

**VERACODE**

# Using WebAssembly to run, extend, and secure your .NET application

Niels Tanis  
Sr. Principal Security Researcher

*SWETUG  
GÖTEBORGS*

# 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

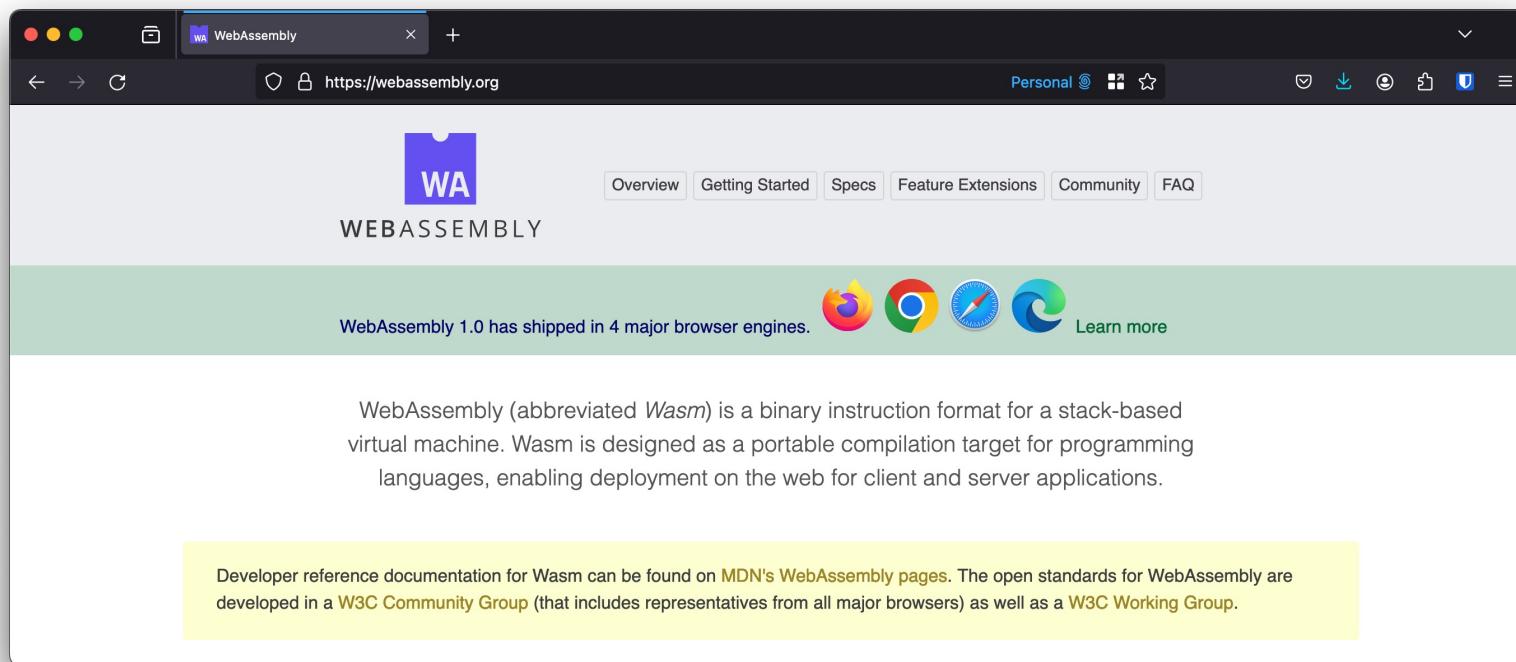
**VERACODE**



*SWETUGS*

@nielstanis@infosec.exchange

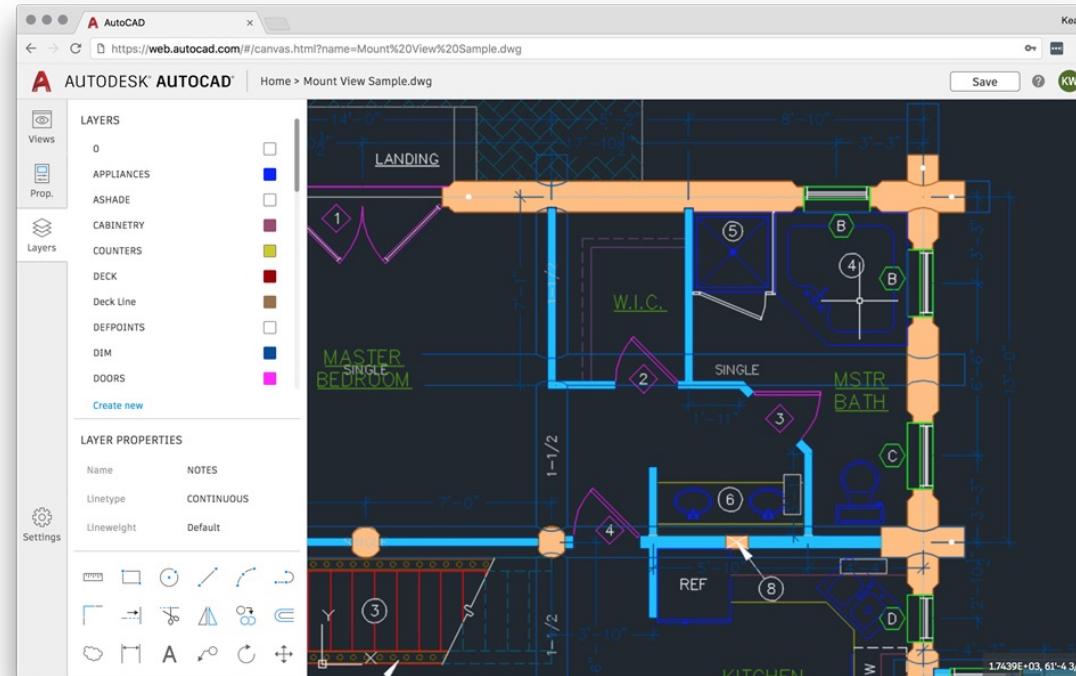
# WebAssembly



SWETUGS

@nielstanis@infosec.exchange

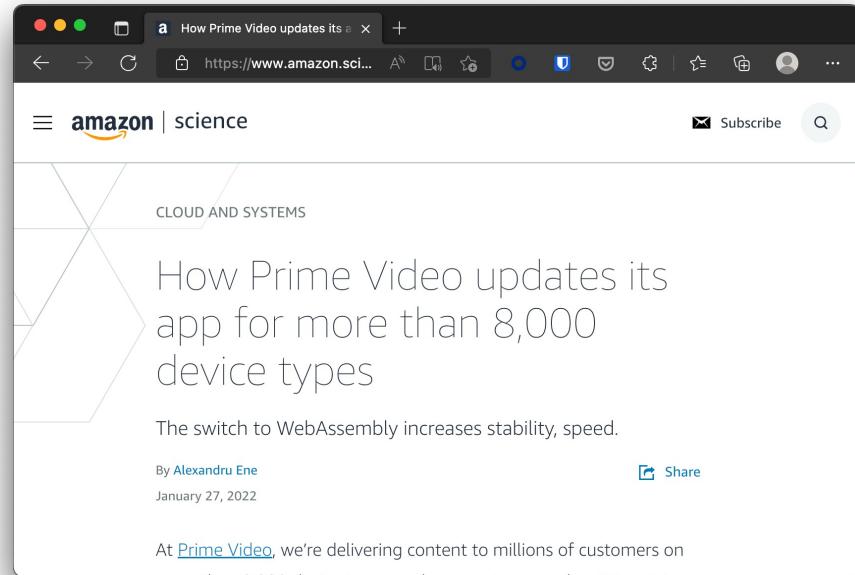
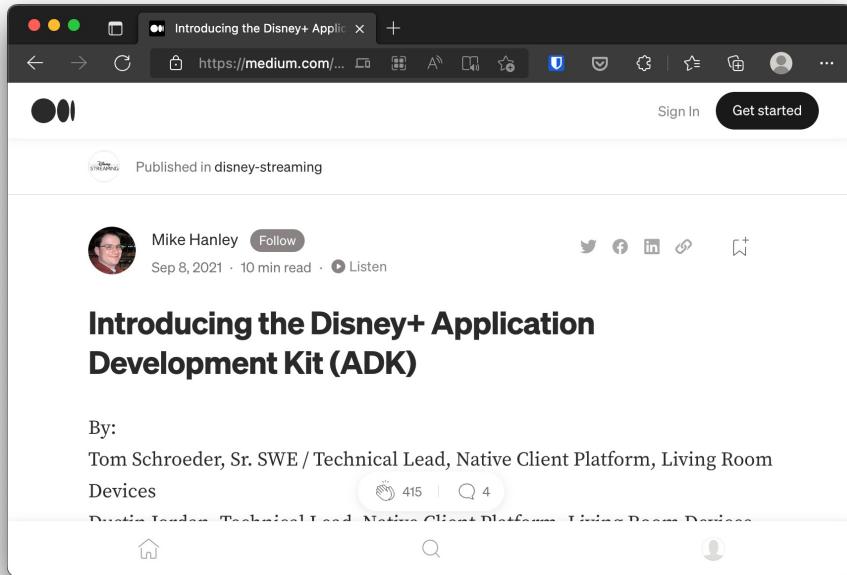
# WebAssembly - AutoCAD



SWETUGG

 @nielstanis@infosec.exchange

# WebAssembly - SDK's



SWETUGS

@nielstanis@infosec.exchange

# Agenda

- Introduction
- WebAssembly Design & Internals
- Running .NET on WebAssembly
- Extending .NET with WebAssembly
- Securing .NET with WebAssembly
- Conclusion
- Q&A

SWETUG6

@nielstanis@infosec.exchange

# WebAssembly Design

- **Be fast, efficient, and portable**

- Executed in near-native speed across different platforms

- **Be readable and debuggable**

- In low-level bytecode but also human readable

- **Keep secure**

- Run on sandboxed execution environment

- **Don't break the web**

- Ensure backwards compatibility



WEBASSEMBLY

SWETUGS

@nielstanis@infosec.exchange

# WebAssembly

- Binary instruction format for stack-based virtual machine similar to .NET CLR running MSIL or JVM running bytecode
- Designed as a portable compilation target



WEBASSEMBLY

SWETUGS

@nielstanis@infosec.exchange

# WebAssembly

- The security model of WebAssembly:
  - Protect users from buggy or malicious modules
  - Provide developers with useful primitives and mitigations for developing safe applications



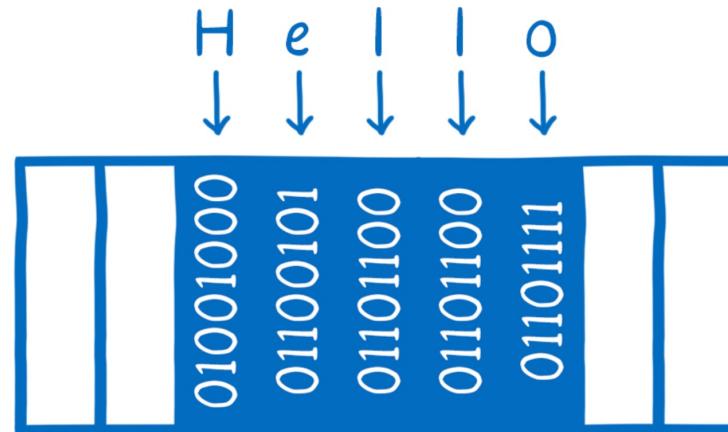
WEBASSEMBLY

SWETUGS

@nielstanis@infosec.exchange

# WebAssembly Memory

- Isolated per WASM module
- A contiguous, mutable array of uninterpreted bytes



SWETUG6

@nielstanis@infosec.exchange

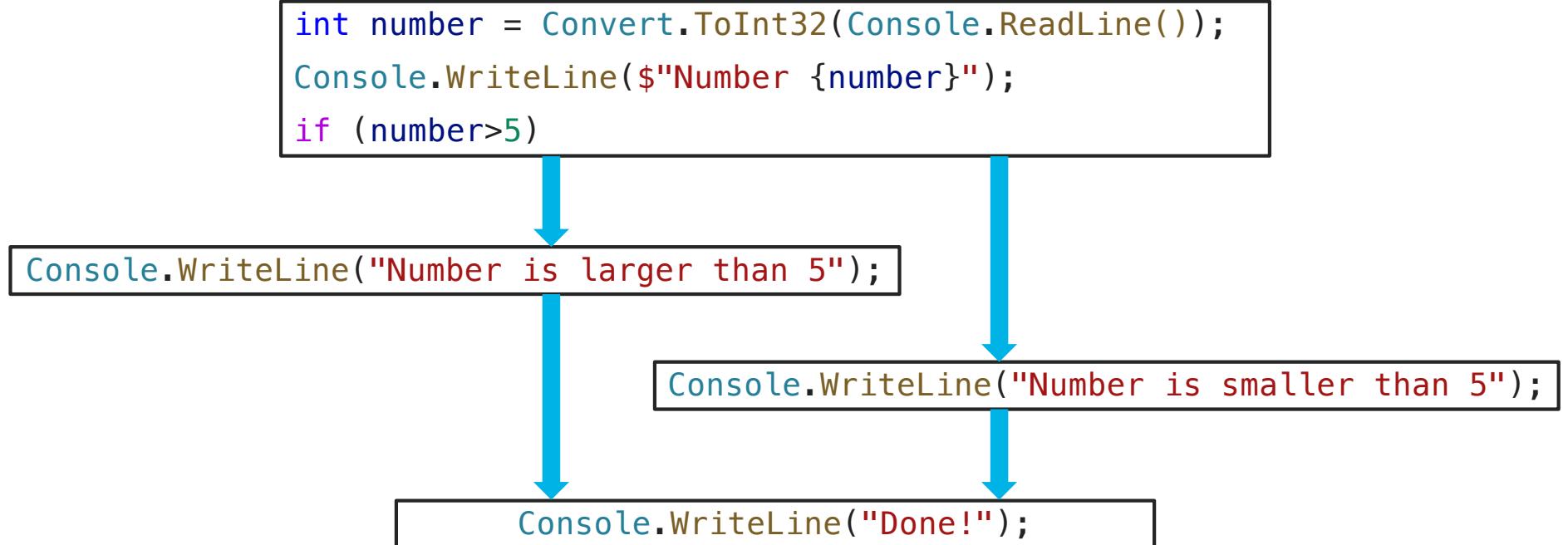
# WebAssembly Control-Flow Integrity

```
int number = Convert.ToInt32(Console.ReadLine());
Console.WriteLine($"Number {number}");
if (number>5)
{
    Console.WriteLine("Number is larger than 5");
}
else
{
    Console.WriteLine("Number is smaller than 5");
}
Console.WriteLine("Done!");
```

SWETUG6

Ⓜ @nielstanis@infosec.exchange

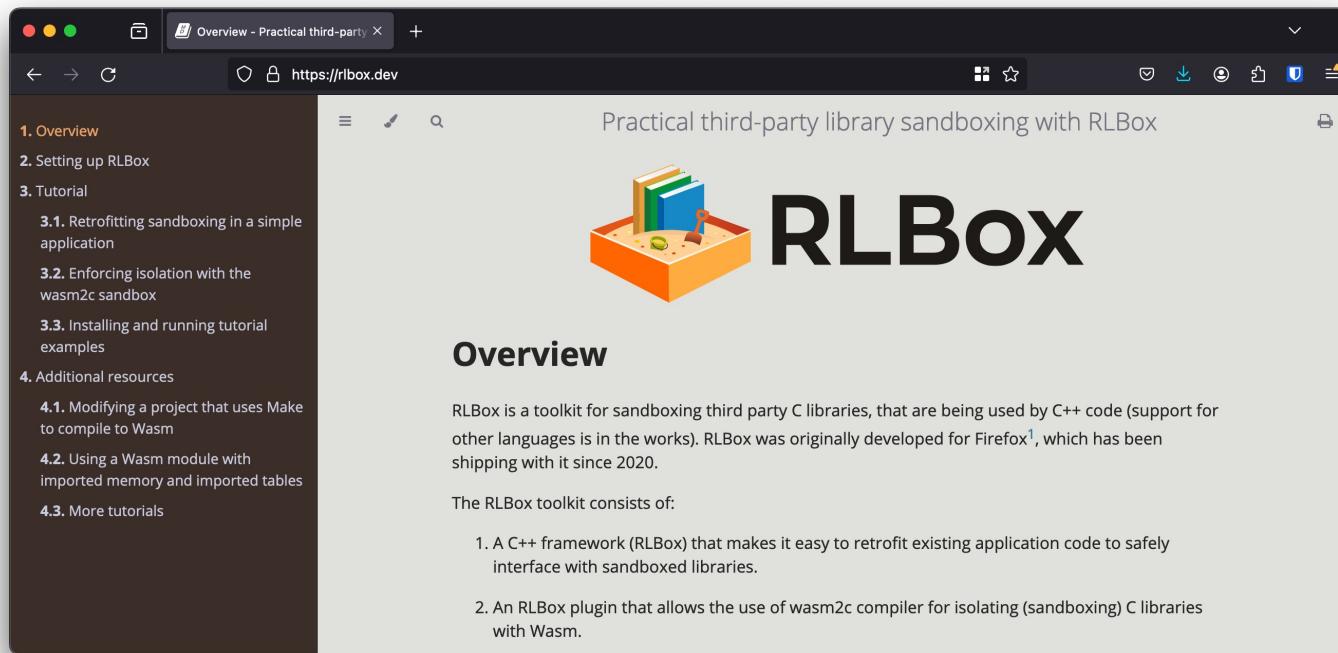
# WebAssembly Control-Flow Integrity



SWETUGS

@nielstanis@infosec.exchange

# FireFox RLBox



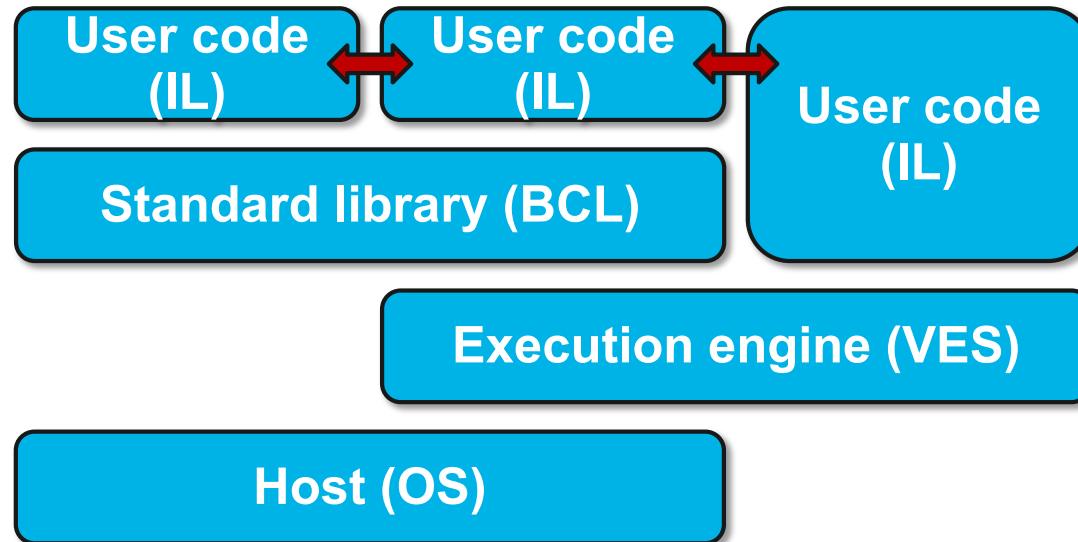
The screenshot shows a Firefox browser window with the URL <https://rlbox.dev>. The page title is "Overview - Practical third-party". The main content area features the text "Practical third-party library sandboxing with RLBox" and a logo of a yellow sandcastle containing three books. Below the logo, the word "RLBox" is written in large, bold, black letters. The left sidebar contains a navigation menu:

- 1. Overview
- 2. Setting up RLBox
- 3. Tutorial
  - 3.1. Retrofitting sandboxing in a simple application
  - 3.2. Enforcing isolation with the wasm2c sandbox
  - 3.3. Installing and running tutorial examples
- 4. Additional resources
  - 4.1. Modifying a project that uses Make to compile to Wasm
  - 4.2. Using a Wasm module with imported memory and imported tables
  - 4.3. More tutorials

SWETUG6

@nielstanis@infosec.exchange

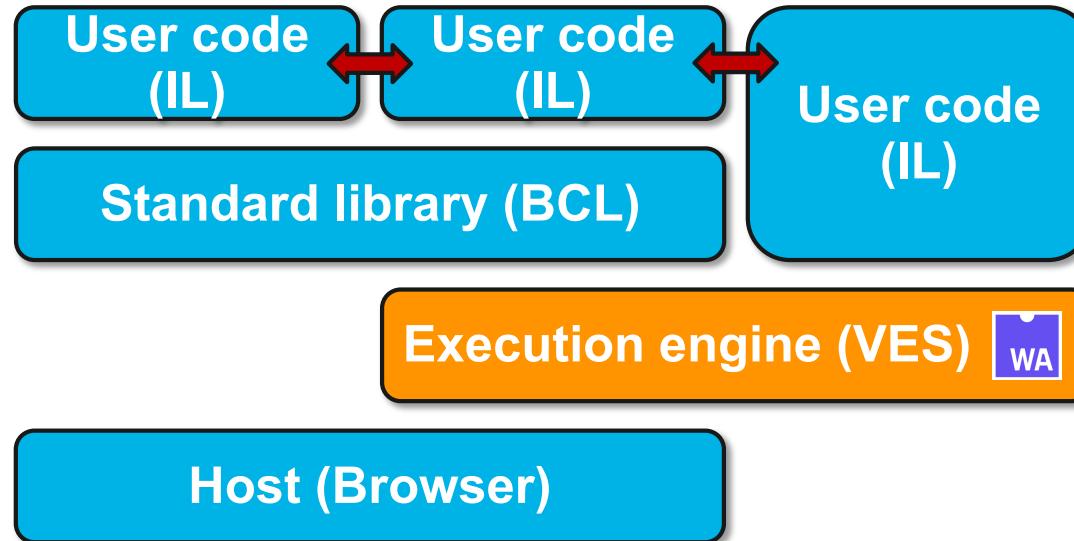
# Running .NET on WebAssembly



SWETUGS

@nielstanis@infosec.exchange

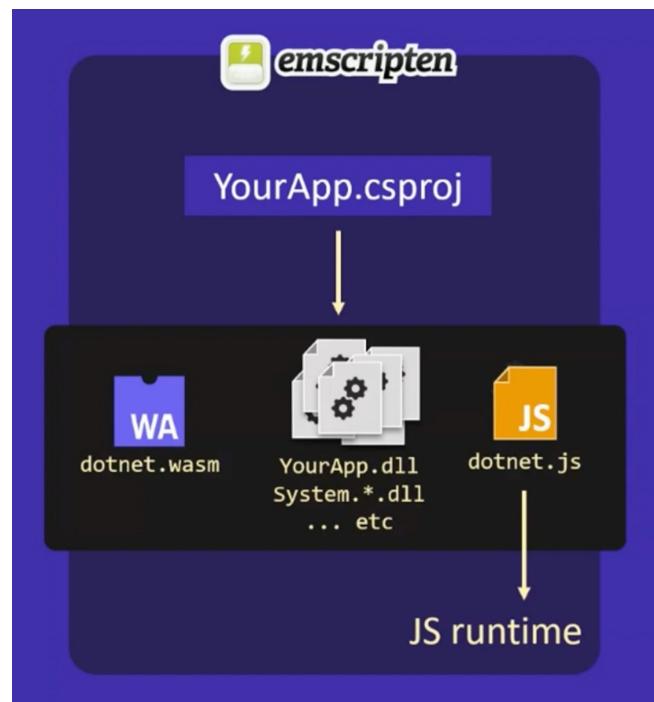
# Running .NET on WebAssembly



SWETUGS

@nielstanis@infosec.exchange

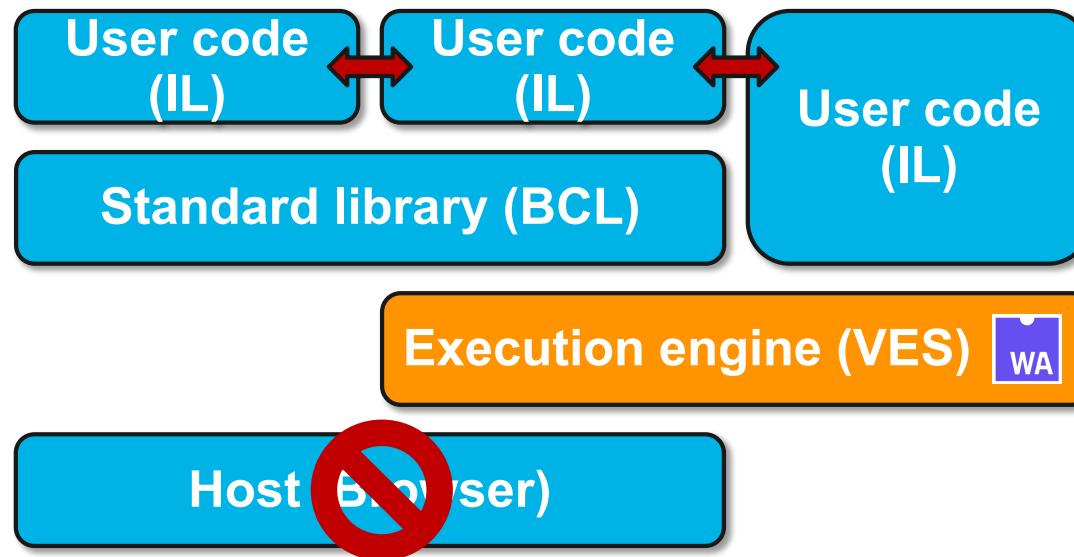
# Blazor WebAssembly



SWETUGS

@nielstanis@infosec.exchange

# Running .NET on WebAssembly



SWETUGS

@nielstanis@infosec.exchange

# WebAssembly System Interface WASI

- Introduced in March 2019 by Bytecode Alliance
- WasmTime implementation as reference
- POSIX inspired, engine-independent, non-Web system-oriented API for WebAssembly



SWETUGS

@nielstanis@infosec.exchange

# WebAssembly System Interface WASI

- Strong sandbox with Capability Based Security
- Preview1, supports e.g. FileSystem actions
- Preview2 supports a lot more!
- Anyone recall .NET Standard? ☺



SWETUGS

@nielstanis@infosec.exchange

# Docker vs WASM & WASI

The screenshot shows a Twitter interface on a dark-themed browser window. The main tweet is from **Solomon Hykes** (@solomonstre) at 9:39 p.m. on March 27, 2019, via the Twitter Web Client. The tweet reads:

If WASM+WASI existed in 2008, we wouldn't have needed to created Docker. That's how important it is. Webassembly on the server is the future of computing. A standardized system interface was the missing link. Let's hope WASI is up to the task!

Below the tweet is a reply from **Lin Clark** (@linclark) at 9:39 p.m. on March 27, 2019:

WebAssembly running outside the web has a huge future. And that future gets one giant leap closer today with...

Announcing WASI: A system interface for running WebAssembly outside the web (and inside it too)

[hacks.mozilla.org/2019/03/standa...](https://hacks.mozilla.org/2019/03/standa...)

Deze collectie weergeven

SWETUG6

@nielstanis@infosec.exchange

# Docker vs WASM & WASI

Twitter Tweet by Solomon Hykes (@solomonstre)

"So will wasm replace Docker?" No, but imagine a future where Docker runs linux containers, windows containers and wasm containers side by side. Over time wasm might become the most popular container type. Docker will love them all equally, and run it all :)

Tweet vertalen

Reply from Solomon Hykes (@solomonstre · 27 mrt. 2019): If WASM+WASI existed in 2008, we wouldn't have needed to created Docker. That's how important it is. Webassembly on the server is the future of computing. A standardized system interface was the missing link. Let's hope WASI is up to the task! [twitter.com/linclark/statu...](https://twitter.com/linclark/status/110380844000000000)

Deze collectie weergeven

4:50 a.m. · 28 mrt. 2019 · Twitter Web App

56 Retweets 5 Geciteerde Tweets 165 Vind-ik-leuks

SWETUG6

@nielstanis@infosec.exchange

# Docker & WASM

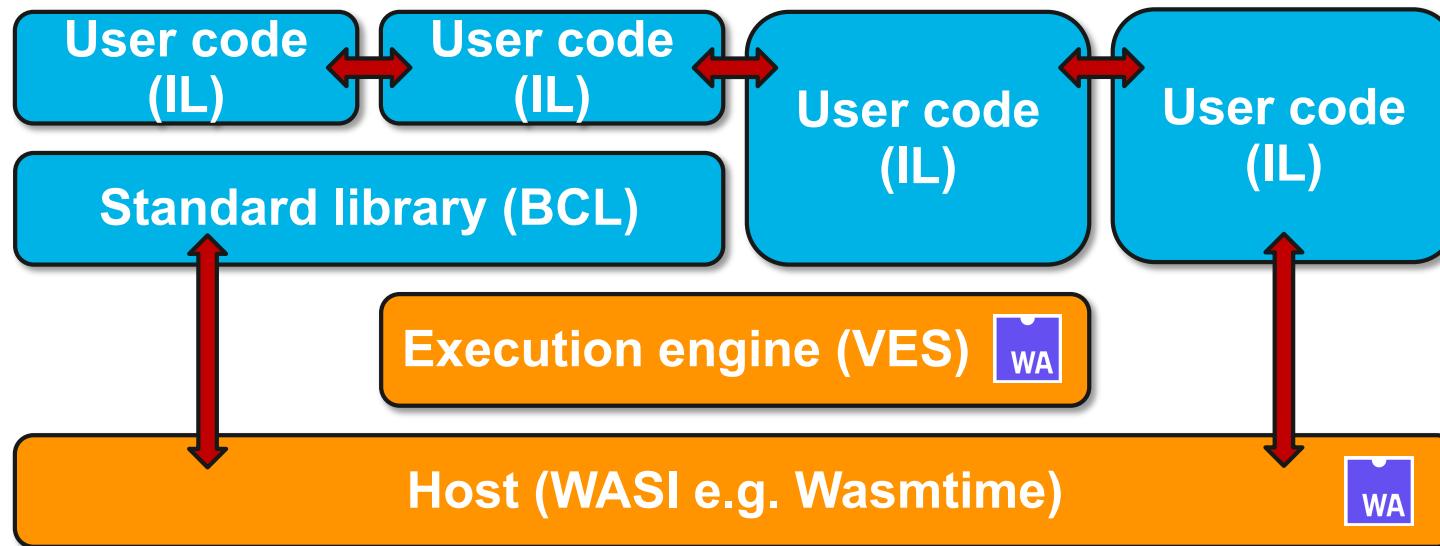
The screenshot shows a web browser window with the title "Introducing the Docker+Wasm Technical Preview". At the top, there's a purple banner with the text "Wasm is a fast, light alternative to Linux containers — try it out today in the [Docker+Wasm Technical Preview](#)". Below the banner is the Docker+Wasm logo. The main heading is "Introducing the Docker+Wasm Technical Preview". Below the heading is a photo of Michael Irwin, the author, with the text "MICHAEL IRWIN" and "Oct 24 2022". A paragraph at the bottom states: "The [Technical Preview of Docker+Wasm](#) is now available! Wasm has been producing a lot of buzz recently, and this feature will make it easier for you to quickly build applications targeting Wasm runtimes."

The screenshot shows a diagram titled "Introducing the Docker+Wasm Technical Preview" with a URL "https://www.docker.com/wasm". The diagram illustrates the Docker Engine architecture for Docker+Wasm. It shows the Docker Engine at the top, which interacts with a central "containerd" component. The "containerd" component then branches into three different runtime paths: "containerd-shim", "containerd-shim", and "containerd-wasm-shim". Each path leads to a "runc" process, which then leads to a "Container process". The "containerd-wasm-shim" path also includes a "wasmedge" component and a "Wasm Module". Arrows indicate the flow from the Docker Engine down through containerd to the specific shim and then to runc and finally the Container process.

SWETUG6

 @nielstanis@infosec.exchange

# WebAssembly System Interface WASI



SWETUGS

@nielstanis@infosec.exchange

# Experimental WASI SDK for .NET



SWETUGS

@nielstanis@infosec.exchange

# .NET 8 WASI-Experimental

The screenshot shows a web browser window with the Microsoft Dev Blogs URL: <https://devblogs.microsoft.com/dotnet/extending-web-assembly-to-the-cloud/>. The page content is as follows:

## wasi-experimental workload

.NET 8 includes a new workload called **wasi-experimental**. It builds on top of the Wasm functionality used by Blazor, extending it to run in **wasmtime** and invoke WASI interfaces. It is far from done, but already enables useful functionality.

Let's move on from theory to demonstrating the new capabilities.

After installing the [.NET 8 SDK](#), you can install the **wasi-experimental** workload.

```
dotnet workload install wasi-experimental
```

Note: This command may require admin permissions, for example with **sudo** on Linux and macOS.

You also need to install **wasmtime** to run the Wasm code you are soon going to produce.

Try a simple example with the **wasi-console** template.

```
$ dotnet new wasiconsole -o wasiconsole
$ cd wasiconsole
$ cat Program.cs
using System;

Console.WriteLine("Hello, WASI Console!");
```

SWETUGS

@nielstanis@infosec.exchange

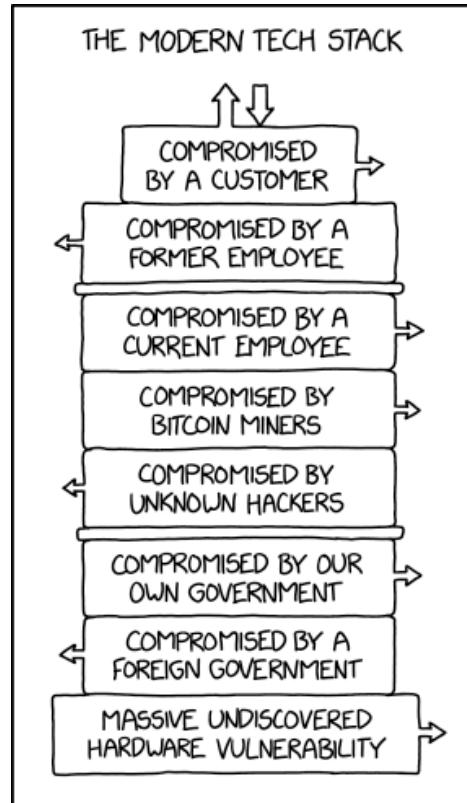
# Extending .NET with WASM

- WasmTime.NET NuGet package
- Can run WASM inside of any .NET application
- Extend with Rust based WASM module
- Limit capabilities
- Demo time!

SWETUGS

@nielstanis@infosec.exchange

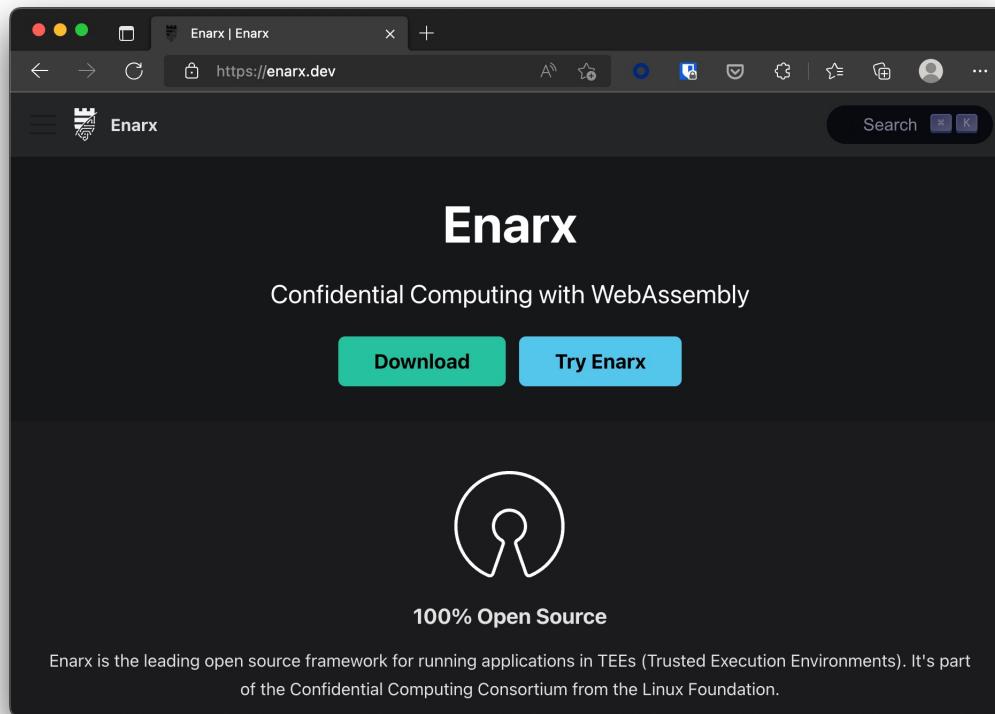
# Trusted Computing - XKCD 2166



SWETUG6

@nielstanis@infosec.exchange

# Enarx

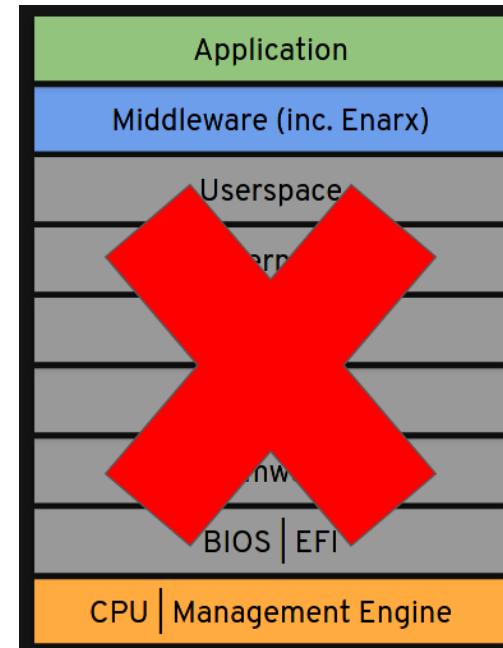


SWETUG6

@nielstanis@infosec.exchange

# Enarx Threat Model

- Don't trust the host
- Don't trust the host owner
- Don't trust the host operator
- Hardware cryptographically verified
- Software audited and cryptographically verified

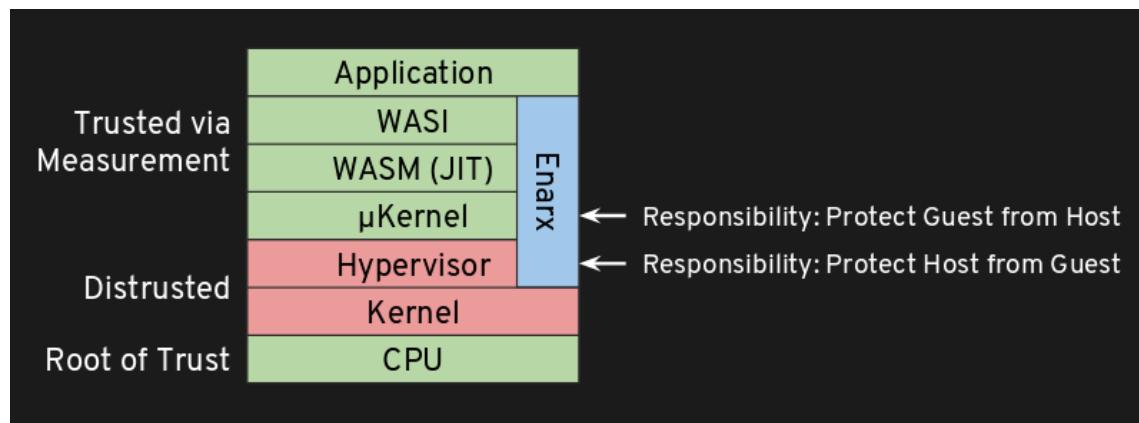


SWETUG6

@nielstanis@infosec.exchange

# Enarx

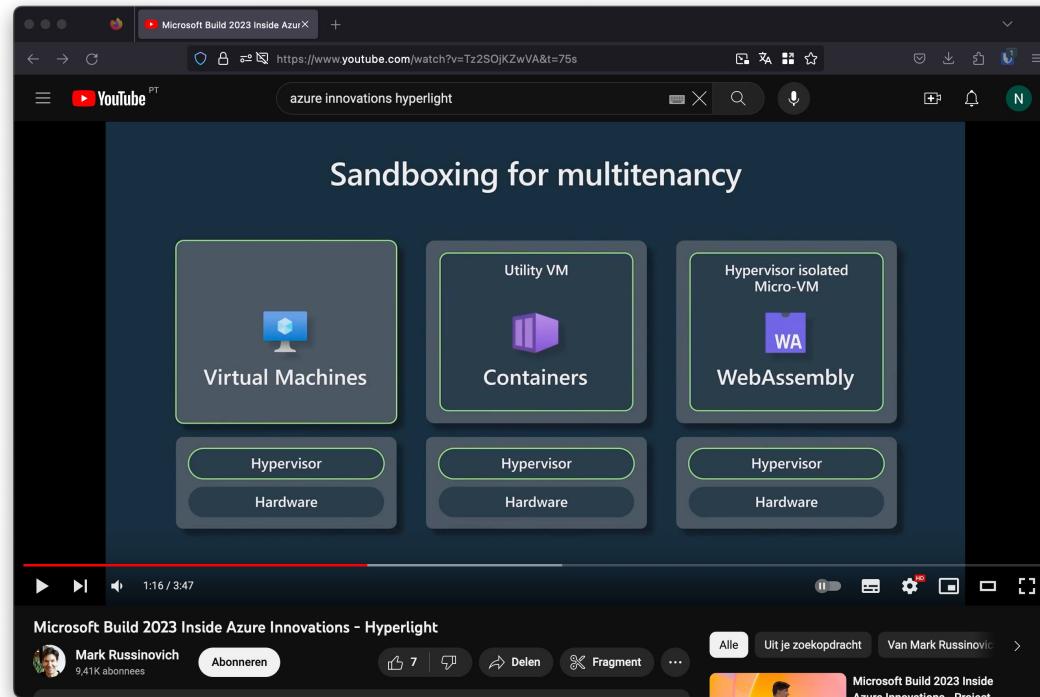
- Leverages Trusted Execution Environment (TEE) direct on processor
  - AMD's SEV, Intel's SGX and IBM's PEF
- Attestation of hardware and Enarx runtime



SWETUG6

@nielstanis@infosec.exchange

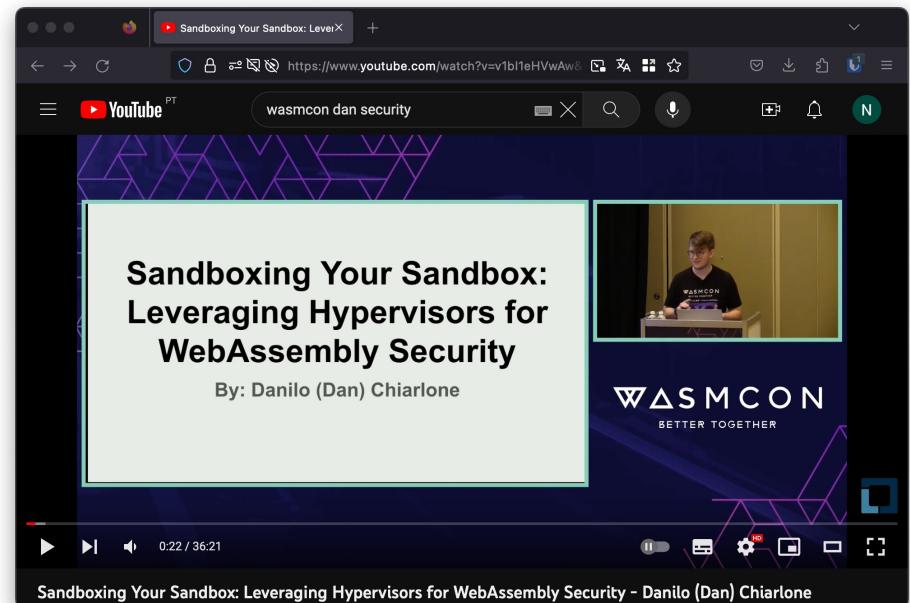
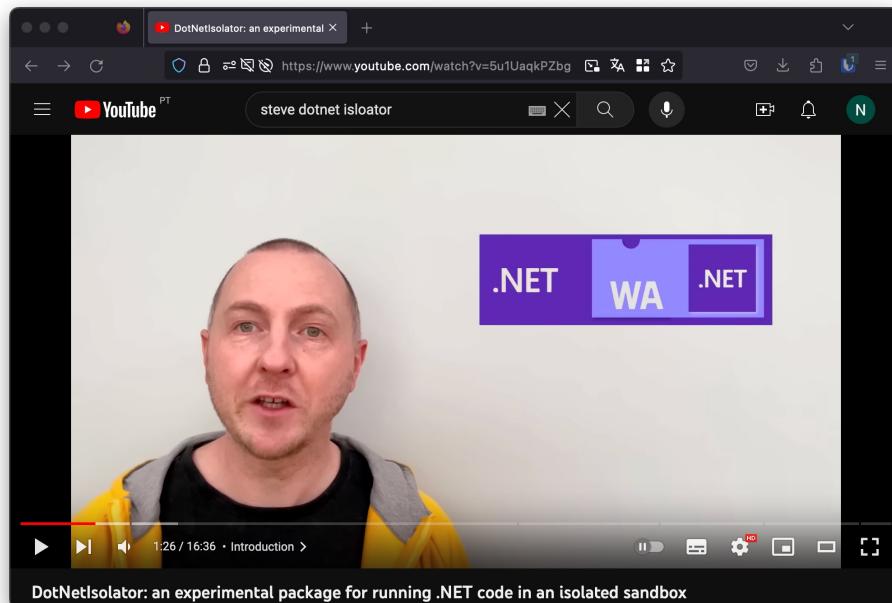
# Project Hyperlight



SWETUG6

@nielstanis@infosec.exchange

# DotNetIsolator & Project Hyperlight

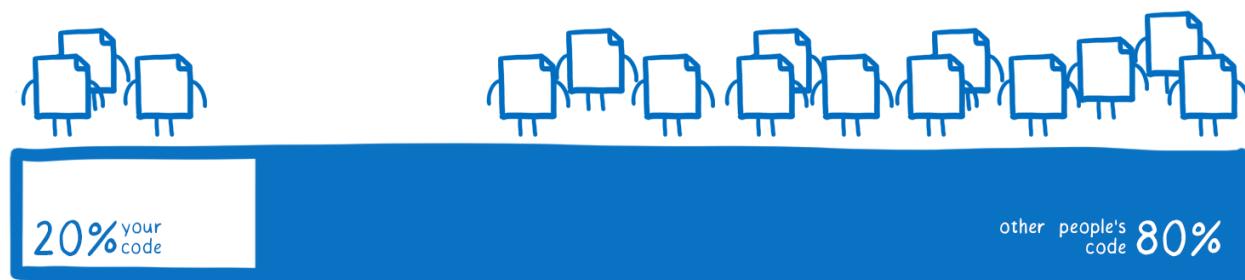


SWETUG6

@nielstanis@infosec.exchange

# WASM - What's next?

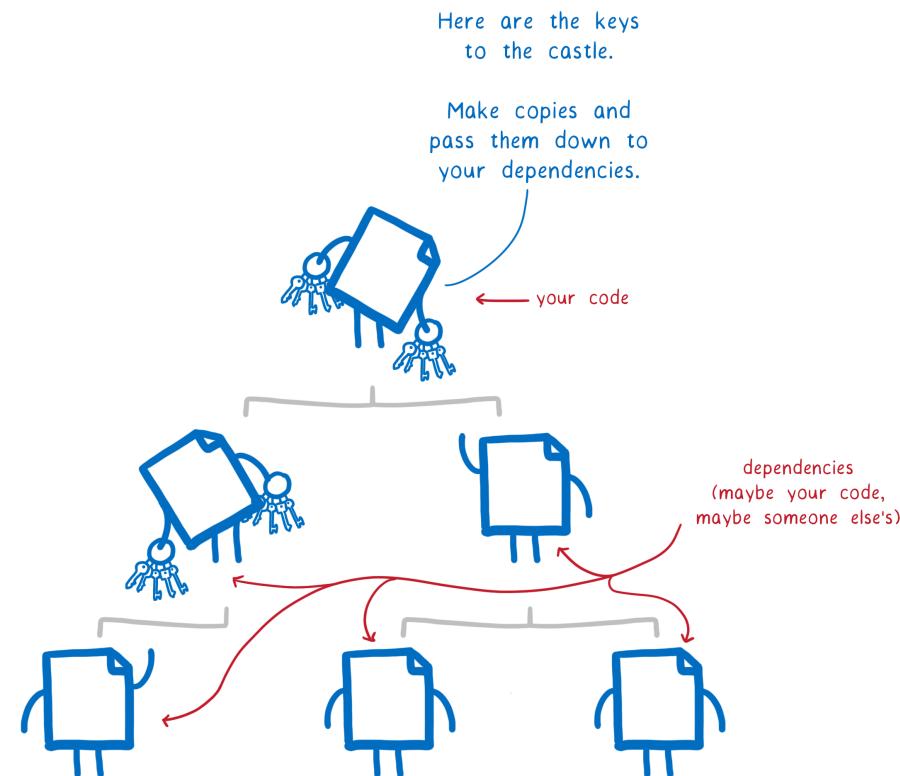
composition of an  
average code base



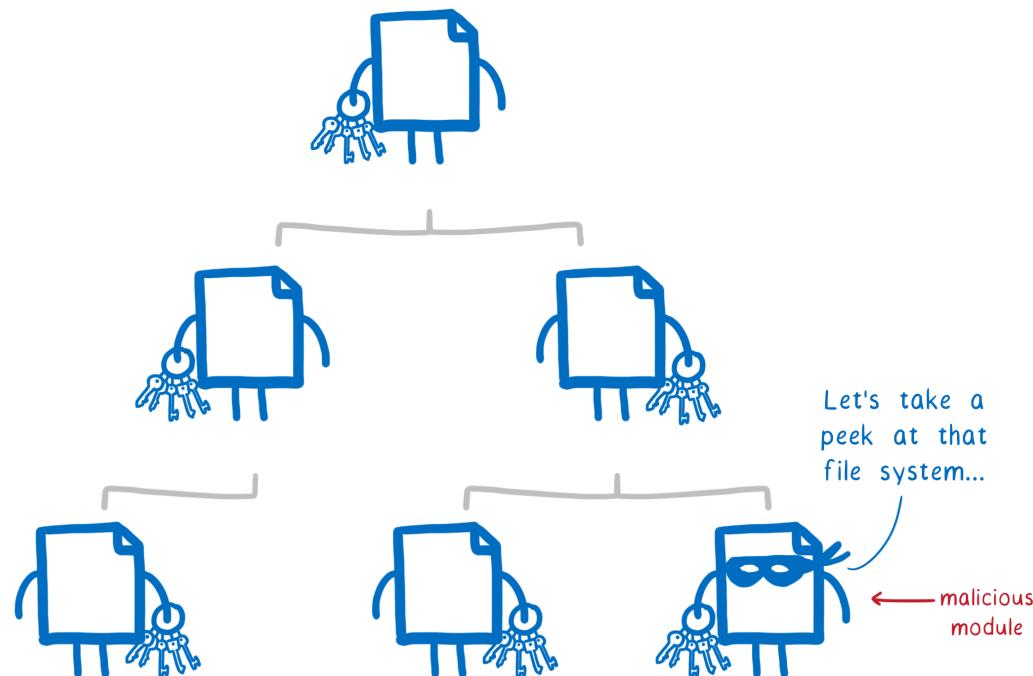
SWETUGS

@nielstanis@infosec.exchange

# Dependencies



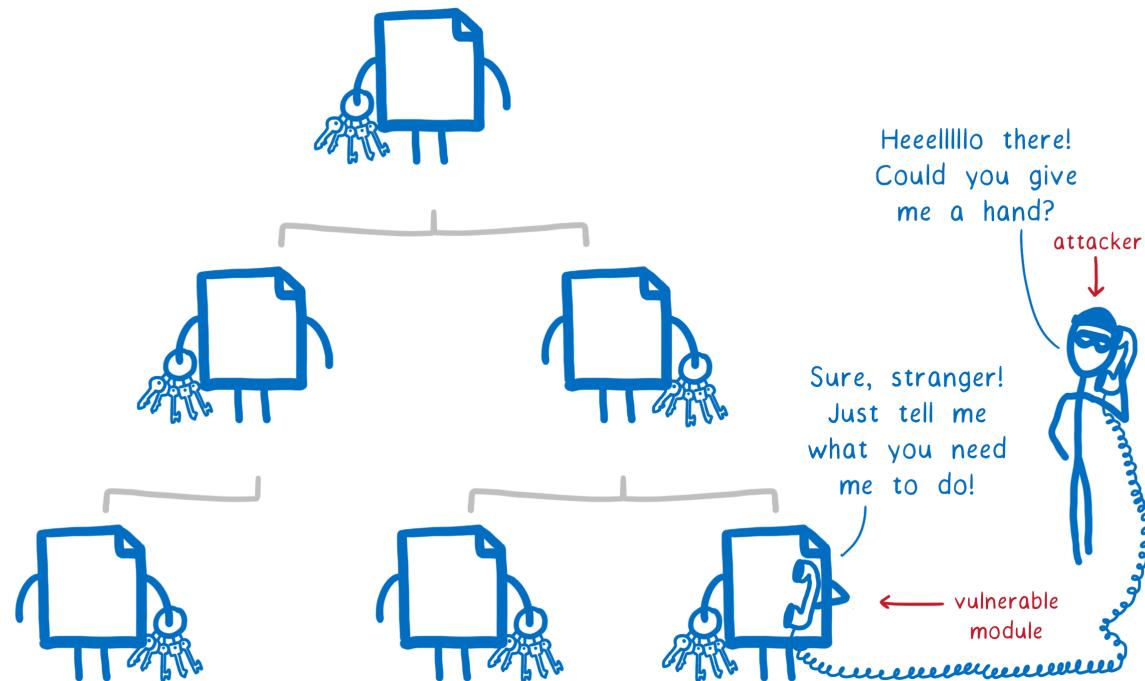
# Malicious module



SWETUG6

@nielstanis@infosec.exchange

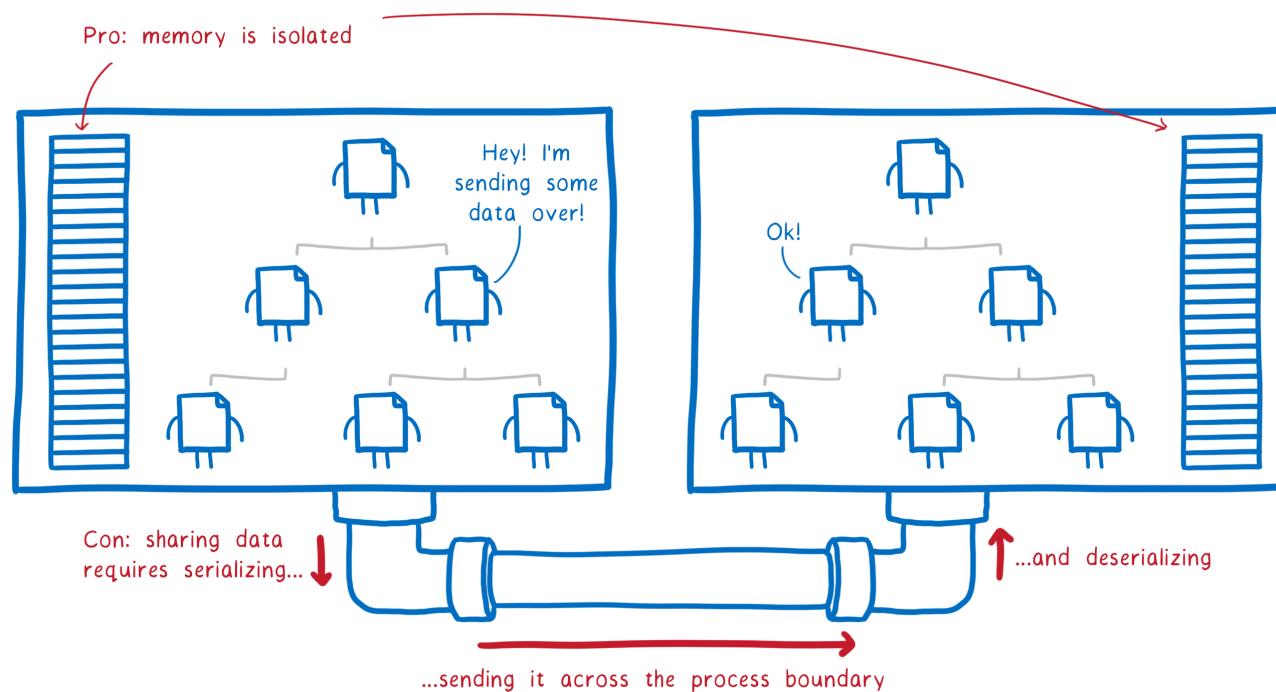
# Vulnerable module



SWETUG6

@nielstanis@infosec.exchange

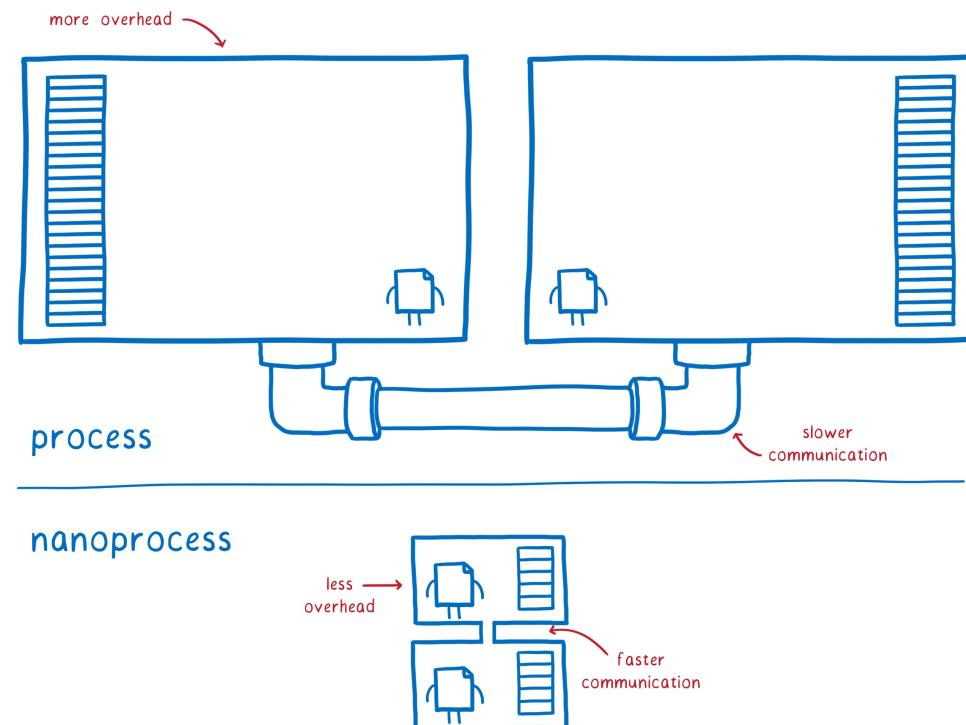
# Process Isolation



SWETUG6

@nielstanis@infosec.exchange

# WebAssembly Nano-Process



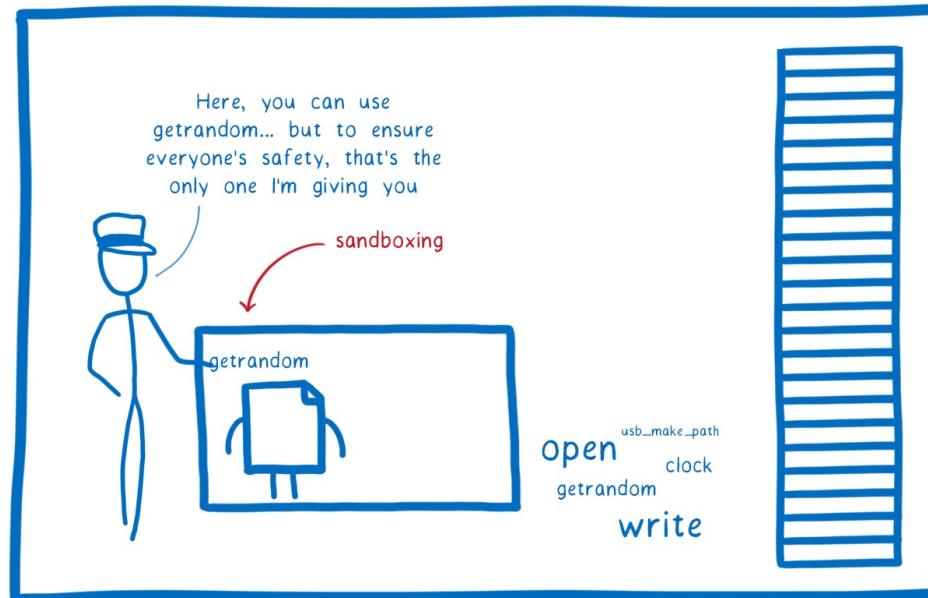
\* not drawn to scale

 @nielstanis@infosec.exchange

SWETUGS

# WebAssembly Nano-Process

## 1. Sandboxing

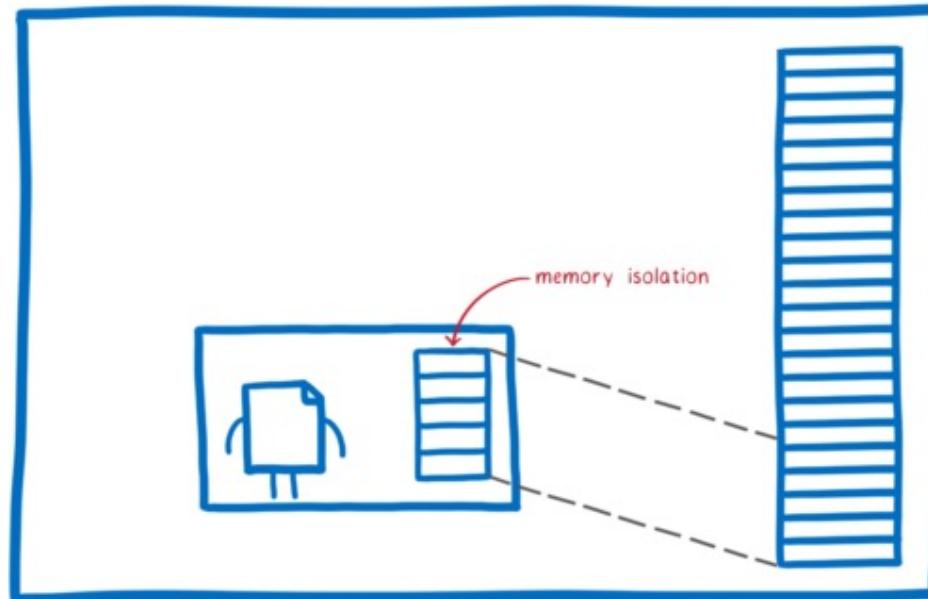


SWETUG6

@nielstanis@infosec.exchange

# WebAssembly Nano-Process

## 2. Memory model

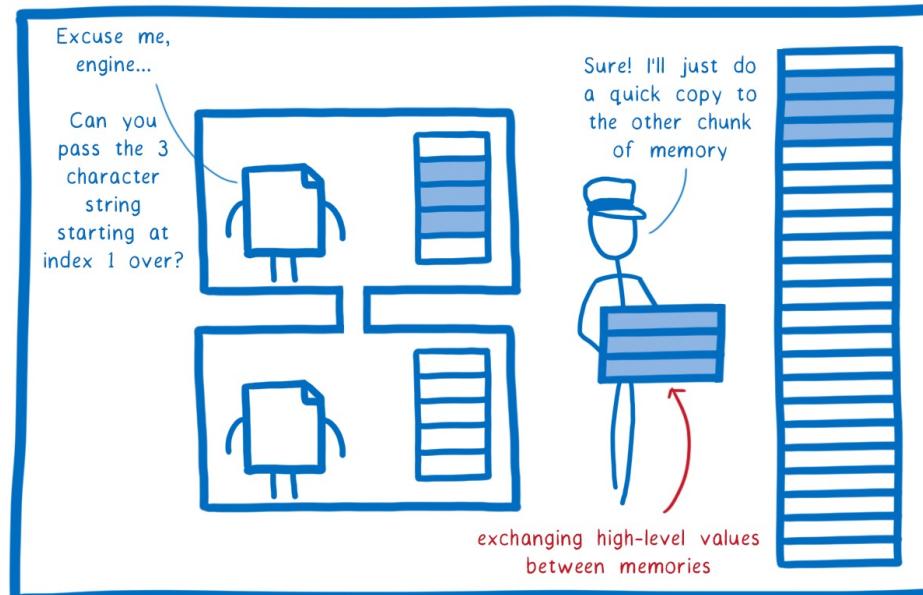


SWETUGS

@nielstanis@infosec.exchange

# WebAssembly Nano-Process

## 3. Interface Types

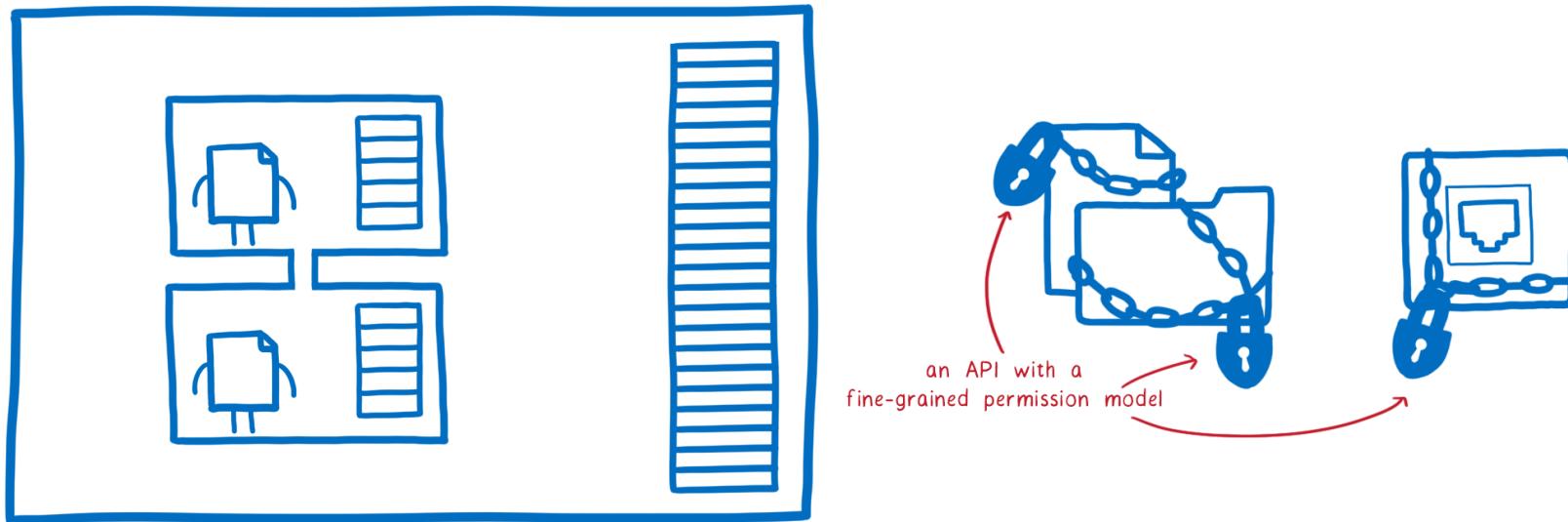


SWETUG6

@nielstanis@infosec.exchange

# WebAssembly Nano-Process

## 4. WebAssembly System Interface

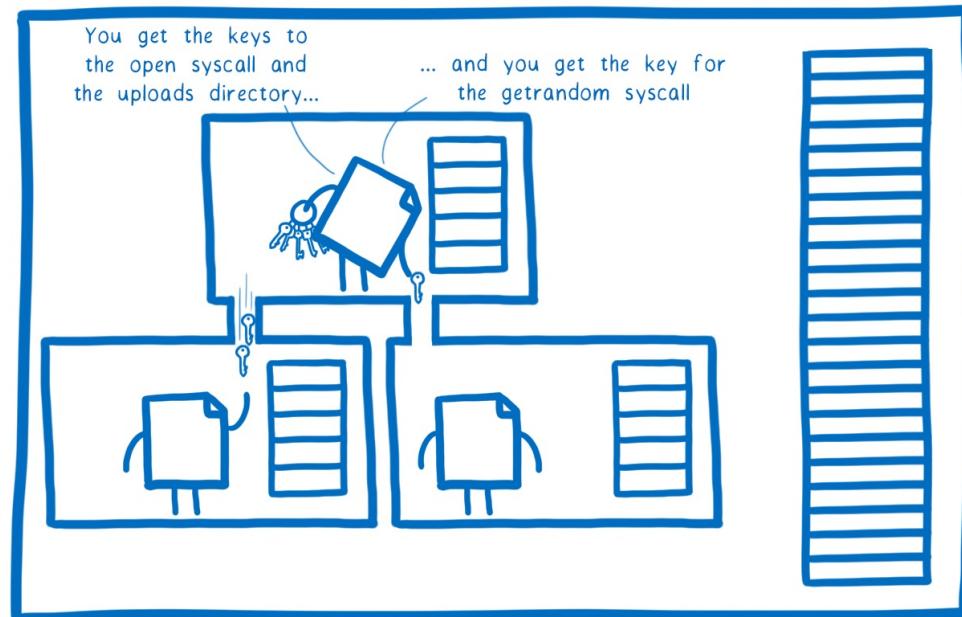


SWETUG6

@nielstanis@infosec.exchange

# WebAssembly Nano-Process

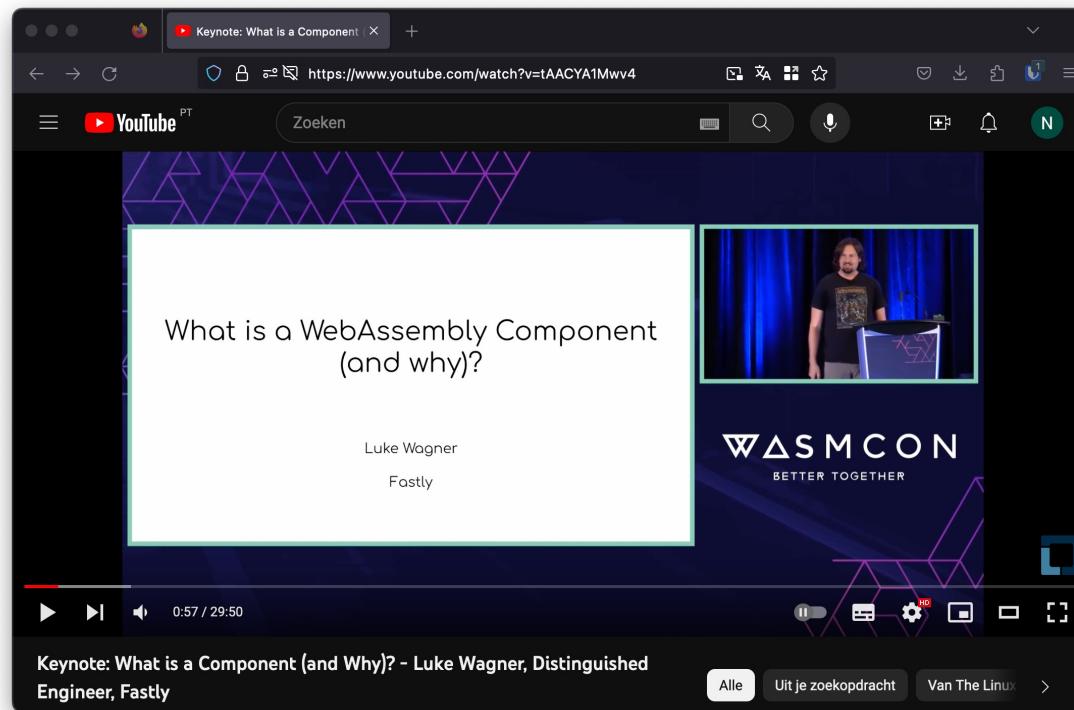
## 5. The missing link



SWETUG6

@nielstanis@infosec.exchange

# WebAssembly Component Model



SWETUG6

@nielstanis@infosec.exchange

# WASI Preview 2

The screenshot shows a web browser window with a dark theme. The title bar reads "WASI Preview 2 Launched - sunfishcode". The address bar shows the URL "https://blog.sunfishcode.online/wasi-preview2/". The page content is from "sunfishcode's blog" and is titled "A blog by sunfishcode". The main article is titled "WASI Preview 2 Launched" and was posted on January 25, 2024. The text discusses the launch of WASI Preview 2, the present state of development, and the future of the standard.

**WASI Preview 2 Launched**

Posted on January 25, 2024

The WASI Subgroup has just voted to launch WASI Preview 2! This blog post is a brief look at the present, past, and future of WASI.

**The present**

The Subgroup voted to launch Preview 2!

This is a major milestone! We made it! At the same time, the journey is only just beginning. But let's talk this moment to step back and look at what this means.

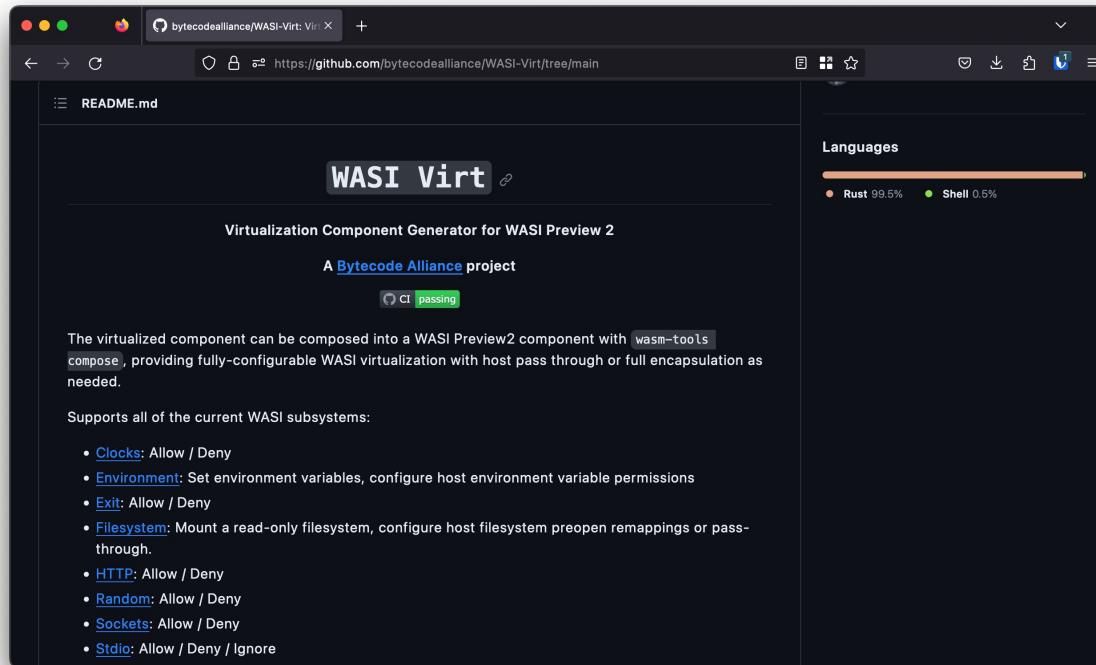
Most immediately, what this means is that the WASI Subgroup officially says that the Preview 2 APIs are stable. There is still a lot more to do, in writing more documentation, more tests, more toolchains, more implementations, and there are a lot more features that we all want to add. This vote today is a milestone along the way, rather than a destination in itself.

It also means that WASI is now officially based on the Wasm [component model](#), which makes it cross-language and virtualizable. Figuring out what a component model even is, designing it, implementing it, and building APIs using it has been a

SWETUGS

@nielstanis@infosec.exchange

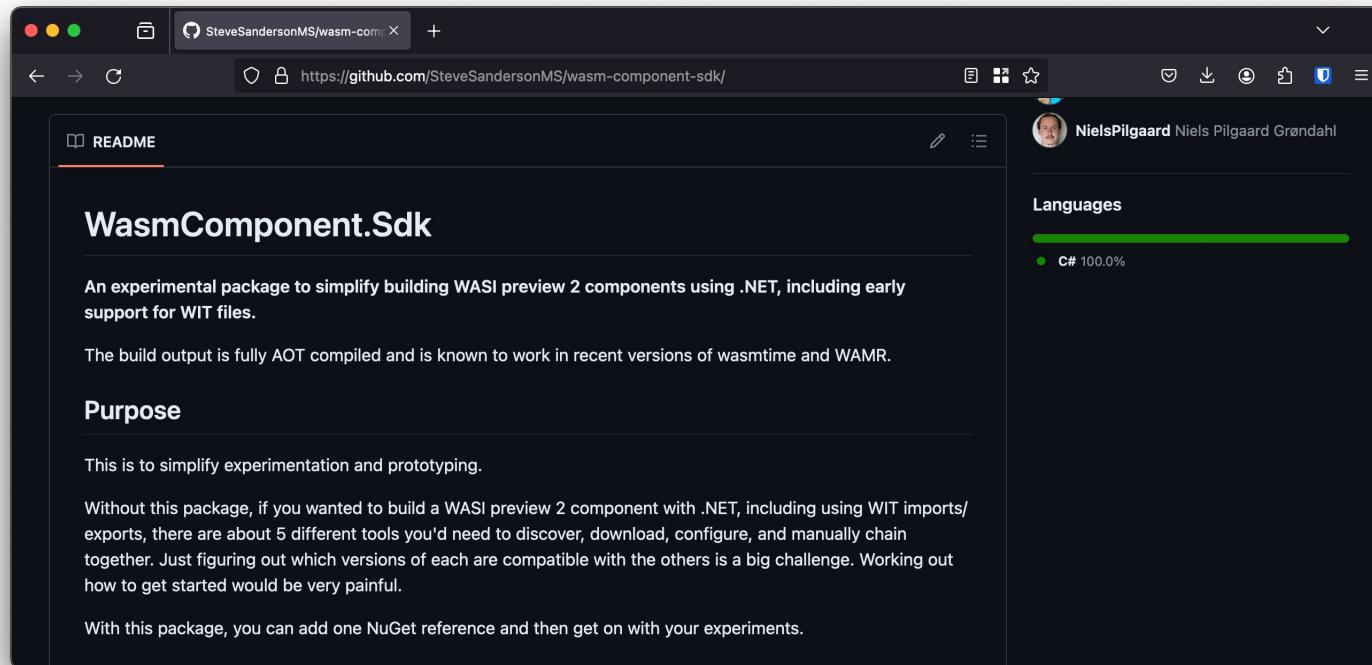
# WASI Virt



SWETUG6

@nielstanis@infosec.exchange

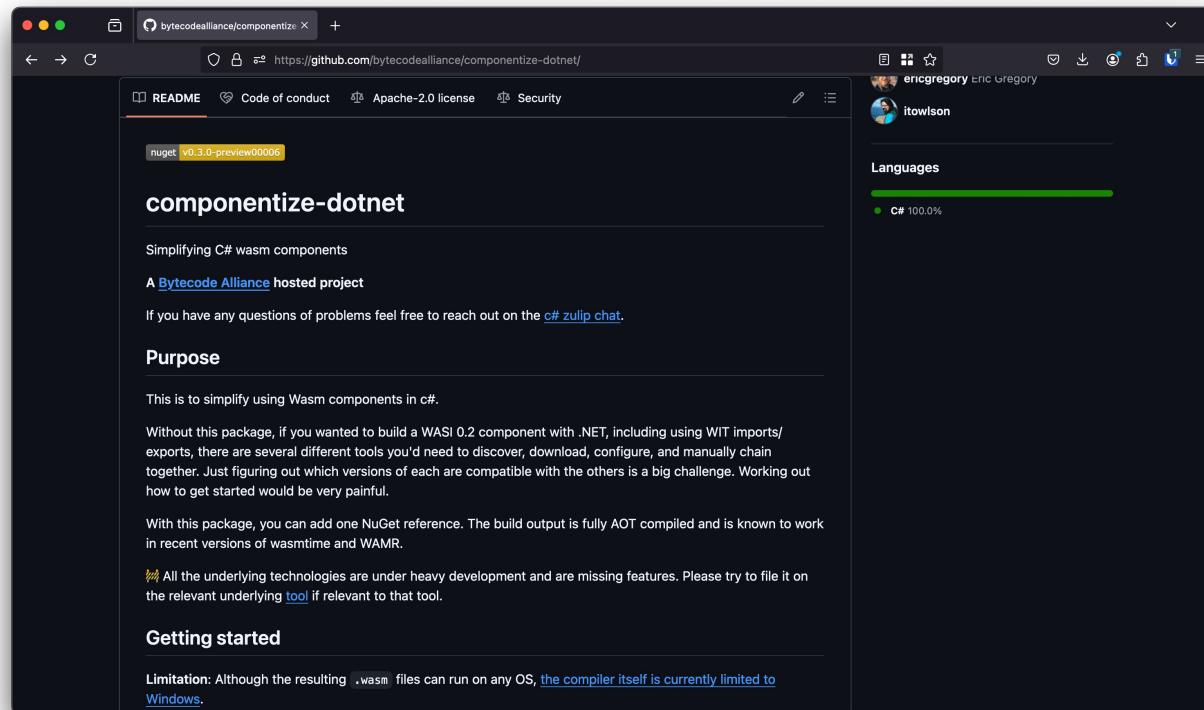
# WasmComponent.SDK



SWETUGS

@nielstanis@infosec.exchange

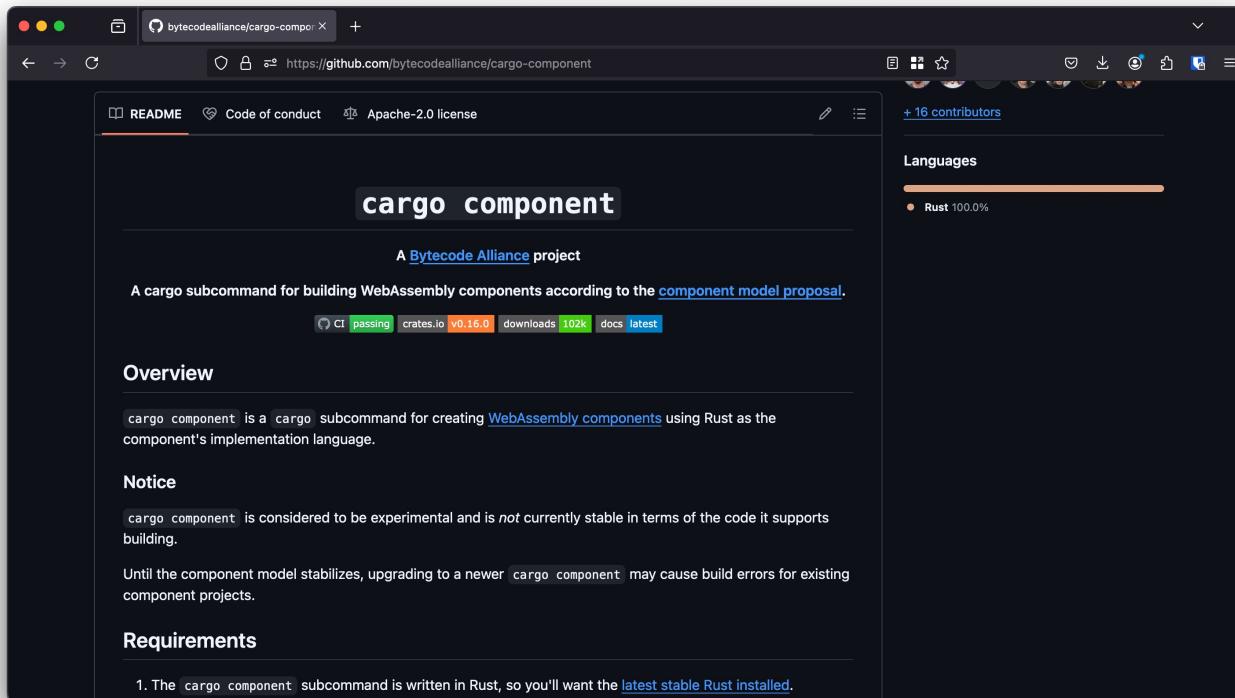
# Componentize-dotnet



SWETUG6

@nielstanis@infosec.exchange

# Demo WASI Preview 2 in Rust



SWETUGS

@nielstanis@infosec.exchange

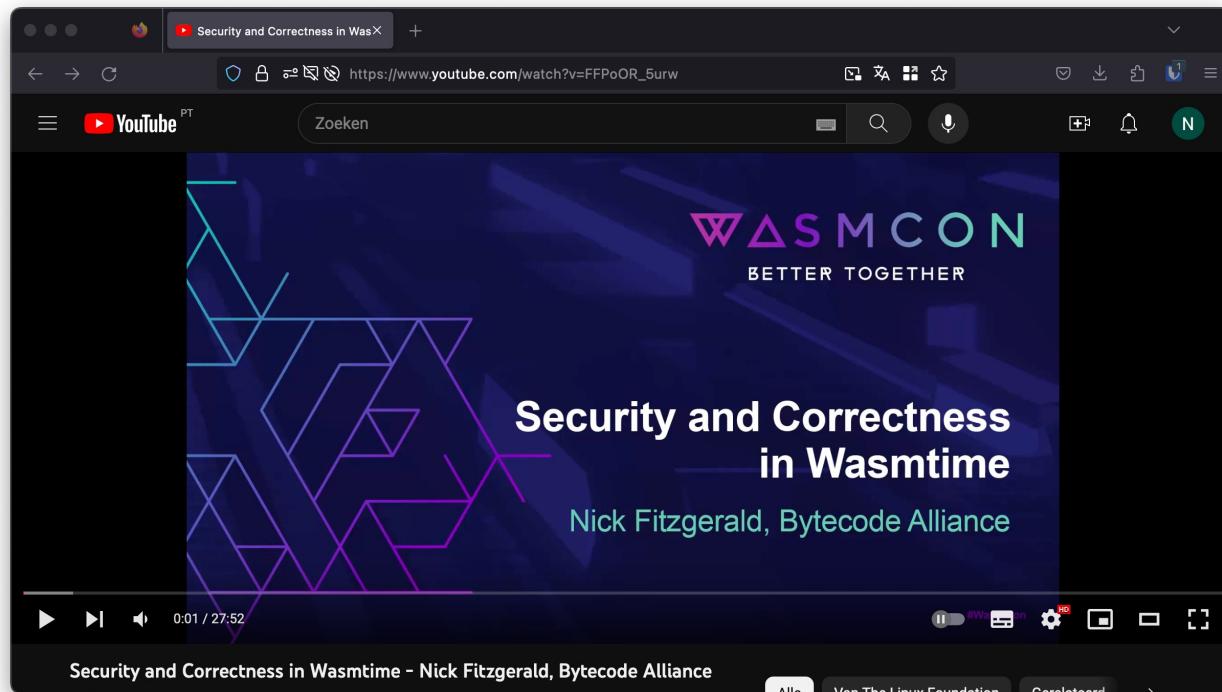
# Runtimes and Security

- Most security research published focusses on correctness of WASM runtimes/VM's
- Bytecode Alliance Blogpost September 2022:
  - "Security and Correctness in Wasmtime"
  - Written in Rust → Using all it's LangSec features
  - Continues Fuzzing & formal verification
  - Security process & vulnerability disclosure

SWETUG6

@nielstanis@infosec.exchange

# Runtimes and Security



SWETUG6

@nielstanis@infosec.exchange

# Conclusion

- Cloud Native ❤️ WebAssembly
- WebAssembly has a lot of potential to be used to run, extend, and secure your applications!
- Its as secure as the WebAssembly runtime implementation!
- WASI Preview 2 big milestone! componentize-dotnet good step forward for .NET ecosystem

SWETUGS

Ⓜ @nielstanis@infosec.exchange

# Questions?

- <https://github.com/nielstanis/swetugg2024wasm/>
- ntanis at Veracode.com
- @nielstanis@infosec.exchange
- <https://blog.fennec.dev>
- Tack! Thank you!

SWETUGG

✉️ @nielstanis@infosec.exchange