

# Using WebAssembly to run, extend and secure your app

Niels Tanis  
Sr. Principal Security Researcher



# 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



@niels.fennec.dev @nielstanis@infosec.exchange

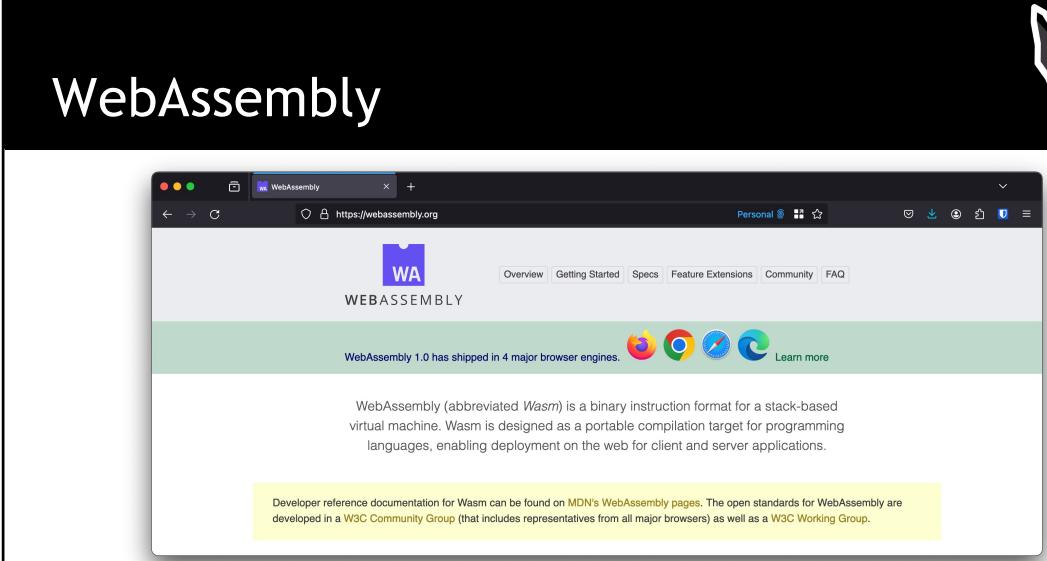


# Agenda

- Introduction
- WebAssembly 101
- Running on WebAssembly
- Extending with WebAssembly
- Securing with WebAssembly
- Conclusion
- Q&A



@niels.fennec.dev @nielstanis@infosec.exchange



The screenshot shows the official WebAssembly website at <https://webassembly.org>. The page features a large "WebAssembly" title on the left and a stylized fox logo on the right. Below the title is a browser window displaying the homepage. The browser window has a tab labeled "WebAssembly". The main content area includes a "WA" logo, navigation links for Overview, Getting Started, Specs, Feature Extensions, Community, and FAQ, and a section stating "WebAssembly 1.0 has shipped in 4 major browser engines." with icons for Firefox, Chrome, Safari, and Edge. A descriptive paragraph explains what WebAssembly is, and a yellow box at the bottom provides developer documentation details.

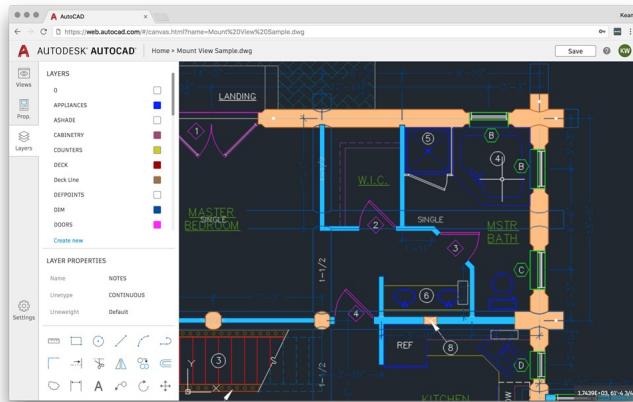
**SecAppDev**

**@niels.fennec.dev** **@nielstanis@infosec.exchange**

<https://webassembly.org/>



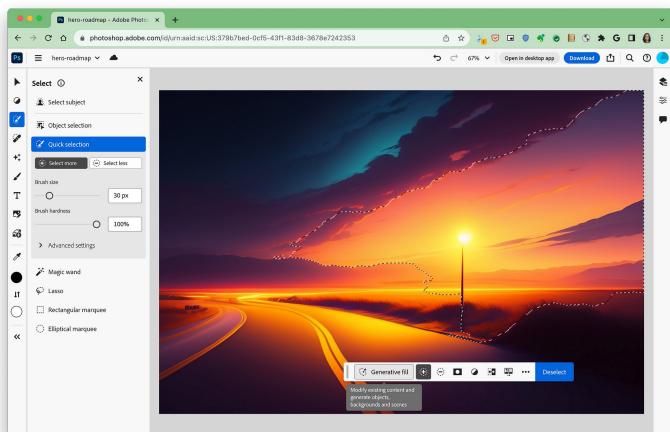
# WebAssembly - AutoCAD



🦋 @niels.fennec.dev ⚡ @nielstanis@infosec.exchange



# WebAssembly - Photoshop



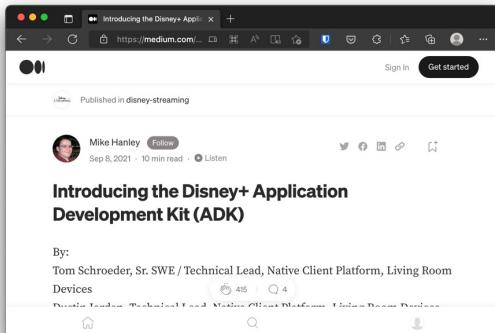
@niels.fennec.dev



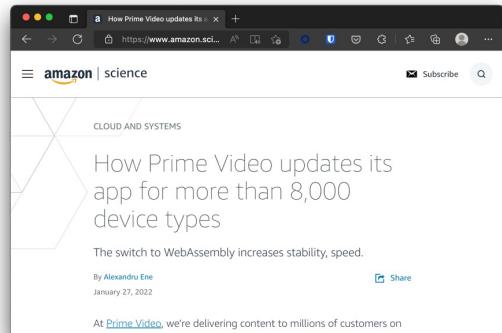
@nielstanis@infosec.exchange



# WebAssembly - SDK's



A screenshot of a Medium article titled "Introducing the Disney+ Application Development Kit (ADK)". The article is published by Mike Hanley and Tom Schroeder. It has 415 views and 4 comments. The content discusses the ADK, which allows developers to build native mobile and web applications using standard tools and frameworks. The article includes code snippets and screenshots of the resulting apps.



A screenshot of a blog post titled "How Prime Video updates its app for more than 8,000 device types". The post is from the "CLOUD AND SYSTEMS" section of the amazon | science website. It discusses how Prime Video switched to WebAssembly to support over 8,000 device types. The post includes a quote from Alexandru Enă and a link to Prime Video's GitHub repository.



🔗 @niels.fennec.dev 📩 @nielstanis@infosec.exchange

<https://medium.com/disney-streaming/introducing-the-disney-application-development-kit-adk-ad85ca139073>

<https://www.amazon.science/blog/how-prime-video-updates-its-app-for-more-than-8-000-device-types>



# Prime Video UI - Rust & WASM

The screenshot shows a presentation slide from InfoQ. The title is "REBUILDING THE PRIME VIDEO UI WITH RUST AND WEBASSEMBLY". Below the title, it says "ALEXANDRUENE PRINCIPAL ENGINEER @ PRIME VIDEO". There is a small thumbnail of a video player interface. At the bottom right, there is a QCon San Francisco logo.



🔗 @niels.fennec.dev 📩 @nielstanis@infosec.exchange

<https://www.infoq.com/presentations/prime-video-rust/>



## WebAssembly - Do you use it?

- Does your organization use WASM?
- Which main tech stacks used?
- WASM for what kind of apps?
  - Mobile
  - Cloud Native
  - Other...



## WEBASSEMBLY



@niels.fennec.dev @nielstanis@infosec.exchange

<https://hacks.mozilla.org/2017/02/creating-and-working-with-webassembly-modules/>  
<https://webassembly.org/>



# 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



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



@niels.fennec.dev



@nielstanis@infosec.exchange

<https://hacks.mozilla.org/2017/02/creating-and-working-with-webassembly-modules/>  
<https://webassembly.org/>



## WebAssembly Type System

- WebAssembly's type only supports:
  - i32 (32-bit integer)
  - i64 (64-bit integer)
  - f32 (32-bit float)
  - f64 (64-bit float)
- No strings, no objects, no complex data types.
- Basic operations on numerical values.



@niels.fennec.dev @nielstanis@infosec.exchange



## WebAssembly Stack Based VM

- In a stack-based VM, operations primarily manipulate a last-in-first-out (LIFO) stack of values, rather than working with named registers as in register-based architectures.



@niels.fennec.dev @nielstanis@infosec.exchange



## WebAssembly Stack Based VM

- WebAssembly maintains an operand stack during execution:
  - Values are pushed onto the stack by instructions
  - Operations pop their operands from the stack
  - Results are pushed back onto the stack
  - The stack is separate from linear memory and is not directly accessible



@niels.fennec.dev @nielstanis@infosec.exchange



## Code some WebAssembly

Let's create some WAT!

[http://github.com/niestanis/secappdev25wasm](https://github.com/niestanis/secappdev25wasm)



# WEBASSEMBLY

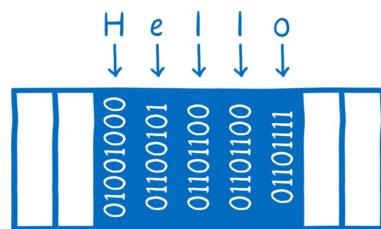
@niels.fennec.dev @niestanis@infosec.exchange





## WebAssembly Memory

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



@niels.fennec.dev @nielstanis@infosec.exchange

Code some WebAssembly



Let's work with some Memory!



WEBASSEMBLY

@niels.fennec.dev @nielstanis@infosec.exchange

# WebAssembly Control-Flow Integrity



- Control-Flow Integrity (CFI) ensures that program execution follows only valid paths as defined by the program's source code.
- In traditional native code, attackers can exploit memory vulnerabilities to hijack execution flow:
  - Redirecting it to malicious code
  - Chaining together existing code fragments in unintended ways (like Return-Oriented Programming attacks).



@niels.fennec.dev @nielstanis@infosec.exchange



## CFI Example

```
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!");
```



 @niels.fennec.dev  @nielstanis@infosec.exchange



## CFI Example

```
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!");
```



🦋 @niels.fennec.dev 🐧 @nielstanis@infosec.exchange



## CFI: Structured Control-Flow Only

- No arbitrary jump instructions (no goto, jmp or equivalent)
- All branches have explicit, validated targets within the same function
- No ability to jump to arbitrary memory addresses
- All control flow is validated before execution



@niels.fennec.dev



@nielstanis@infosec.exchange



## CFI: Protected Function Calls

```
(module
  (func $safe_function (param i32) (result i32)
    local.get 0
    i32.const 1
    i32.add
  )

  (func $caller (param i32) (result i32)
    local.get 0
    call $safe_function ; Direct call - target validated at compile time
  )
)
```



 @niels.fennec.dev  @nielstanis@infosec.exchange



## CFI: Indirect Calls

- Function table entries are validated at module instantiation
- Runtime checks ensure the target index is within table bounds
- Runtime type checking ensures the called function matches the expected signature
- The table is protected from direct memory manipulation



@niels.fennec.dev



@nielstanis@infosec.exchange

Code some WebAssembly



Indirect calling!



WEBASSEMBLY



@niels.fennec.dev @nielstanis@infosec.exchange



## CFI: Indirect Calls

```
(module
  (table 2 funcref)
  (func $f1 (result i32)
    i32.const 42)
  (func $f2 (result i32)
    i32.const 13)
  (elem (i32.const 0) $f1 $f2)
  (type $return_i32 (func (result i32)))
  (func (export "callByIndex") (param $i i32) (result i32)
    local.get $i
    call_indirect (type $return_i32))
)
```



@niels.fennec.dev



@nielstanis@infosec.exchange



## CFI: Separation of Code and Data

- Code sections cannot be modified at runtime
- Linear memory (accessible data) cannot contain executable code
- No instruction exists to convert data to code No self-modifying code capabilities



 @niels.fennec.dev  @nielstanis@infosec.exchange

RLBox



# RLBox



@niels.fennec.dev @nielstanis@infosec.exchange

<https://rlbox.dev/>

<https://hacks.mozilla.org/2020/02/securing-firefox-with-webassembly/>



## RLBox in Firefox

- RLBox uses **WebAssembly as an intermediate compilation target** for creating sandboxed versions of third-party C/C++ libraries.
- Rather than running WebAssembly code directly, RLBox employs a unique "**WebAssembly and Back Again**" approach.



[@niels.fennec.dev](https://twitter.com/nielsfennec) [@nielstanis@infosec.exchange](mailto:@nielstanis@infosec.exchange)



## RLBox in Firefox

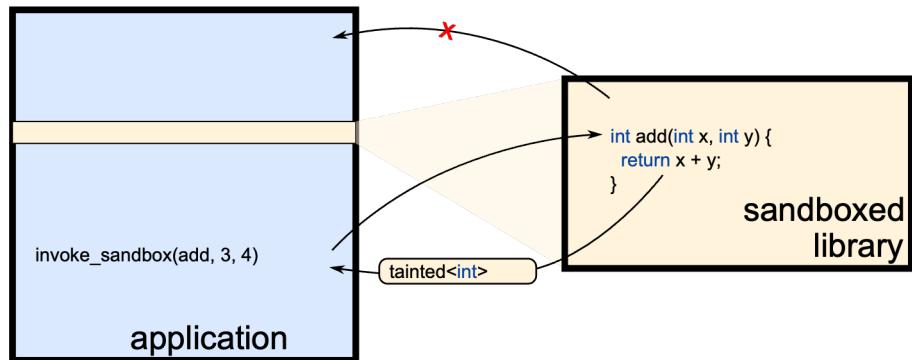
- RLBox considers all values that originate in the sandbox as untrusted and "taints" them.
- Tainted values are essentially opaque values that cannot be used directly by the application code.
- RLBox's type system marks all data coming out of the sandbox as "tainted" and ensures, through compiler errors, that developers sanitize potentially unsafe data before using it.



@niels.fennec.dev @nielstanis@infosec.exchange



## RLBox in Firefox



🦋 @niels.fennec.dev 🐸 @nielstanis@infosec.exchange



## RLBox in Firefox

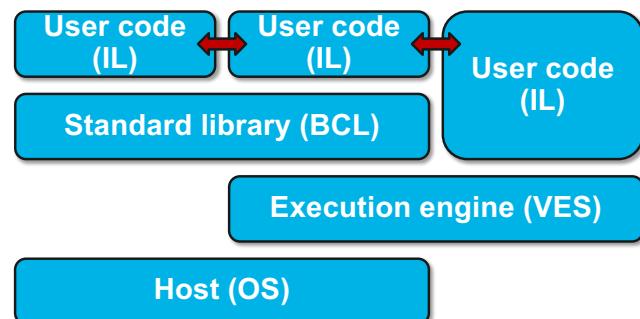
- RLBox is currently deployed in Firefox to isolate five different modules:
  - Graphite - Font rendering engine - Firefox 95
  - Hunspell - Spell checker - Firefox 95
  - Ogg - Multimedia container format - Firefox 95
  - Expat - XML parser - Firefox 96
  - Woff2 - Web font compression format - Firefox 96



@niels.fennec.dev @nielstanis@infosec.exchange



# Running .NET on WebAssembly



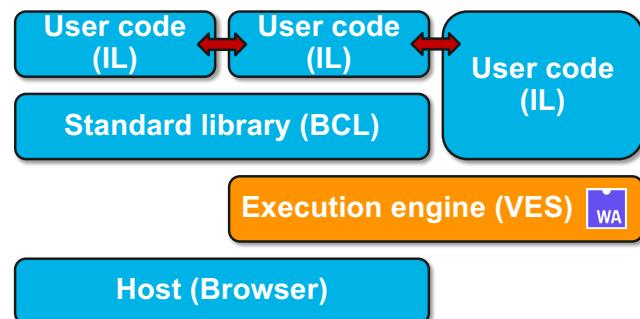
@niels.fennec.dev @nielstanis@infosec.exchange

Diagram:

<https://github.com/itowlson/wasmday22/blob/main/slides/Wasm%20Interfaces%20and%20.NET.pptx>



# Running .NET on WebAssembly



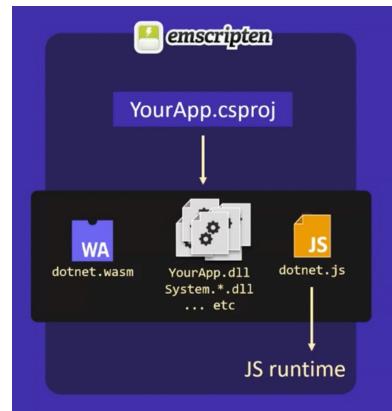
🐦 @niels.fennec.dev 🐾 @nielstanis@infosec.exchange

Diagram:

<https://github.com/itowlson/wasmday22/blob/main/slides/Wasm%20Interfaces%20and%20.NET.pptx>



# Blazor WebAssembly



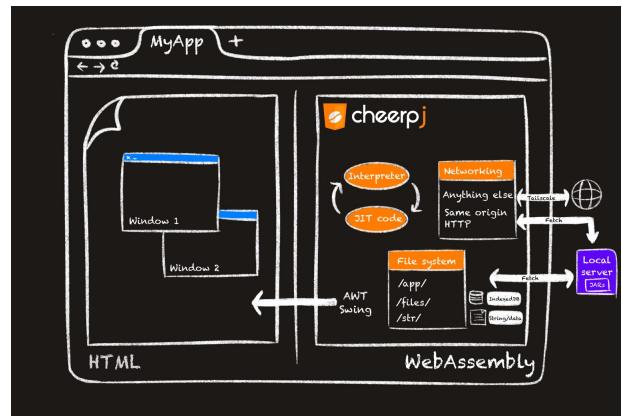
SecAppDev

@niels.fennec.dev @nielstanis@infosec.exchange



## JVM on WASM with CheerpJ

The magic behind CheerpJ is Cheerp, which was used for compiling full Java SE 8 and Java SE 11 runtimes based on OpenJDK.



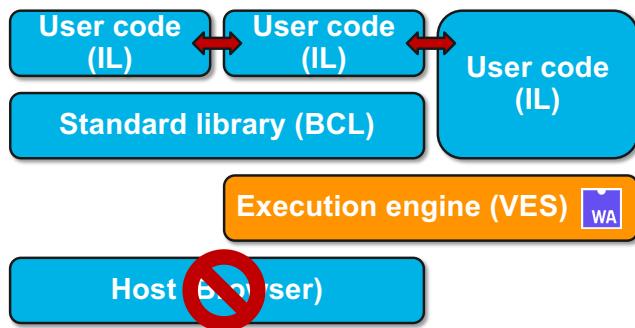
[@niels.fennec.dev](https://cheerpj.com/docs/explanation/architecture) [@nielstanis@infosec.exchange](mailto:@nielstanis@infosec.exchange)

<https://cheerpj.com/docs/explanation/architecture>

<https://cheerpjdemos.leaningtech.com/PHETDemos.html#demo>



# Running .NET on WebAssembly



@niels.fennec.dev @nielstanis@infosec.exchange

Diagram:

<https://github.com/itowlson/wasmday22/blob/main/slides/Wasm%20Interfaces%20and%20.NET.pptx>

## 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



[@niels.fennec.dev](https://twitter.com/nielsfennec) [@nielstanis@infosec.exchange](mailto:@nielstanis@infosec.exchange)

## WebAssembly System Interface WASI



- Strong sandbox with Capability Based Security
- Right now, supports e.g. FileSystem actions, Sockets, CLI and HTTP at version 0.2
- Future support for promise/async and streams
- Anyone recall .NET Standard? 😊



@niels.fennec.dev @nielstanis@infosec.exchange

# Docker vs WASM & WASI

A screenshot of a Twitter web client window. The tweet is from user @solomonstrel (Solomon Hykes) and 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!

The tweet has a timestamp of 9:39 p.m. - 27 mrt. 2019 and a note that it was posted via Twitter Web Client.



@niels.fennec.dev @nielstanis@infosec.exchange

# Docker vs WASM & WASI



A screenshot of a Twitter post from Solomon Hykes (@solomonstre). The tweet reads: "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 :)

The post was made on March 27, 2019, at 4:50 a.m. It has 56 retweets, 5 quoted tweets, and 165 likes. The Twitter interface shows various icons for navigation and interaction.



 @niels.fennec.dev  @nielstanis@infosec.exchange



# Docker & WASM

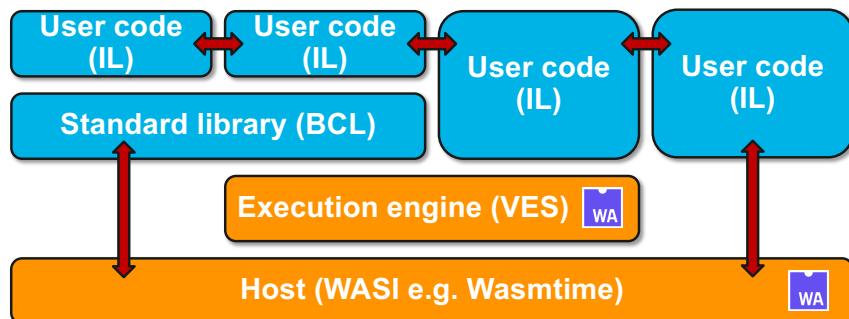
The screenshot shows two side-by-side browser windows. The left window displays the homepage for 'Introducing the Docker+Wasm Technical Preview' by Michael Irwin on October 24, 2022. It features a brief introduction and a note about the availability of the Technical Preview. The right window shows a detailed architectural diagram of the Docker+Wasm stack. At the top is the 'Docker Engine'. Below it is a container labeled 'containerd'. Inside 'containerd', there are three sub-components: 'containerd-shim', 'containerd-shim', and 'containerd-wasm shim'. Each 'shim' component contains a 'runc' process and a 'Container process'. The 'containerd-wasm shim' also includes 'wasmedge' and a 'Wasm Module'.



🐦 @niels.fennec.dev 🐾 @nielstanis@infosec.exchange



# WebAssembly System Interface WASI



🐦 @niels.fennec.dev 🐾 @nielstanis@infosec.exchange



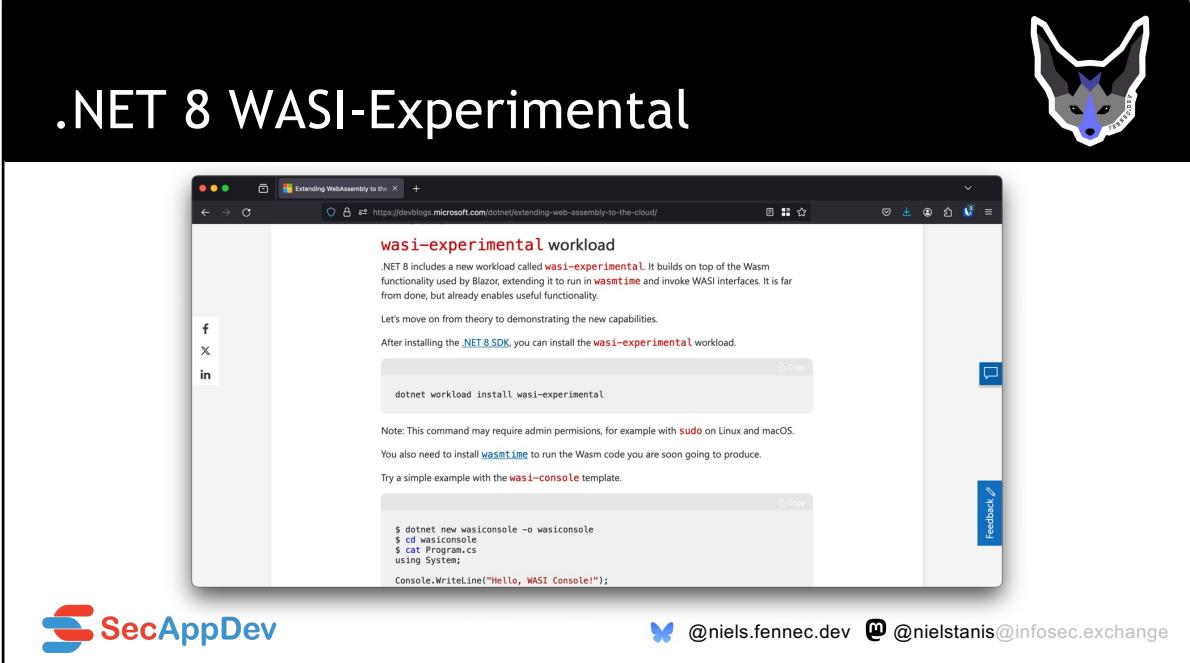
# Experimental WASI SDK for .NET



SecAppDev

@niels.fennec.dev @nielstanis@infosec.exchange

<https://github.com/SteveSandersonMS/dotnet-wasi-sdk>



**.NET 8 WASI-Experimental**

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!");
```



 [@niels.fennec.dev](#)  [@nielstanis@infosec.exchange](#)

<https://github.com/dotnet/runtime/issues/65895>

<https://github.com/SteveSandersonMS/dotnet-wasi-sdk>

<https://devblogs.microsoft.com/dotnet/extending-web-assembly-to-the-cloud/>



## 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!

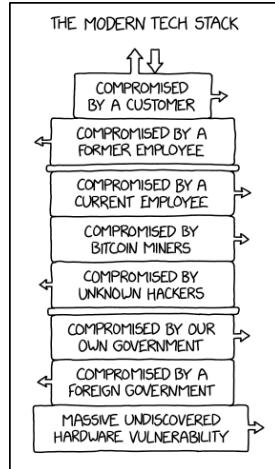


@niels.fennec.dev @nielstanis@infosec.exchange



# Trusted Computing - XKCD 2166

 SecAppDev



ls.fennec.dev  @nielstanis@infosec.exchange

<https://xkcd.com/2166/>

The screenshot shows the Enarx website (<https://enarx.dev>) running in a web browser. The main content area displays the Enarx logo, the title "Enarx", and the subtitle "Confidential Computing with WebAssembly". It features two prominent buttons: "Download" and "Try Enarx". Below these buttons are three sections: "100% Open Source", "Easy Deployment", and "Cloud Native, Hardware Neutral". Each section includes an icon and a brief description. The "100% Open Source" section notes that Enarx is the leading open source framework for confidential computing in TEEs. The "Easy Deployment" section highlights support for WasmEdge and the ability to deploy applications without rewrites. The "Cloud Native, Hardware Neutral" section emphasizes CPU-architecture independence and transparent application code across multiple targets. A sidebar on the left contains the "Enarx" logo and links to "Docs", "Resources", "Community", and "Initiatives". The top right corner of the slide features a stylized fox head logo.

Enarx

Enarx

100% Open Source

Easy Deployment

Cloud Native, Hardware Neutral

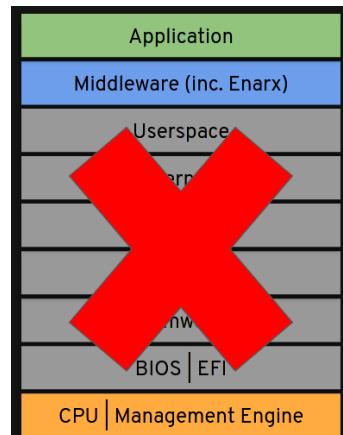
[@niels.fennec.dev](https://enarx.dev) [@nielstanis@infosec.exchange](mailto:@nielstanis@infosec.exchange)

<https://enarx.dev/>



## 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

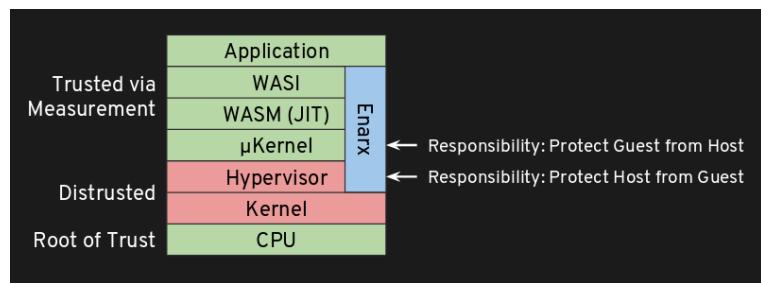


[@niels.fennec.dev](https://twitter.com/nielsfennec) [@niestanis@infosec.exchange](https://twitter.com/niestanis)



# 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



🔗 @niels.fennec.dev 📩 @nielstanis@infosec.exchange



# Enarx

- **Each execution:**

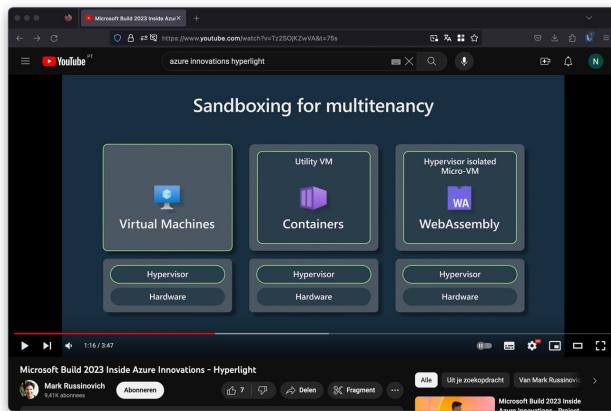
- Attestation: Enarx checks that the host to which you're planning to deploy is a genuine TEE instance.
- Packaging: Once the attestation is complete and the TEE instance verified, the Enarx management component encrypts the application, along with any required data.
- Provisioning: Enarx then sends the application and data along to the host for execution in the Enarx Keep.

- **TEE provides:** Data Confidentiality, Data Integrity, Code Integrity



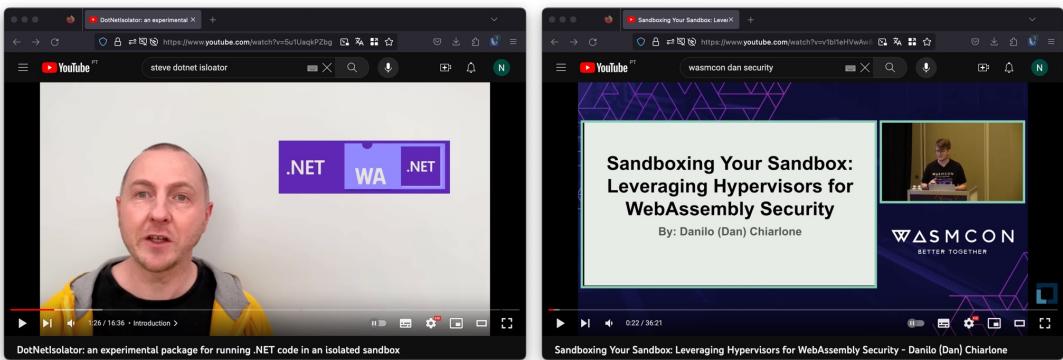
@niels.fennec.dev @nielstanis@infosec.exchange

# Project Hyperlight



🦋 @niels.fennec.dev 🐧 @nielstanis@infosec.exchange

# DotNetIsolator & Project Hyperlight



@niels.fennec.dev @nielstanis@infosec.exchange



# Hyperlight CNCF Sandbox

The screenshot shows a web browser displaying a blog post titled "Hyperlight Wasm: Fast, secure, and OS-free". The post is dated March 26, 2025, and has a 10-minute estimated read time. It is written by Yosh Wuyts, Senior Developer Advocate, and Lucy Memon, Software engineer and researcher, Microsoft. The content discusses the introduction of Hyperlight, its use as an open-source Rust library for executing small embedded functions using hypervisor-based protection, and its integration into the Cloud Native Computing Foundation (CNCF) Sandbox program. The page includes social sharing icons for Facebook, X, and LinkedIn, and a "CONTENT TYPE" section.



🔗 @niels.fennec.dev 📩 @nielstanis@infosec.exchange

<https://opensource.microsoft.com/blog/2025/03/26/hyperlight-wasm-fast-secure-and-os-free/>

# WASM - What's next?



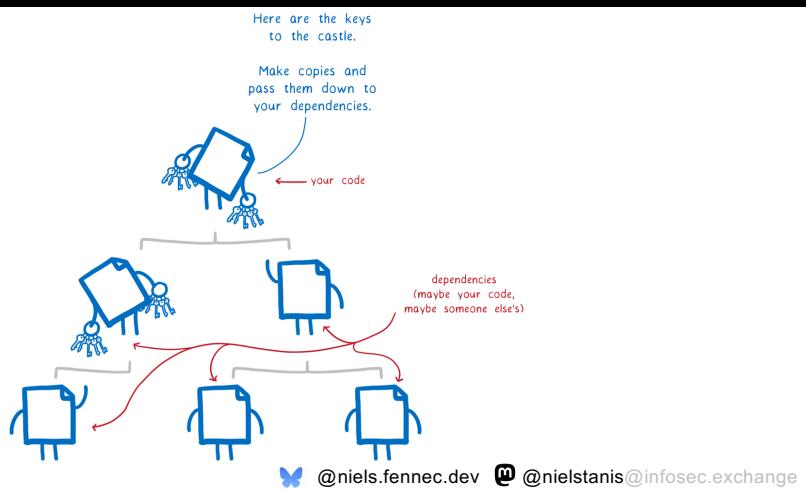
composition of an  
average code base



@niels.fennec.dev @nielstanis@infosec.exchange

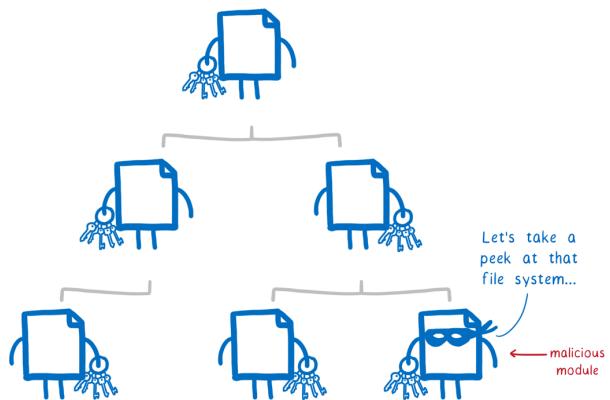


# Dependencies





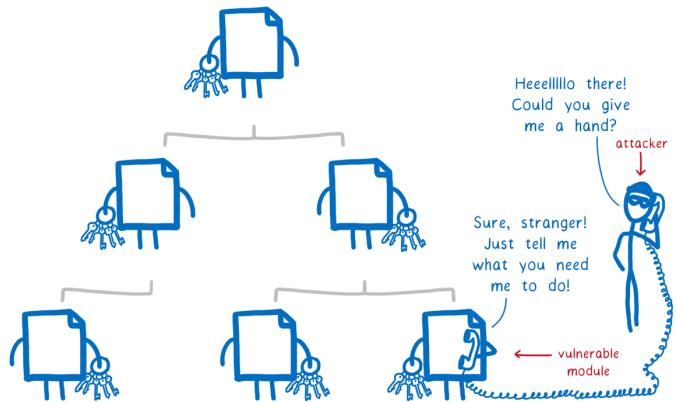
## Malicious module



@niels.fennec.dev @nielstanis@infosec.exchange



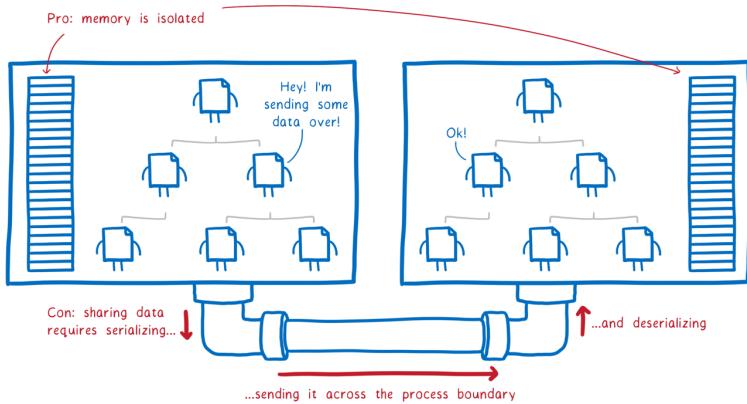
## Vulnerable module



🔗 @niels.fennec.dev 📩 @nielstanis@infosec.exchange

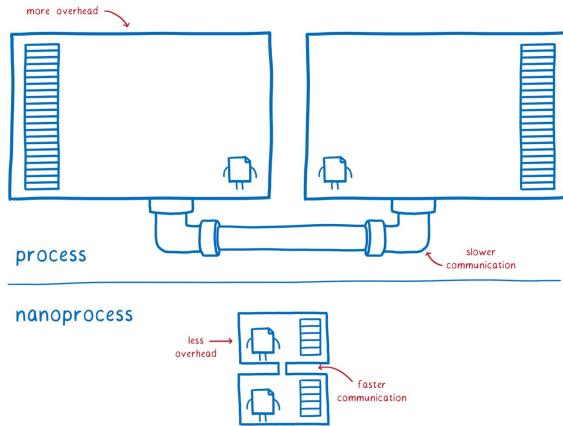


# Process Isolation





# WebAssembly Nano-Process



SecAppDev



@niels.fennec.dev

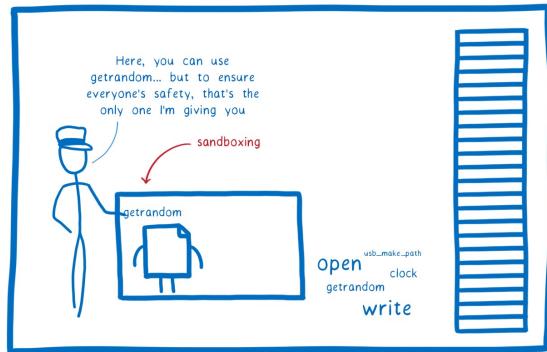


@nielstanis@infosec.exchange



# WebAssembly Nano-Process

## 1. Sandboxing

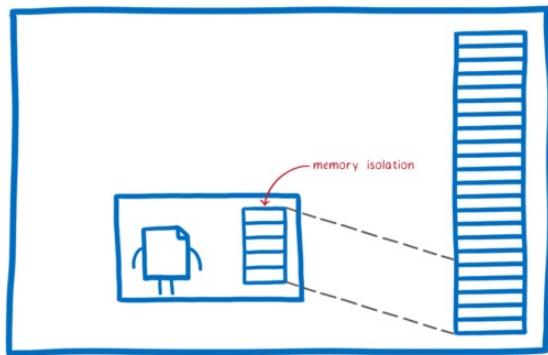


@niels.fennec.dev @nielstanis@infosec.exchange



# WebAssembly Nano-Process

## 2. Memory model

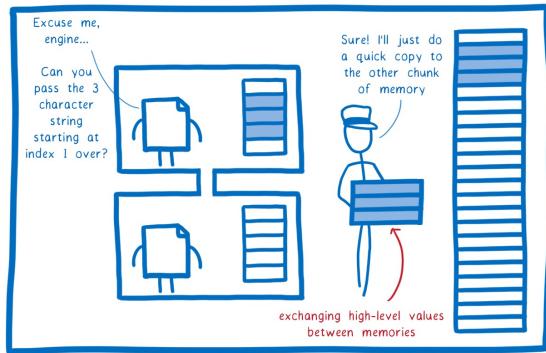


@niels.fennec.dev @nielstanis@infosec.exchange



# WebAssembly Nano-Process

## 3. Interface Types

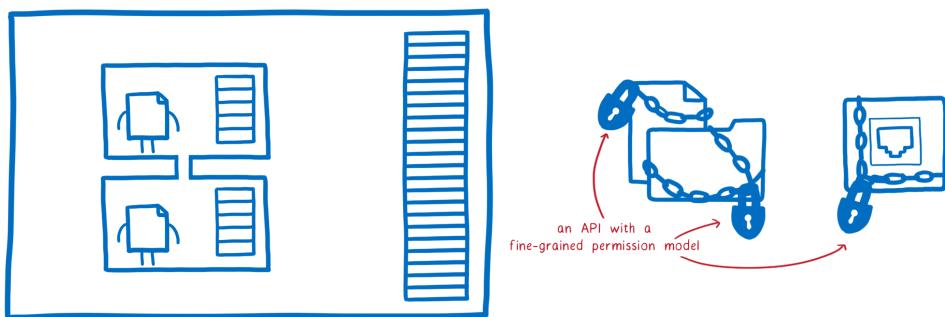


 @niels.fennec.dev  @nielstanis@infosec.exchange



# WebAssembly Nano-Process

## 4. WebAssembly System Interface

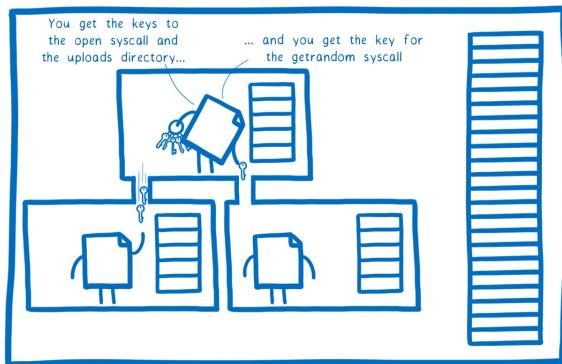


@niels.fennec.dev @nielstanis@infosec.exchange



# WebAssembly Nano-Process

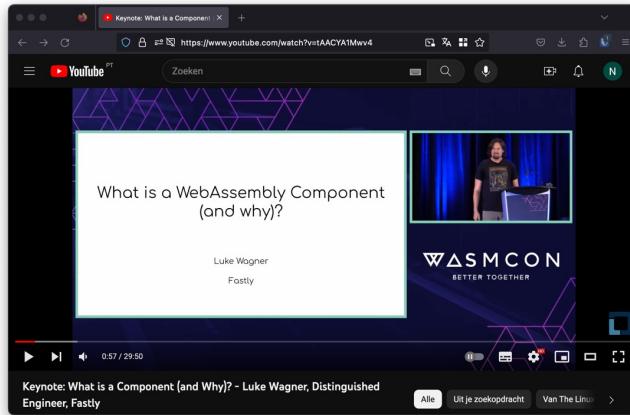
## 5. The missing link



@niels.fennec.dev @nielstanis@infosec.exchange



# WebAssembly Component Model

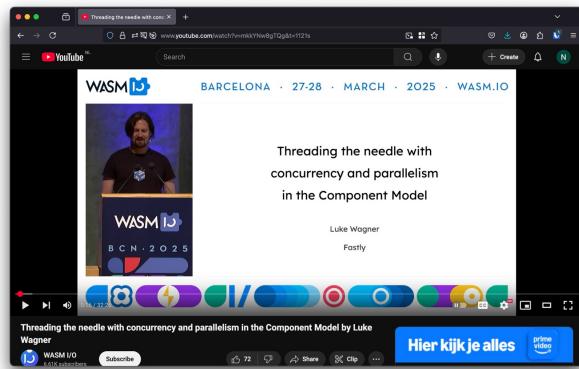


 SecAppDev

 @niels.fennec.dev  @nielstanis@infosec.exchange

<https://www.youtube.com/watch?v=tAACYA1Mwv4>

# WebAssembly Component Model



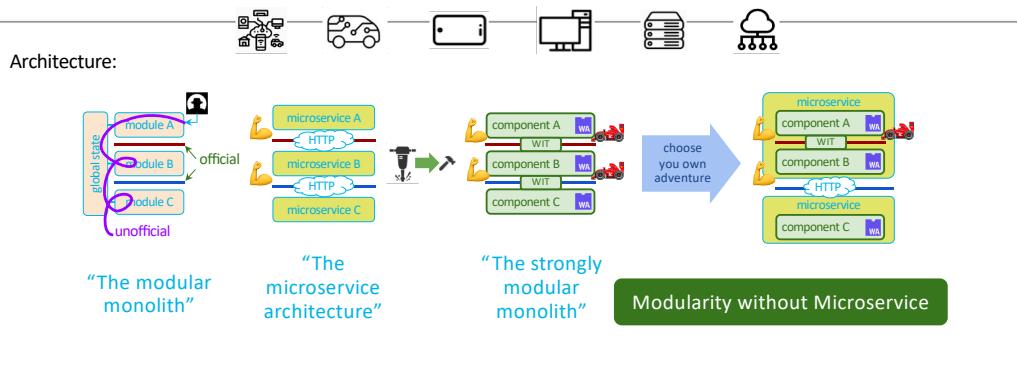
🦋 @niels.fennec.dev 🐀 @nielstanis@infosec.exchange

<https://www.youtube.com/watch?v=mkkYNw8gTQg&t=1121s>

# WASI 0.3



?



🦋 @niels.fennec.dev 🐾 @nielstanis@infosec.exchange

<https://www.youtube.com/watch?v=mkkYNw8gTQg&t=1121s>

**WASI Preview 2 (0.2)**

**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**

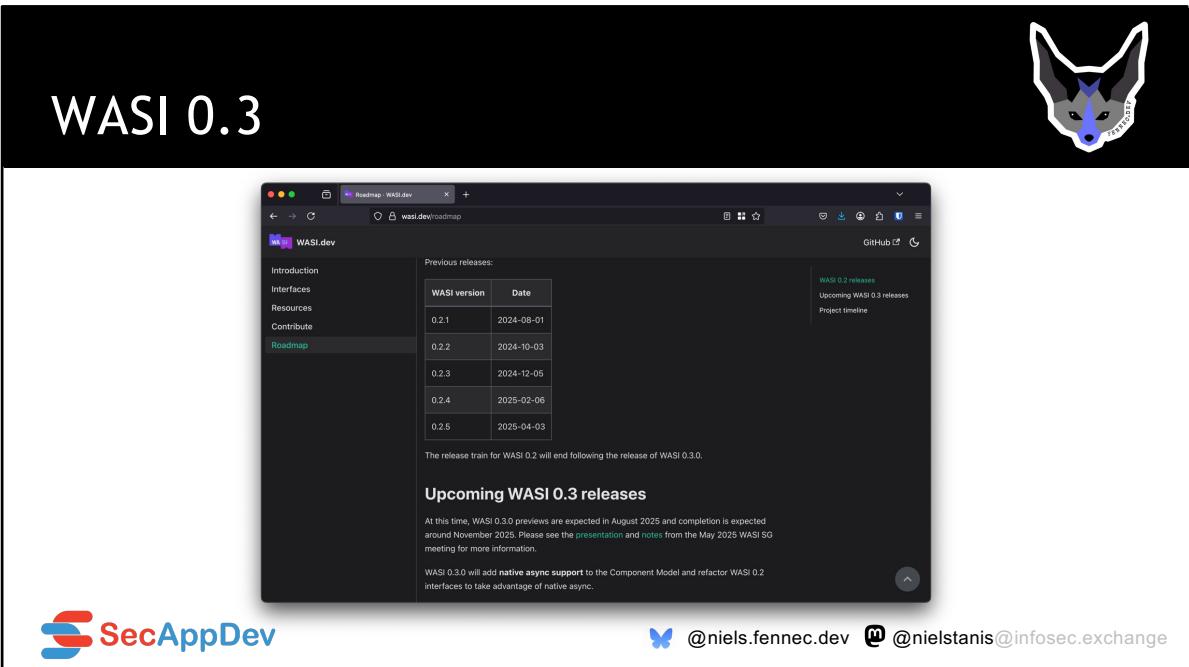
The Subgroup voted to launch Preview 2!

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

  @niels.fennec.dev  @nielstanis@infosec.exchange



The screenshot shows a dark-themed web browser window displaying the WASI.dev roadmap page. The title bar says "Roadmap · WASI.dev". The main content area has a table showing previous releases:

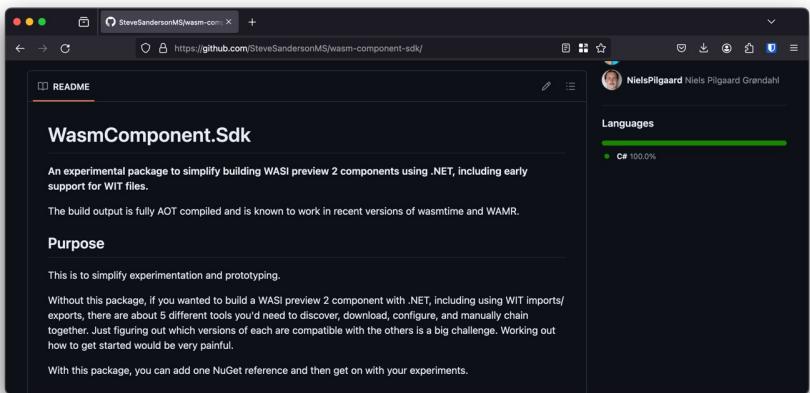
WASI version	Date
0.2.1	2024-08-01
0.2.2	2024-10-03
0.2.3	2024-12-05
0.2.4	2025-02-06
0.2.5	2025-04-03

Below the table, it says: "The release train for WASI 0.2 will end following the release of WASI 0.3.0." A section titled "Upcoming WASI 0.3 releases" contains the following text: "At this time, WASI 0.3.0 previews are expected in August 2025 and completion is expected around November 2025. Please see the [presentation](#) and [notes](#) from the May 2025 WASI SG meeting for more information." It also notes: "WASI 0.3.0 will add native async support to the Component Model and refactor WASI 0.2 interfaces to take advantage of native async."

**SecAppDev**   @niels.fennec.dev  @nielstanis@infosec.exchange

<https://www.youtube.com/watch?v=mkkYNw8gTQg&t=1121s>

# WasmComponent.SDK



The screenshot shows a GitHub repository page for "WasmComponent.SDK". The repository has 1 star and 1 fork. It was created by "SteveSandersonMS" on April 19, 2023. The README file contains sections for "Purpose" and "Usage". The "Purpose" section explains the package's goal of simplifying WASI preview 2 component building for .NET. The "Usage" section provides instructions for adding a NuGet reference. On the right side of the repository page, there is a profile card for "NielsPilgaard" showing a bio, languages (C# 100%), and social links.

  @niels.fennec.dev  @nielstanis@infosec.exchange

<https://github.com/SteveSandersonMS/wasm-component-sdk/>

# componentize-dotnet



The screenshot shows the GitHub repository page for 'componentize-dotnet'. The page includes a README file, a code of conduct, an Apache-2.0 license, and security information. It features a 'Purpose' section explaining the goal of simplifying Wasm components in .NET. A note mentions that without this package, building a WASI .NET component would require manually chaining together multiple different tools. The 'Getting started' section includes a note about the compiler being limited to Windows. The repository has 100% coverage in C#.



@niels.fennec.dev @nielstanis@infosec.exchange

<https://github.com/bytocodealliance/componentize-dotnet/>



# WASI Virt

The screenshot shows a GitHub repository page for 'bytecodealliance/WASI-Virt'. The page title is 'WASI Virt' and it describes the project as a 'Virtualization Component Generator for WASI Preview 2' and a 'Bytecode Alliance project'. It includes a 'Languages' section showing Rust at 99.8% and Shell at 0.1%. Below the main content, there's a list of subsystems supported by WASI Virt, including Clocks, Environment, Exit, Filesystem, HTTP, Random, Sockets, and Studio, each with 'Allow / Deny' options.

**SecAppDev**

 @niels.fennec.dev  @nielstanis@infosec.exchange

<https://www.youtube.com/watch?v=tAACYA1Mwv4>



## Runtimes and Security

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



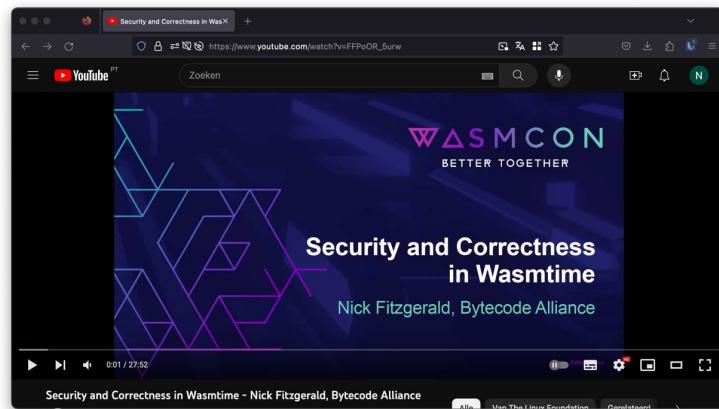
@niels.fennec.dev



@nielstanis@infosec.exchange

<https://bytecodealliance.org/articles/security-and-correctness-in-wasmtime>

# Runtimes and Security



🦋 @niels.fennec.dev 🐸 @nielstanis@infosec.exchange

[https://www.youtube.com/watch?v=FFPoOR\\_5urw](https://www.youtube.com/watch?v=FFPoOR_5urw)

**WebAssembly Lineair Memory**

The screenshot shows a presentation slide with a black header containing the title "WebAssembly Lineair Memory" and a stylized fox logo on the right. Below the header is a white content area showing a browser window displaying a conference paper. The browser window has a dark header with the Usenix logo and navigation links like "ATTEND", "PROGRAM", "PARTICIPATE", "SPONSORS", and "ABOUT". The main content of the browser window is a white page with the title "Everything Old is New Again: Binary Security of WebAssembly" in bold. Below the title is a section for "Authors" listing Daniel Lehmann, Johannes Kinder, and Michael Pradel from the University of Stuttgart. A detailed "Abstract" follows, describing the security challenges of WebAssembly. At the bottom of the browser window, there's a small "co" logo.

**SecAppDev**

**@niels.fennec.dev** **@nielstanis@infosec.exchange**

<https://www.usenix.org/conference/usenixsecurity20/presentation/lehmann>



# WebAssembly Lineair Memory

23 Oct 2024



@niels.fennec.dev @nielstanis@infosec.exchange

<https://arxiv.org/pdf/2410.17925v1.pdf>



## Conclusion

- 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 0.2 big milestone; tooling in progress!
- WASI 0.3 due in August 2025
- Cloud Native ❤️ WebAssembly



🐦 @niels.fennec.dev 🐾 @nielstanis@infosec.exchange



Merci! Bedankt! Thanks!

- <https://github.com/niestanis/secappdev25wasm>
- ntanis at Veracode.com
- @niestanis@infosec.exchange
- <https://blog.fennec.dev>



@niels.fennec.dev @niestanis@infosec.exchange