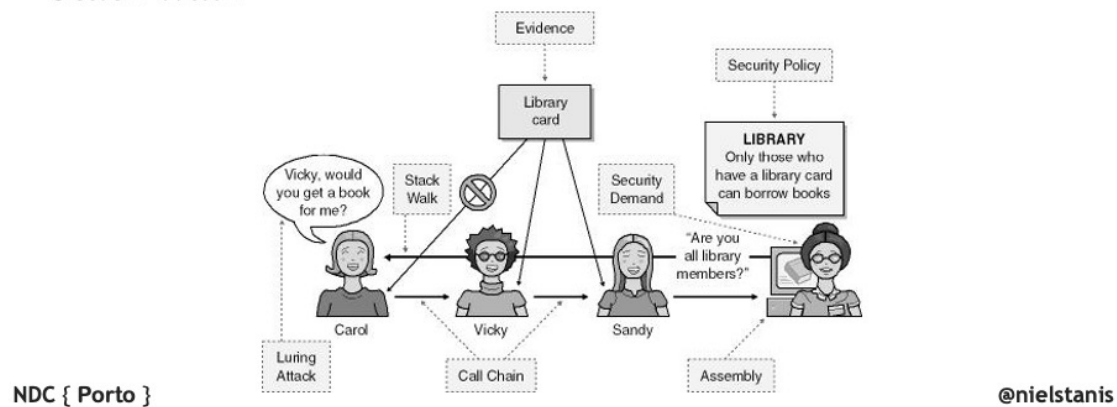


Figure 18-2; Writing Secure Code 2nd Edition

## Code Access Security



- Most practical example, ASP.NET Medium Trust
- CAS is deprecated since .NET Framework 4
- Too complex in administering and use?
- Too early?

NDC { Porto }

@nielstanis

<https://docs.microsoft.com/en-us/previous-versions/dotnet/framework/code-access-security/code-access-security-basics>

<https://docs.microsoft.com/en-us/previous-versions/dotnet/framework/code-access-security/code-access-security-policy-compatibility-and-migration>

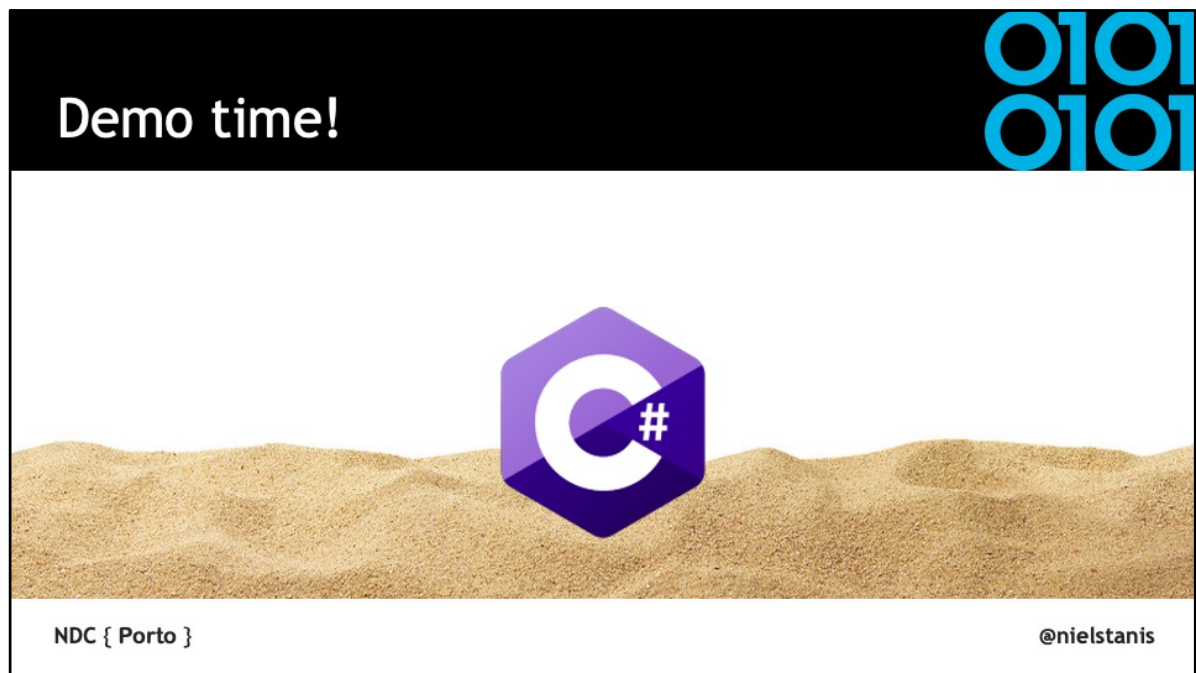


Image: C# logo <https://docs.microsoft.com/en-us/dotnet/csharp/>

## DocumentProcessor Package



- Use package as is!
  - Disclaimer: always comply with library license!
  - Not allowed to reverse engineer/decompile
- We do want to change behaviour:
  - Opening documents directly from URL - SSRF
  - Writing files to any arbitrary directory - Path Traversal
- There are *several* ways to *fix* this!

## AssemblyLoadContext



- Only single AppDomain in .NET Core.
- AssemblyLoadContext replaces the isolation mechanisms provided by multiple AppDomain instances in .NET Framework.
- Conceptually, a load context creates a scope for loading, resolving, and potentially unloading a set of assemblies.

NDC { Porto }

@nielstanis

<https://docs.microsoft.com/en-us/dotnet/api/system.runtime.loader.assemblyloadcontext?view=net-5.0>  
<https://docs.microsoft.com/en-us/dotnet/core/dependency-loading/understanding-assemblyloadcontext>

## AssemblyLoadContext



- It allows multiple versions of the same assembly to be loaded within a single process.
- It does not provide any security features. All code has full permissions of the process.
- But it does allow us to control what gets loaded!

NDC { Porto }

@nielstanis

<https://docs.microsoft.com/en-us/dotnet/api/system.runtime.loader.assemblyloadcontext?view=net-5.0>  
<https://docs.microsoft.com/en-us/dotnet/core/dependency-loading/understanding-assemblyloadcontext>

# AssemblyLoadContext



- Interface project used as shared contract
- Remove DocumentProcessor package from ConsoleApp
  - Add reference to interface project
- Create Library that implements interface
  - Reference interface project and DocumentProcessor Package
  - Self-contained deployment to folder that has all to be loaded by our sandboxed loadcontext

NDC { Porto }

@nielstanis

<https://docs.microsoft.com/en-us/dotnet/api/system.runtime.loader.assemblyloadcontext?view=net-5.0>  
<https://docs.microsoft.com/en-us/dotnet/core/dependency-loading/understanding-assemblyloadcontext>



# Sandboxing DocumentProcessor

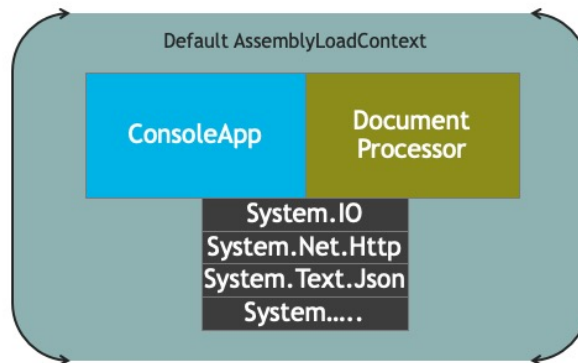


NDC { Porto }

@nielstanis

Image: C# logo <https://docs.microsoft.com/en-us/dotnet/csharp/>

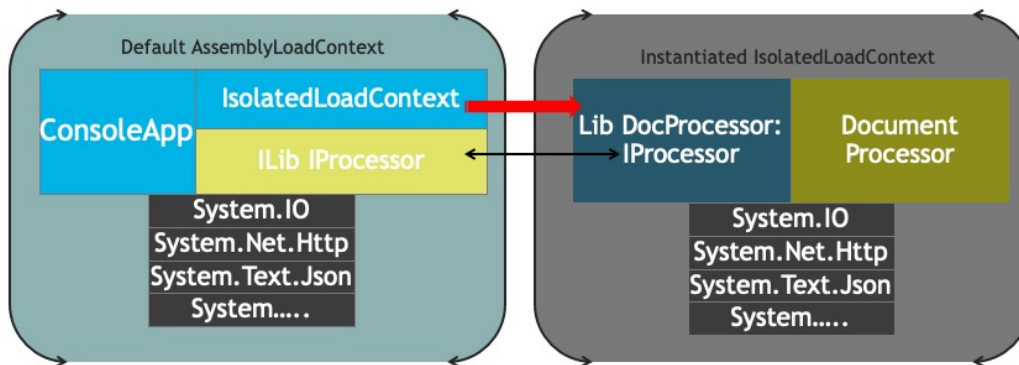
# ConsoleApp Start



NDC { Porto }

@nielstanis

# ConsoleApp & Sandboxed Library



NDC { Porto }

@nielstanis

## Removing Types?



- Self contained set of assemblies, could we not remove types?
- What about trimming that got introduced with .NET 5?
- Maybe we need something more rigorous?

## Patching with Harmony2

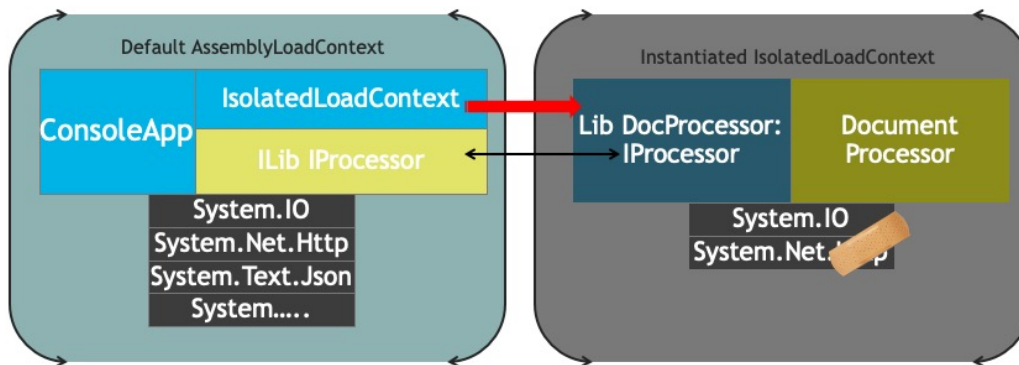


- A library for patching, replacing and decorating .NET and Mono methods during runtime.
  - Patch at runtime (pre- and postfix)
  - Transpile at compile time (rewrite IL)
- Harmony v2
  - Lib.Harmony on NuGet
  - <https://github.com/pardeike/Harmony>



Image: C# logo <https://docs.microsoft.com/en-us/dotnet/csharp/>

# ConsoleApp & Sandboxed Library



NDC { Porto }

@nielstanis

## Conclusion



- Update libraries; security problems get fixed
- Integrate security into your development lifecycle
- Know what libraries are used, where and what's inside and most important what you'd expect from it.



## Conclusion



- Futures of this Sandbox Concept
  - Easier developer integration (e.g. source generator)
  - Package + good guidance on how this can be used in different application contexts like ASP.NET Core.
  - Basic patches/policy that can be applied on libraries

## Questions?



- <https://github.com/nielstanis/ndcporto2022>
- ntanis at Veracode.com
- @NielsTanis on Twitter
- <https://blog.fennec.dev>
- Obrigado! Thanks!