

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

Niels Tanis

VERACODE

0101  
0101

# 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





# 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



# WebAssembly

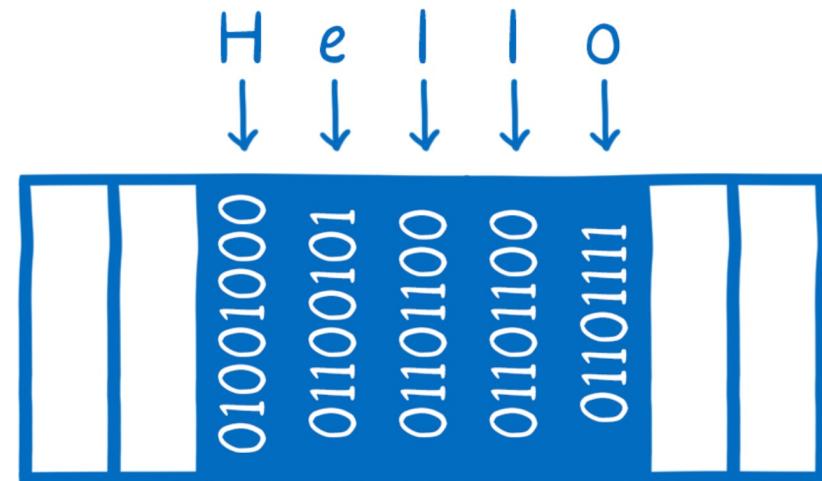
- Binary instruction format for stack-based virtual machine similar to .NET CLR running MSIL or JVM running bytecode
- Designed as a portable compilation target
- The security model of WebAssembly:
  - Protect users from buggy or malicious modules
  - Provide developers with useful primitives and mitigations for developing safe applications



0101  
0101

# WebAssembly Memory

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





# 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!");
```



# WebAssembly Control-Flow Integrity

```
int number = Convert.ToInt32(Console.ReadLine());  
Console.WriteLine($"Number {number}");  
if (number>5)
```

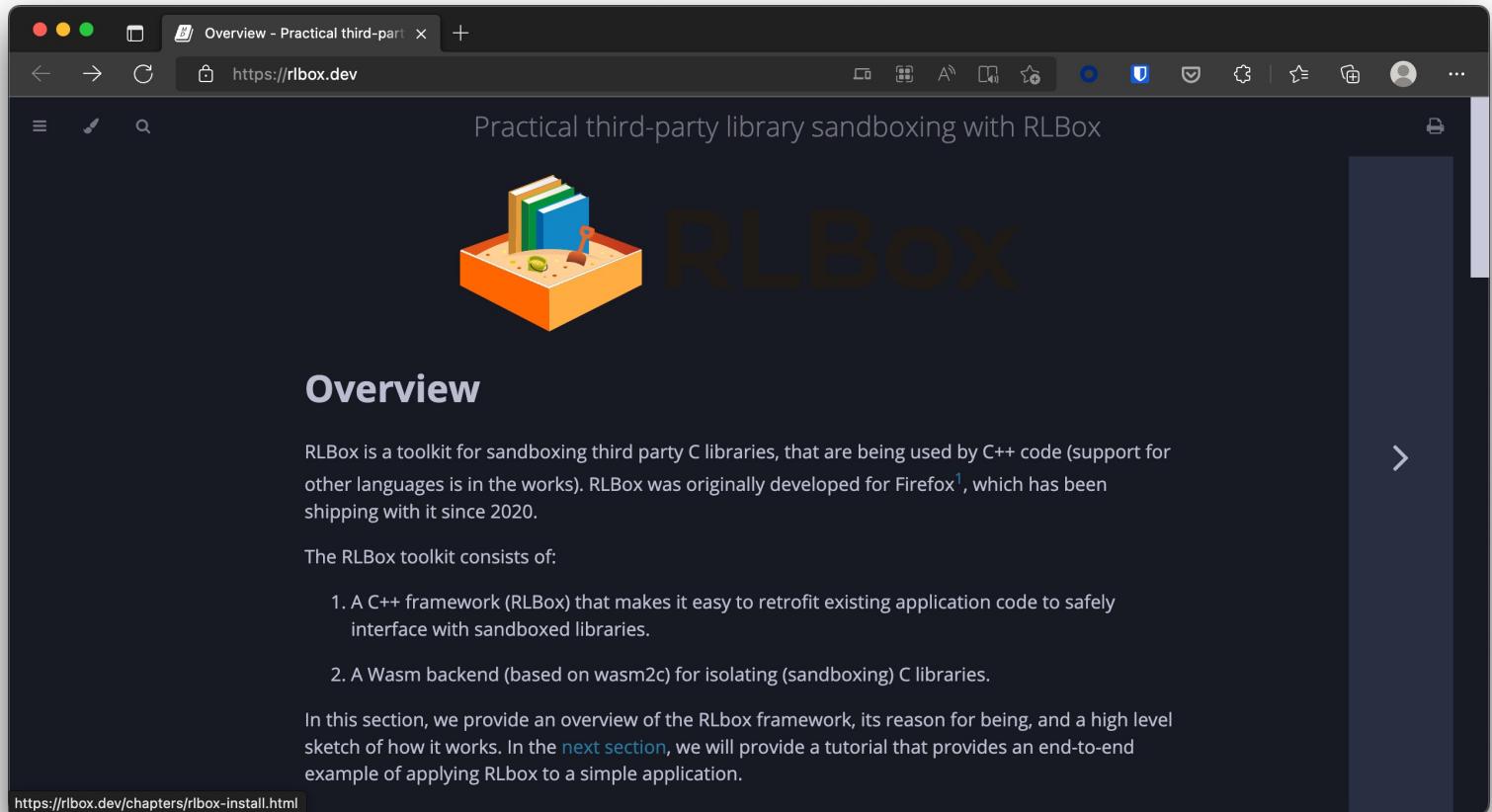
```
Console.WriteLine("Number is larger than 5");
```

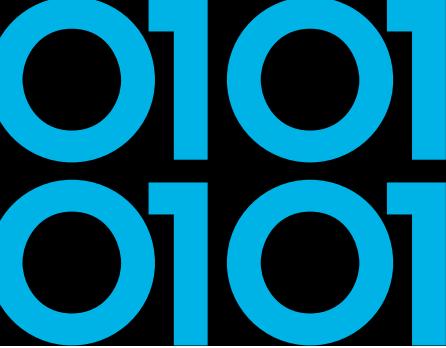
```
Console.WriteLine("Number is smaller than 5");
```

```
Console.WriteLine("Done!");
```

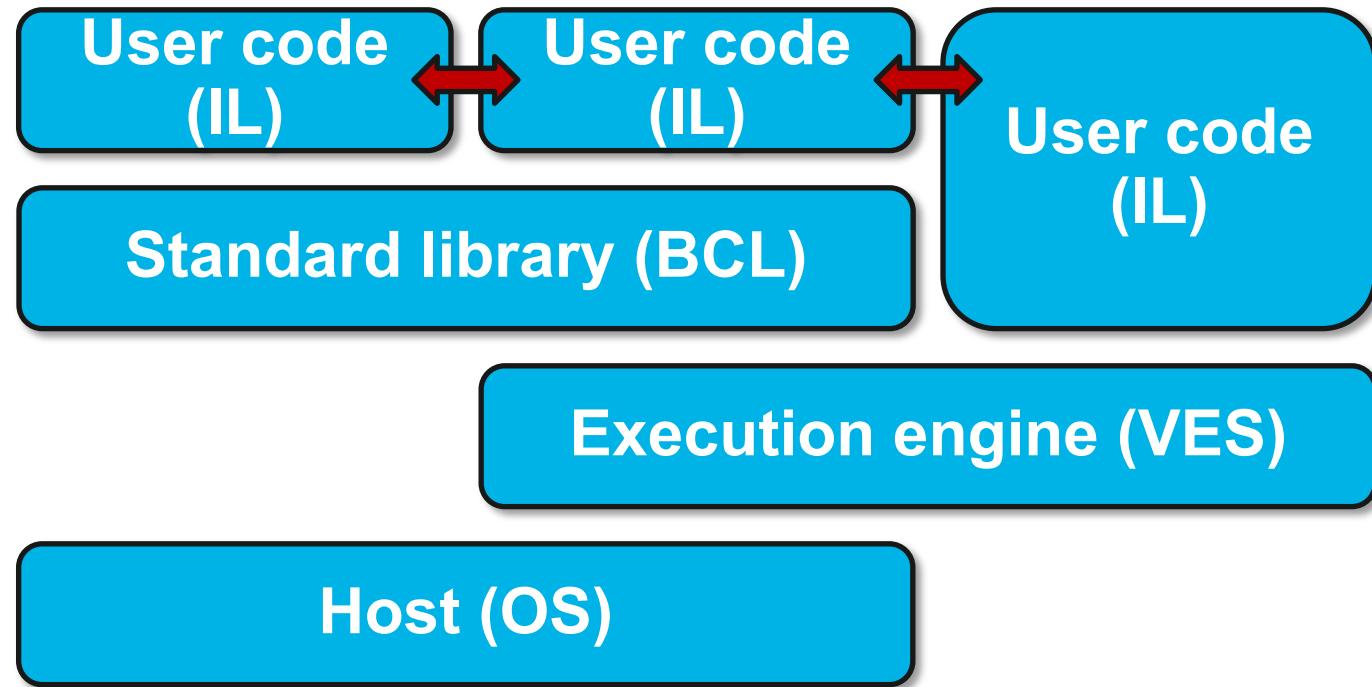
0101  
0101

# FireFox RLBox



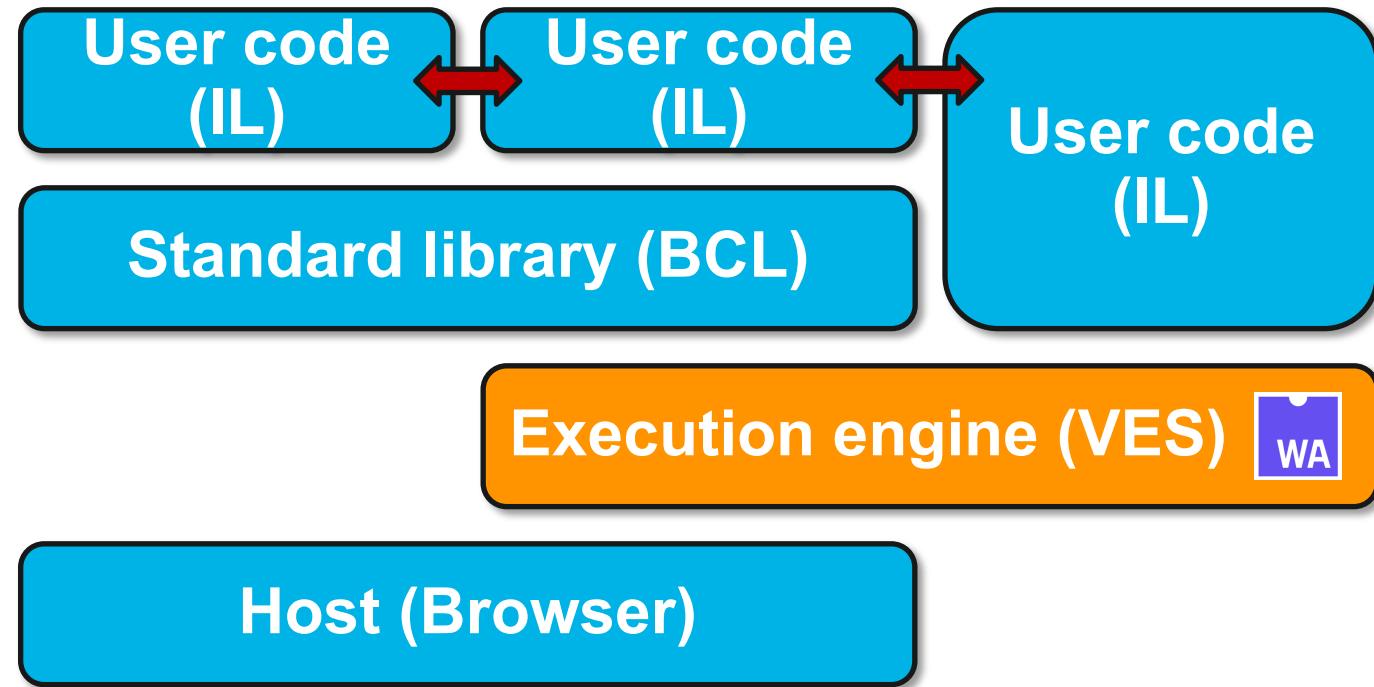


# Running .NET on WebAssembly



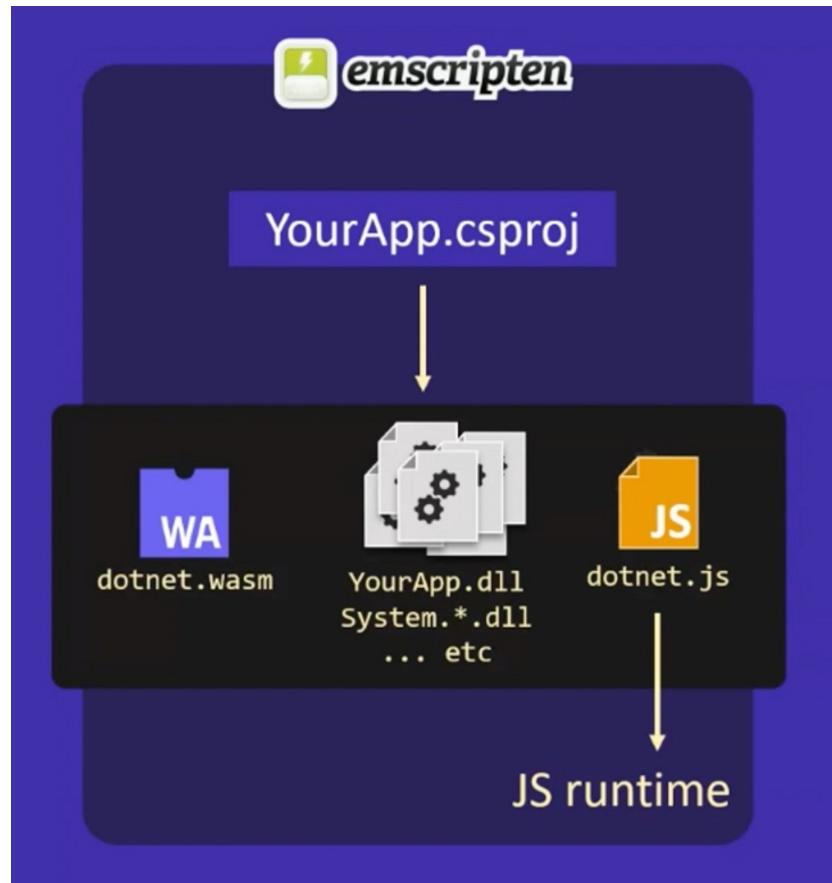


# Running .NET on WebAssembly



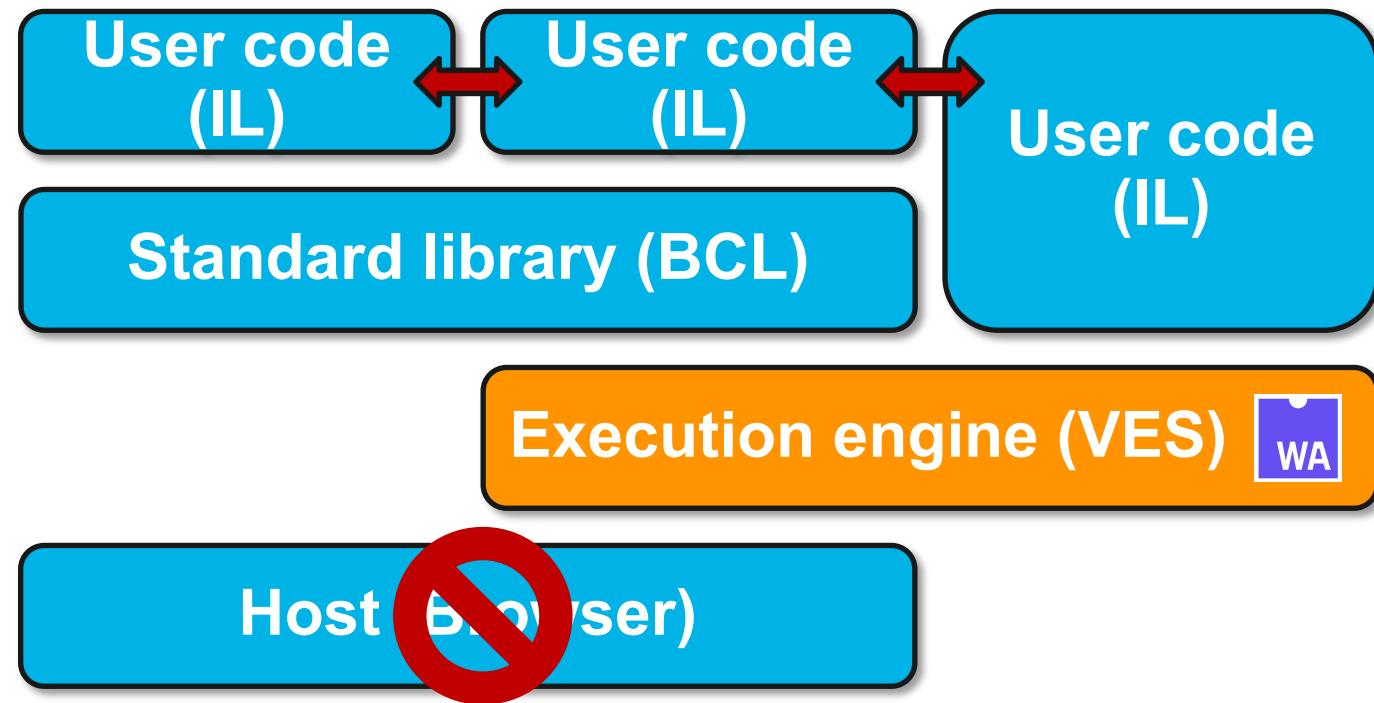
0101  
0101

# Blazor WebAssembly



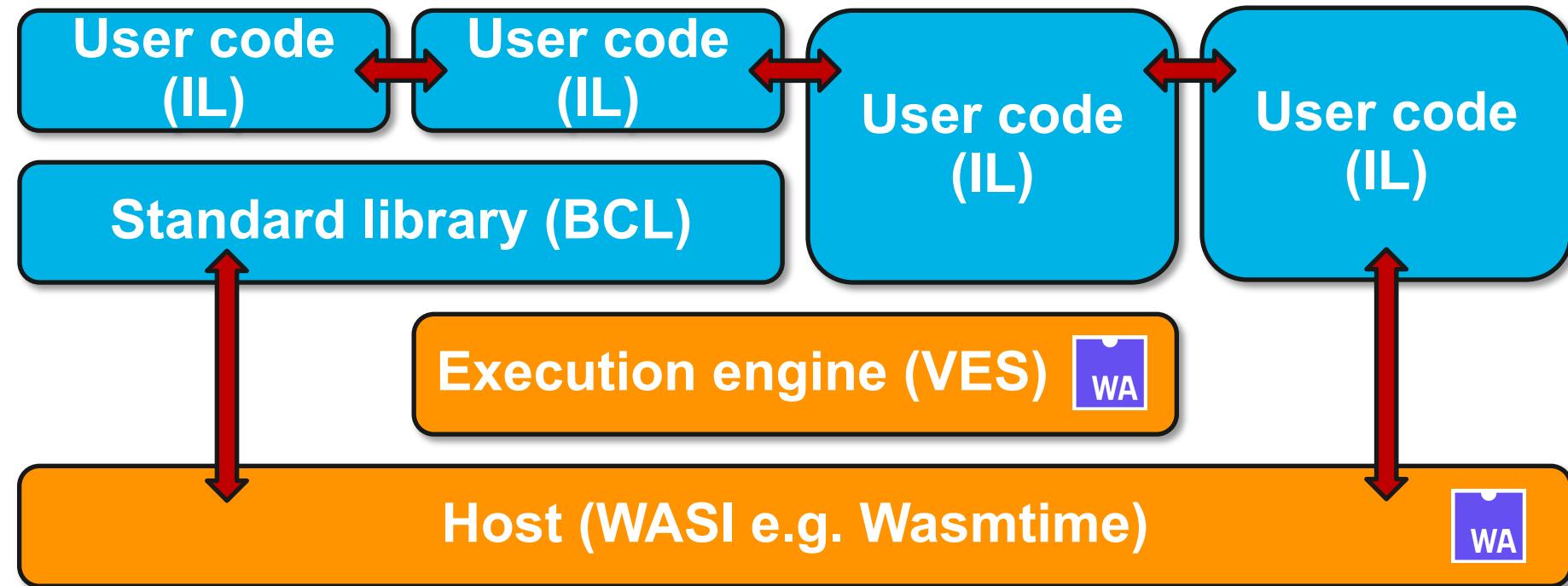


# Running .NET on WebAssembly





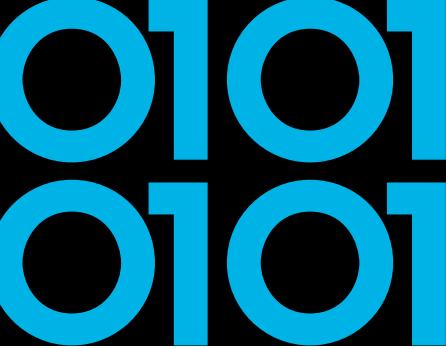
# WebAssembly System Interface WASI





# Experimental WASI SDK for .NET





# .NET 8 WASI-Experimental

The screenshot shows a web browser window displaying a Microsoft DevBlog post titled "Extending WebAssembly to the Cloud". The main heading is "wasi-experimental workload". The text explains that .NET 8 includes a new workload called "wasi-experimental" which builds on Wasm functionality used by Blazor, extending it to run in "wasmtime" and invoke WASI interfaces. It notes that while not fully developed, it already provides useful functionality.

Below the heading, there's a section titled "Let's move on from theory to demonstrating the new capabilities." It instructs users to install the ".NET 8 SDK" and then the "wasi-experimental" workload. A command-line snippet shows the user running "dotnet workload install wasi-experimental". A note states that this command may require admin permissions on Linux and macOS.

The post also mentions the need to install "wasmtime" to run the generated Wasm code. It encourages users to try a simple example using the "wasi-console" template, providing a command-line snippet:

```
$ dotnet new wasiconsole -o wasiconsole  
$ cd wasiconsole  
$ cat Program.cs  
using System;  
  
Console.WriteLine("Hello, WASI Console!");
```

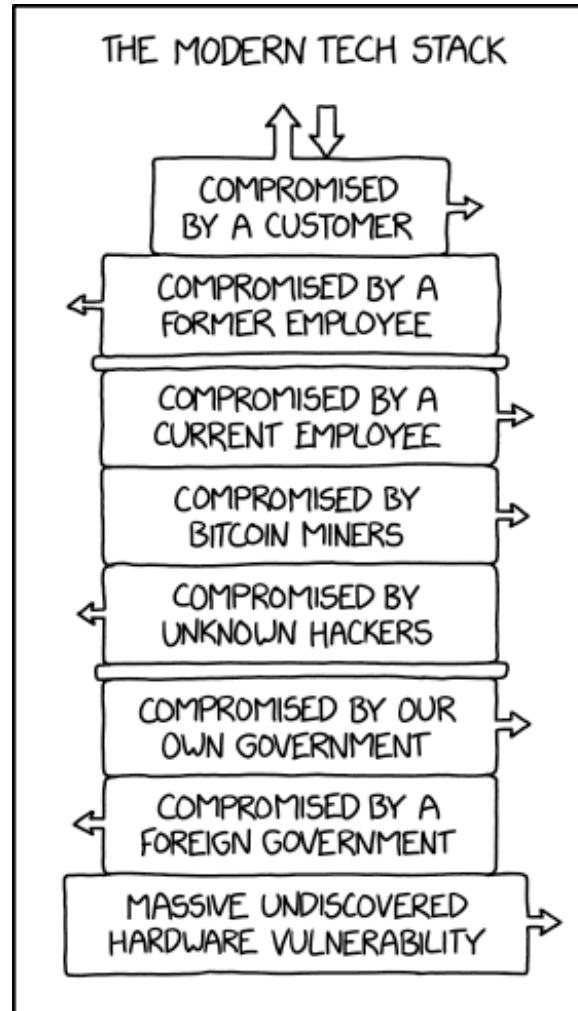


# Extending .NET with WASM

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

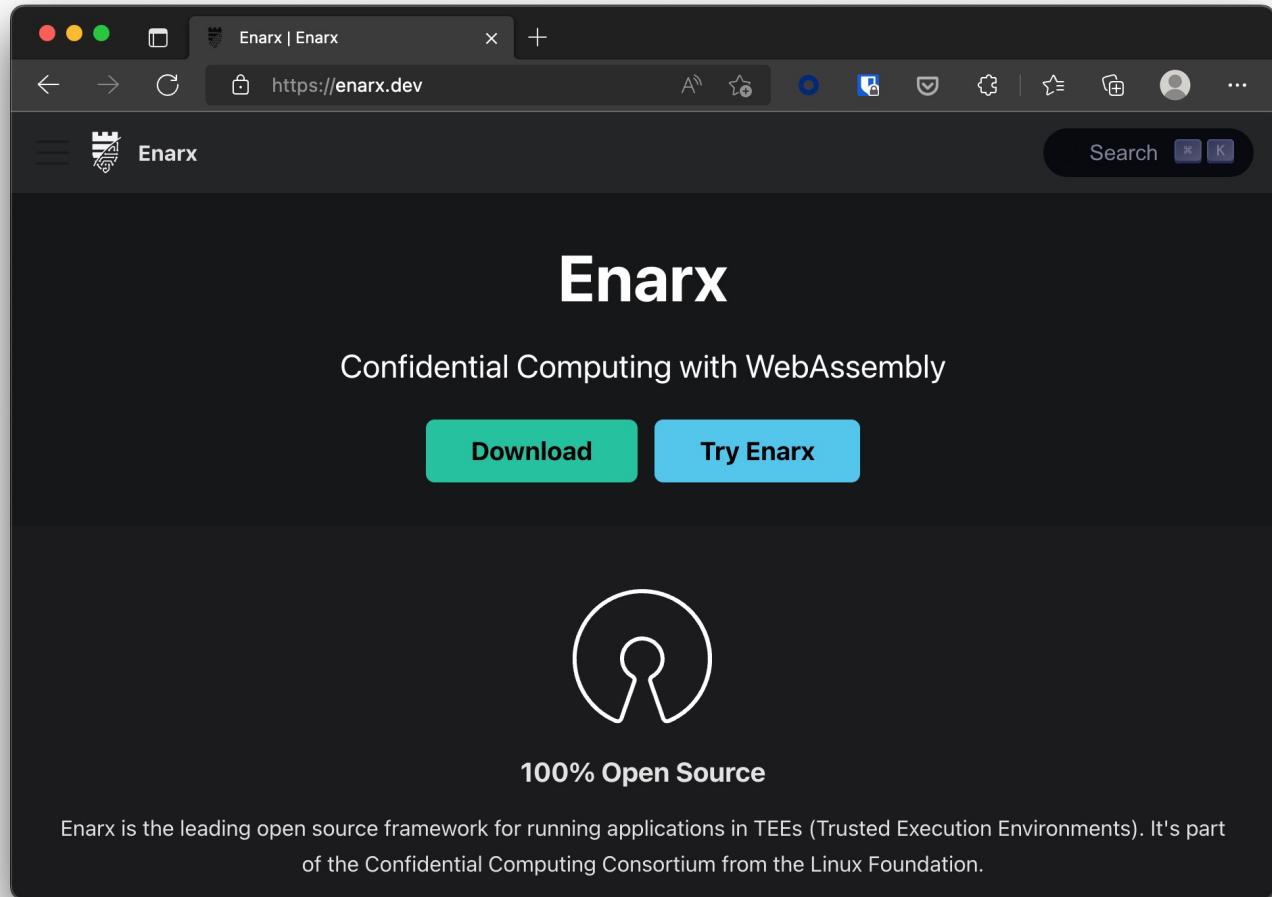
0101  
0101

# Trusted Computing - XKCD 2166



0101  
0101

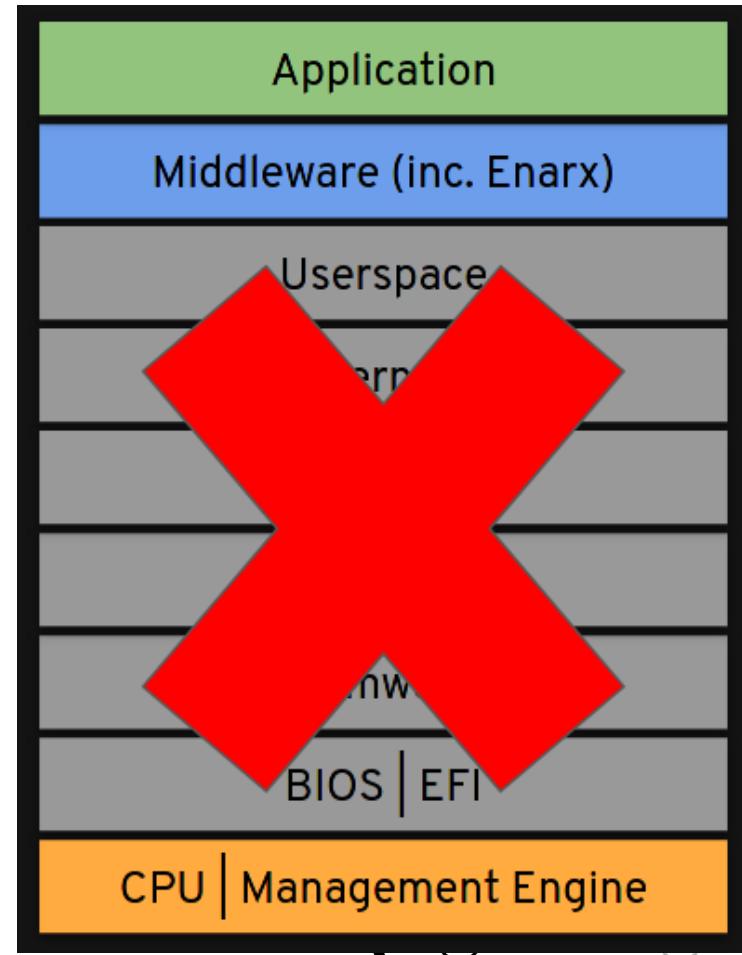
# Enarx





# Enarx Threat Model

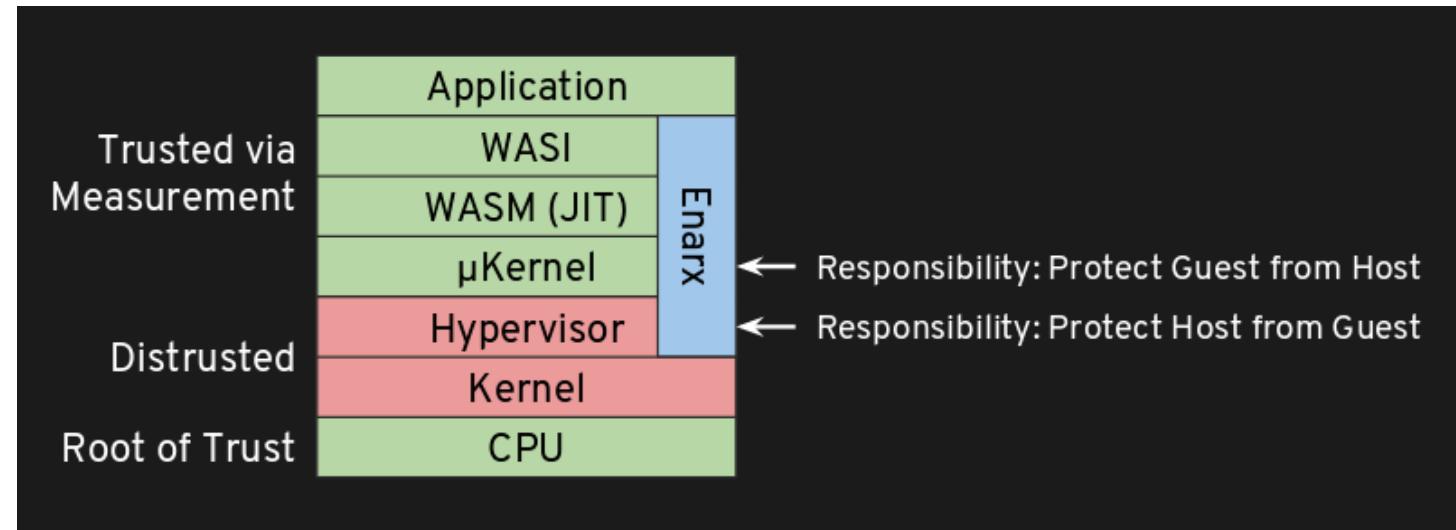
- Don't trust the host
- Don't trust the host owner
- Don't trust the host operator
- Hardware cryptographically verified
- Software audited and cryptographically verified





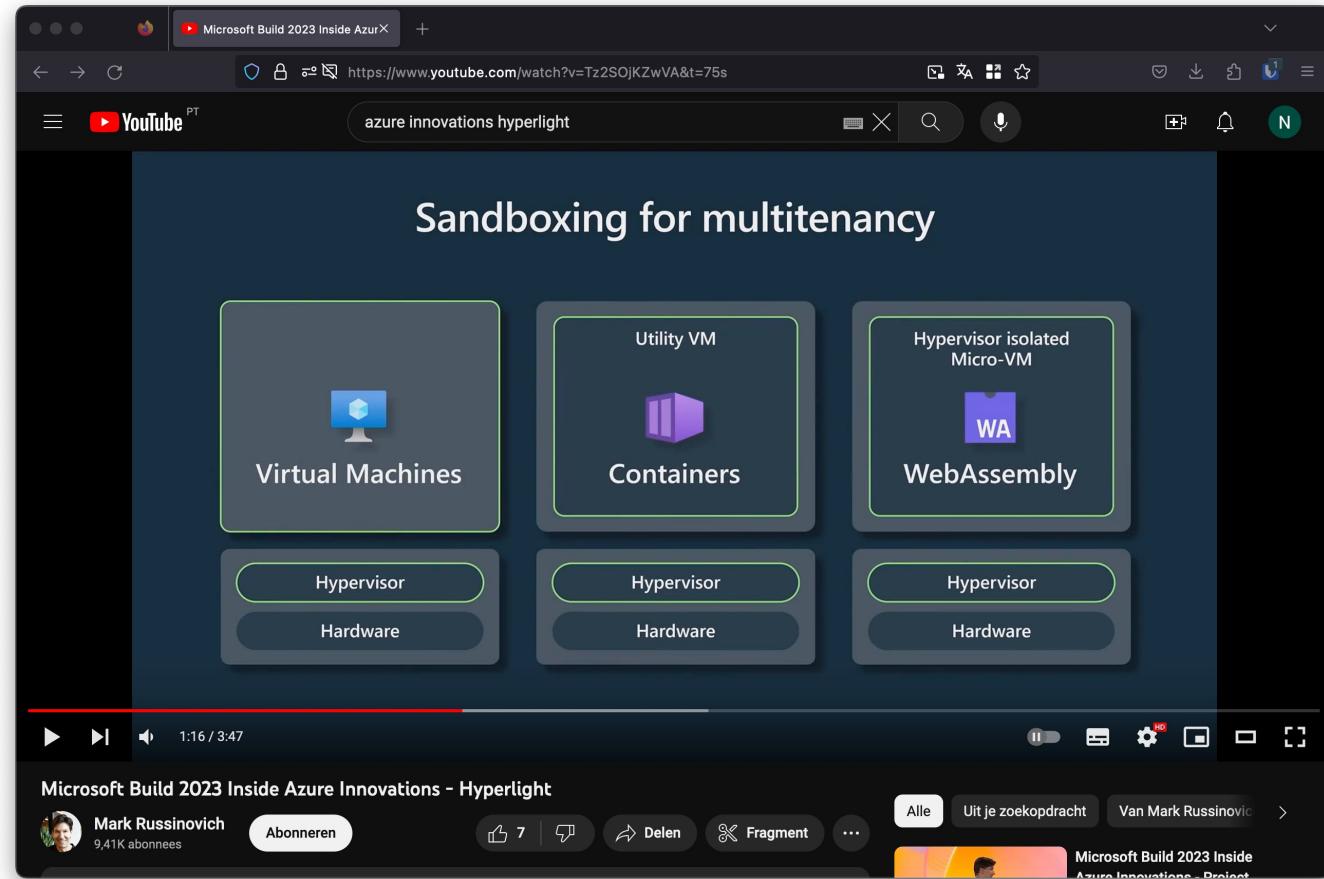
# Enarx

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

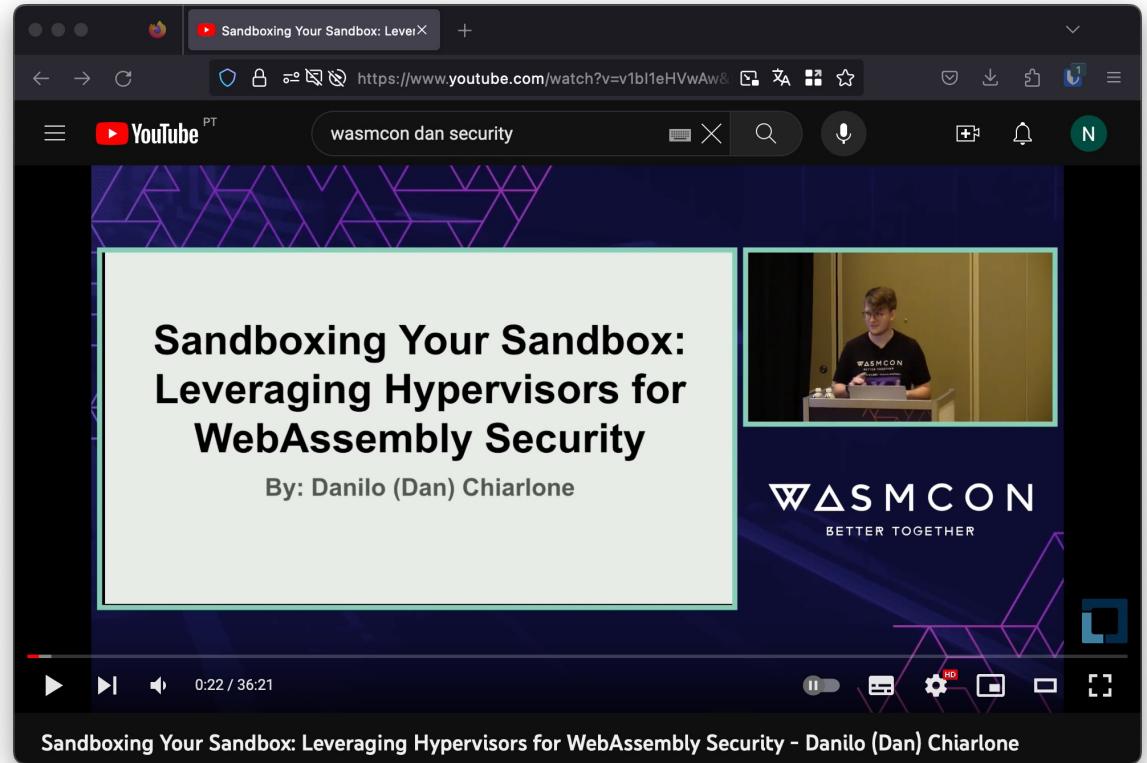
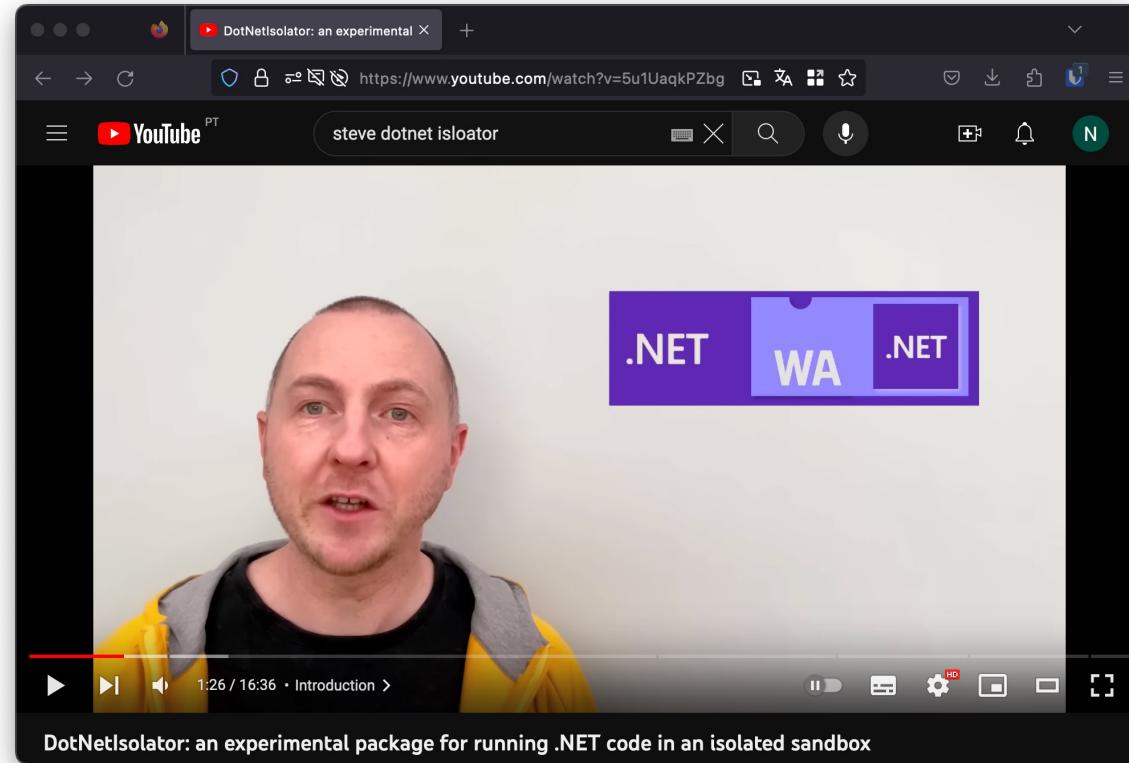


0101  
0101

# Project Hyperlight



# DotNetIsolator & Project Hyperlight



0101  
0101

# WASI Preview 2

The screenshot shows a web browser window with the title bar "WASI Preview 2 Launched - sunfishcode". The URL in the address bar is "https://blog.sunfishcode.online/wasi-preview2/". The page content is as follows:

**sunfishcode's blog**  
A blog by sunfishcode

---

## 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!

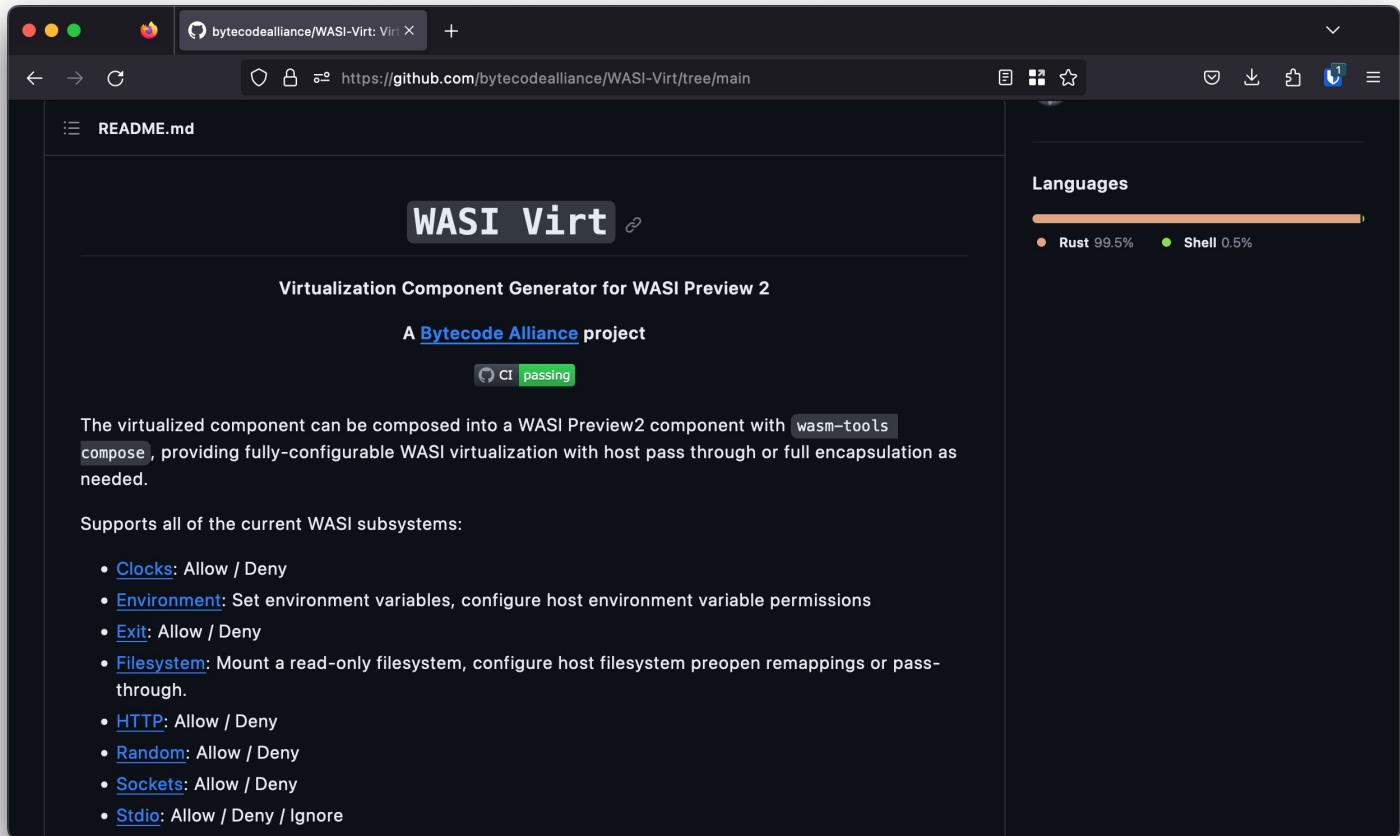
This is a major milestone! We made it! At the same time, the journey is only just beginning. But let's talk this moment to step back and look at what this means.

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

0101  
0101

# WASI Virt



# WasmComponent.SDK

0101  
0101

The screenshot shows a GitHub README page for the `WasmComponent.Sdk` repository. The page has a dark theme. At the top, there's a navigation bar with icons for file, search, and repository details. Below that is a header bar with back, forward, refresh, and search buttons, followed by the URL <https://github.com/SteveSandersonMS/wasm-component-sdk/>. The main content area starts with a `README` section, indicated by a tab at the top left. The title `WasmComponent.Sdk` is bolded. A descriptive paragraph follows, stating: "An experimental package to simplify building WASI preview 2 components using .NET, including early support for WIT files." Another paragraph explains the purpose: "The build output is fully AOT compiled and is known to work in recent versions of wasmtime and WAMR." A `Purpose` section is present, with the text: "This is to simplify experimentation and prototyping." A detailed explanation of the challenge without the package follows: "Without this package, if you wanted to build a WASI preview 2 component with .NET, including using WIT imports/exports, there are about 5 different tools you'd need to discover, download, configure, and manually chain together. Just figuring out which versions of each are compatible with the others is a big challenge. Working out how to get started would be very painful." A final note states: "With this package, you can add one NuGet reference and then get on with your experiments." To the right of the main content, there's a sidebar with a user profile for `NielsPilgaard`, Niels Pilgaard Grøndahl, showing a green progress bar for `Languages` with `C# 100.0%`.

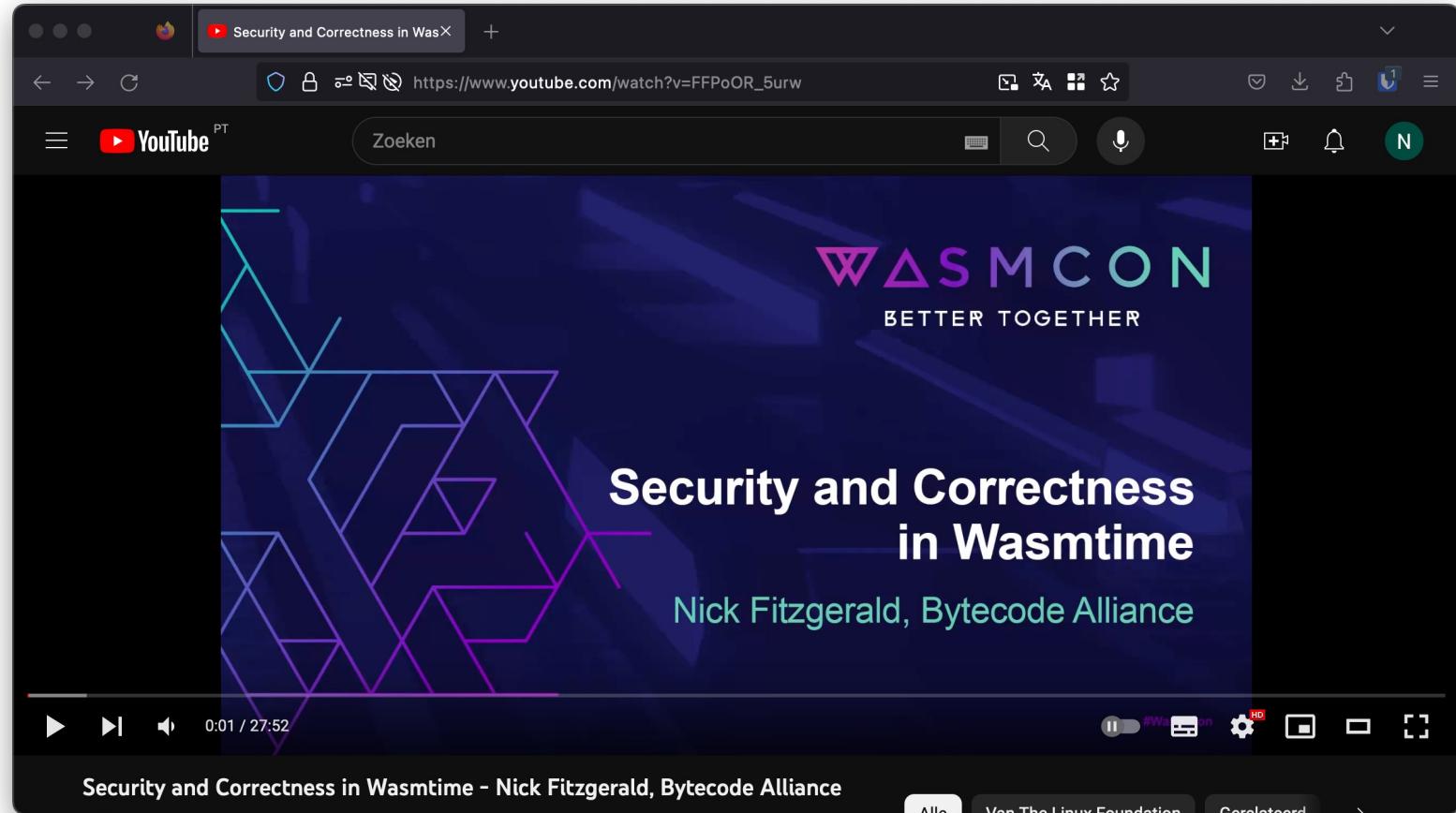


# Runtimes and Security

- 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

0101  
0101

# Runtimes and Security





# 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; now tooling can be implemented!



# Questions?

- <https://github.com/nielstanis/fermyonstream2024>
- ntanis at Veracode.com
- @nielstanis@infosec.exchange
- <https://blog.fennec.dev>
- Bedankt! Thank you!