

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

Niels Tanis





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

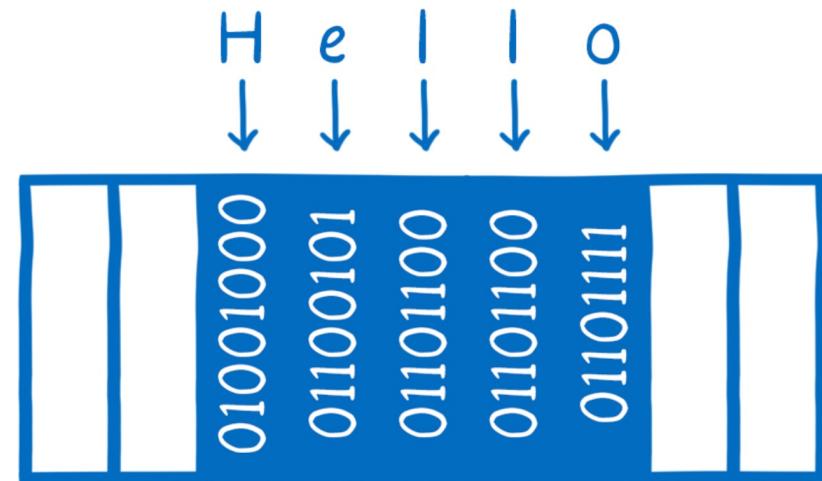
- Binary instruction format for stack-based virtual machine similar to .NET running MSIL
- Designed as a portable compilation target
- The security model of WebAssembly:
 - Protect users from buggy or malicious modules
 - Provide developers with useful primitives and mitigations for developing safe applications



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!");
```



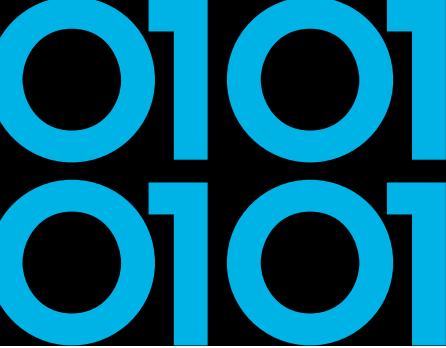
WebAssembly Control-Flow Integrity

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

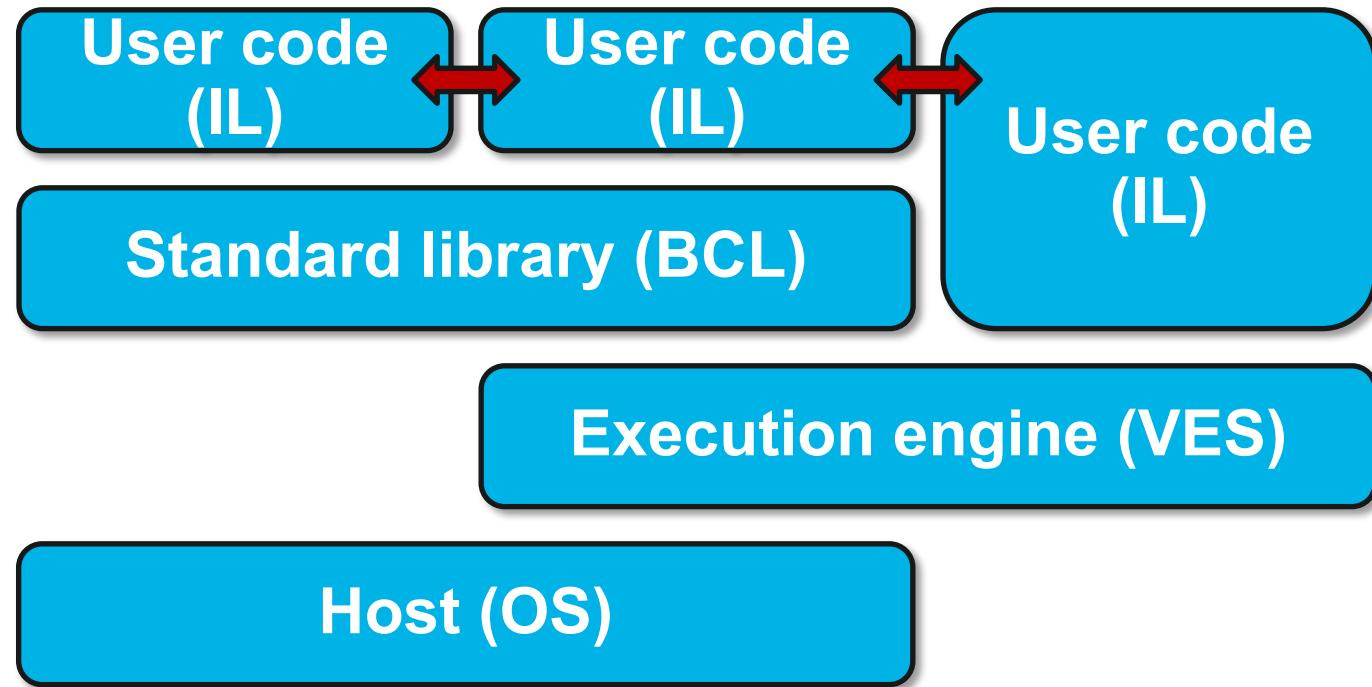
```
Console.WriteLine("Number is larger than 5");
```

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

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

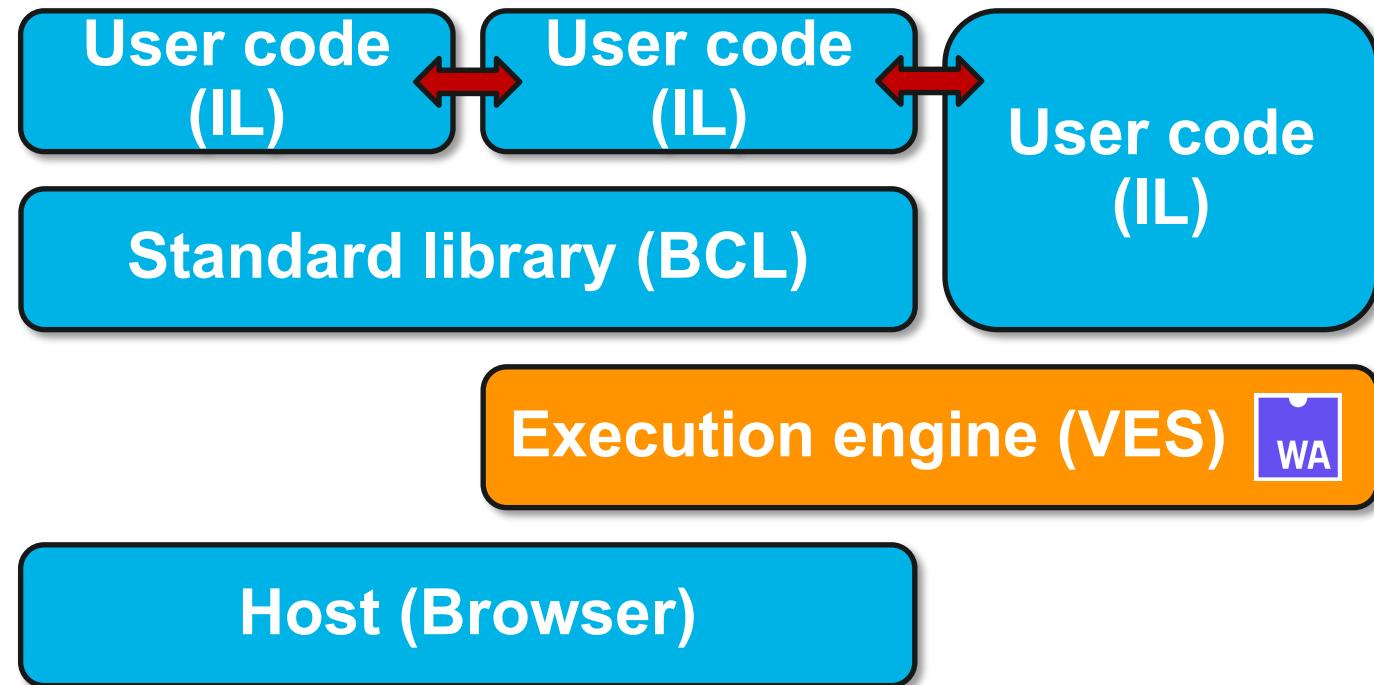


Running .NET on WebAssembly



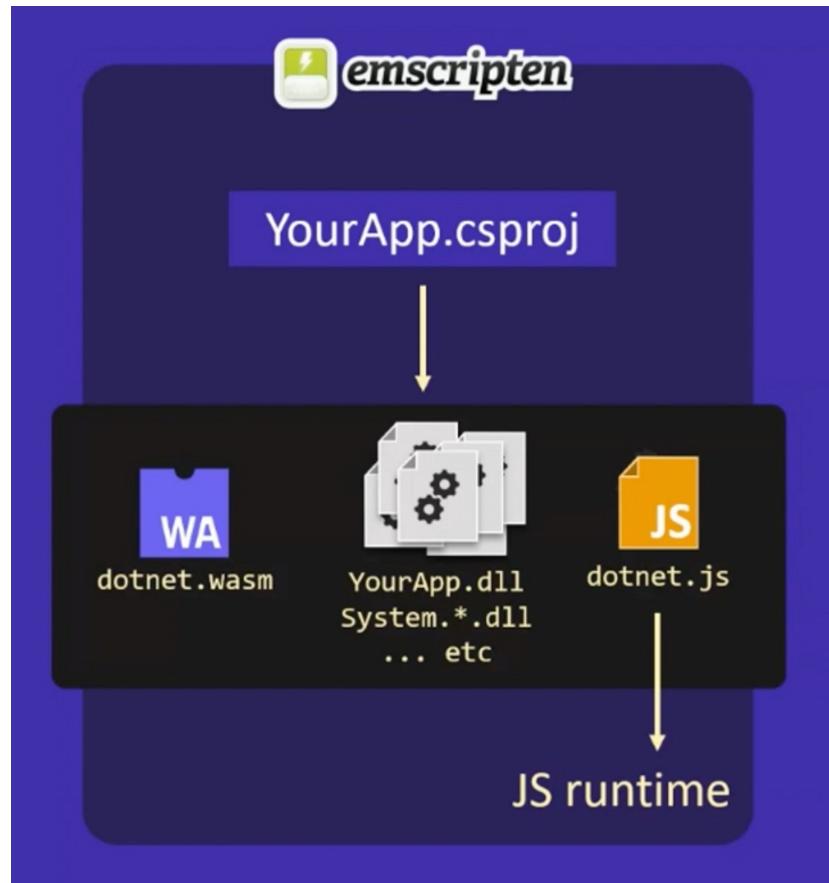


Running .NET on WebAssembly



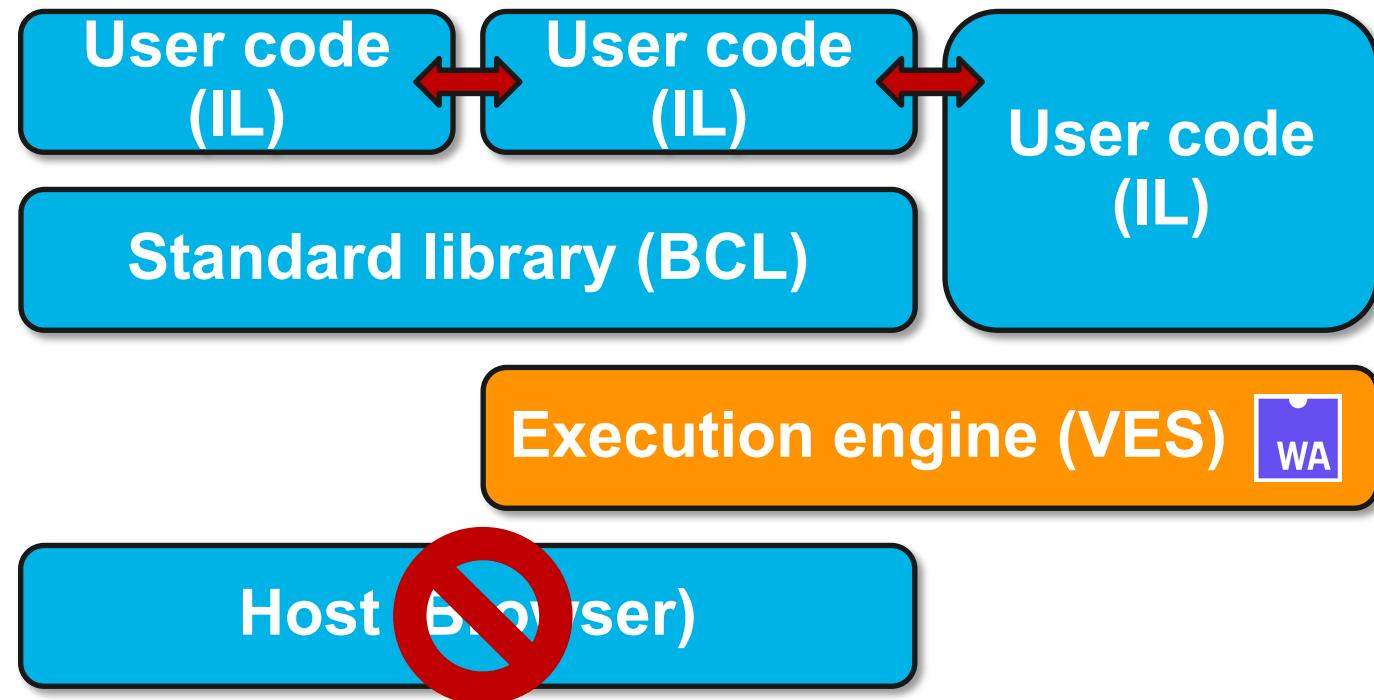
0101
0101

Blazor WebAssembly





Running .NET on WebAssembly



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

WebAssembly System Interface WASI



- Strong sandbox with Capability Based Security
- Right now, supports e.g. FileSystem actions
- Future support for sockets and other system resources.
- Anyone recall .NET Standard? ☺

0101
0101

Docker vs WASM & WASI

A screenshot of a Twitter web client window. The URL in the address bar is <https://twitter.com/s...>. The tweet is from **Solomon Hykes** (@solomonstre) 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!

Below the tweet is a reply from **Lin Clark** (@linclark) dated 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...

[Deze collectie weergeven](#)

The timestamp at the bottom of the tweet is 9:39 p.m. · 27 mrt. 2019 · Twitter Web Client.

0101
0101

Docker vs WASM & WASI

A screenshot of a Twitter web application window. The tweet is from Solomon Hykes (@solomonstre) dated March 27, 2019. The tweet content is: "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 tweet includes a reply from Solomon Hykes (@solomonstre) dated March 28, 2019, stating: "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/status/110844444444444".

At the bottom of the tweet, there is a link to "Deze collectie weergeven".

Below the tweet, the timestamp is 4:50 a.m. · 28 mrt. 2019 · Twitter Web App. The engagement metrics are 56 Retweets, 5 Geciteerde Tweets, and 165 Vind-ik-leuks.

Docker & WASM

0101
0101

The screenshot shows a web browser window with the title "Introducing the Docker+Wasm Technical Preview". The page features a purple header bar with the text "Wasm is a fast, light alternative to Linux containers — try it out today in the Docker+Wasm Technical Preview". Below this is the Docker+Wasm logo. The main content area has a dark blue background with the heading "Introducing the Docker+Wasm Technical Preview" in large white font. At the bottom left is a circular profile picture of Michael Irwin, followed by his name "MICHAEL IRWIN" and the date "Oct 24 2022". A text block 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 web browser window with the same title as the first screenshot. The main content area contains a diagram illustrating the Docker Engine architecture for Docker+Wasm. The diagram shows the "Docker Engine" at the top, which interacts with a "containerd" component. The "containerd" component oversees three separate container instances. Each instance consists of a "containerd-shim" layer, a "runc" layer, and a "Container process" layer. The middle instance also includes a "wasmedge" layer and a "Wasm Module" layer, connected via a "containerd-wasm-shim" layer.

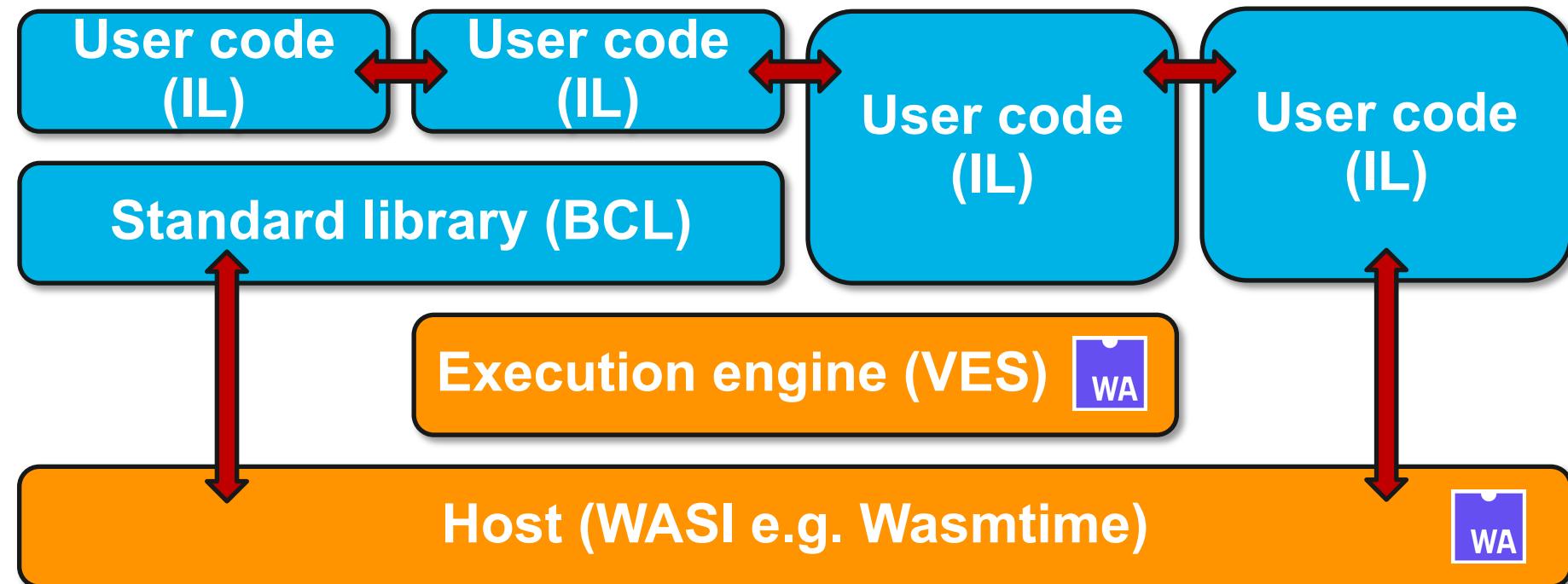
Let's look at an example!

After installing the preview, we can run the following command to start an example Wasm application:

```
docker run -dp 8080:8080 --name=wasm-example --runtime=io.containerd.wasmedge.v1 --platform=wasi/wasm32 michaelirwin244/wasm-example
```



WebAssembly System Interface WASI



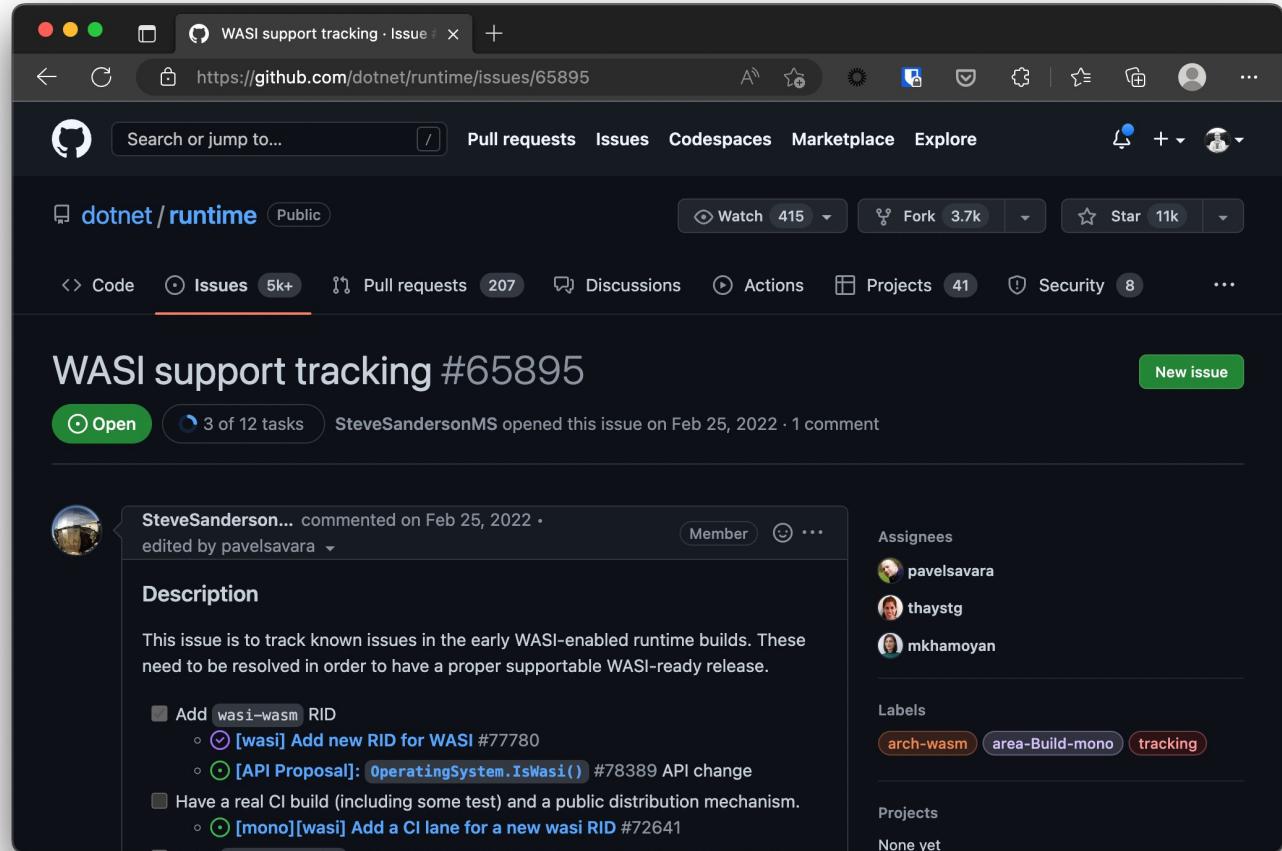


Experimental WASI SDK for .NET



0101
0101

Experimental WASI SDK for .NET





Extending .NET with WASM

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



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



Conclusion

- Cloud Native ❤️ WebAssembly
- WebAssembly has a lot of potential to be used to run, extend, and secure your .NET applications
- Its as secure as the WebAssembly runtime implementation!
- I like top-down approach Bytecode Alliance is taking in moving forward, feedback will change

0101
0101

Questions?

- <https://github.com/nielstanis/dotnetflix2023>
- ntanis at Veracode.com
- @nielstanis@infosec.exchange
- <https://blog.fennec.dev>
- Bedankt! Thank you!