WebAssembly Security

Fuzzing npm/nodejs WebAssembly parsing library with jsfuzz

I asked recently on twitter what should be my next blogpost subject and voters choose this one, so here it is.

In this short blogpost, I will first introduce **jsfuzz, a coverage-guided fuzzer for javascript/nodejs** packages. Then, I'll discuss about the wasm binary parsing library I decided to target. Finally, I'll explain how to **create a jsfuzz target** script and show the **OOM/DoS crash I found**.

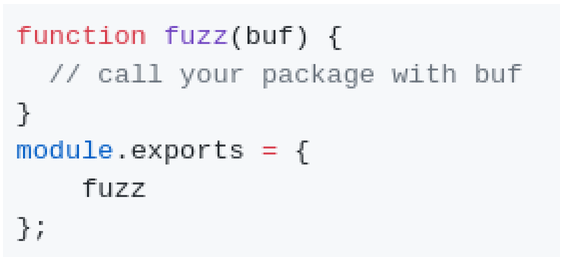
Just a quick reminder before we start, if you are interested about **WebAssembly security (both reversing and fuzzing)**, my next publics trainings will be in:

- 29 March – 01 April 2020 / 🇸🇬 Singapore ==> SHACK/WhiskeyCon.
- 20 – 22 April 2020 / 🇳🇱 Amsterdam ==> HITB.
- 01 – 04 August 2020 / 🇺🇸 Las Vegas ==> Ringzer0.
- Onsite trainings ==> here.

1. Fuzzer: Jsfuzz

Jsfuzz is a **coverage-guided fuzzer for javascript/nodejs packages** developped by Fuzzit. The typical bugs found by jsfuzz are **unhandled exceptions, logic bugs, Denial-of-Service** (DoS) caused by hangs and **excessive memory usage** (OOM).

Jsfuzz is both **user-friendly** and **efficient**, especially when fuzzing parsing libraries. You will only need to implement the following function to start fuzzing your target. All the fuzzing/**mutation logic**, **coverage collection** and **crash detection** will be handle by jsfuzz.



```
function fuzz(buf) {
  // call your package with buf
}
module.exports = {
    fuzz
};
```

Minimal custom jsfuzz target script

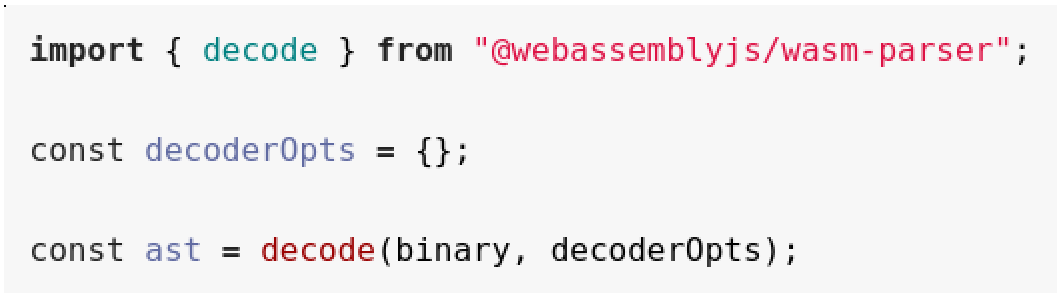
---

2. Target: @webassemblyjs/wasm-parser

webassemblyjs is a set of 22 dedicated packages for WebAssembly module manipulation. Most of them are really popular npm package with **more than 6 millions weekly downloads** each, like those ones:

- @webassemblyjs/ast: AST utils for webassemblyjs
- @webassemblyjs/wasm-parser: WebAssembly binary format parser
- @webassemblyjs/wast-parser: WebAssembly text format parser

I decided to target @webassemblyjs/wasm-parser because the API is really simple and also because I think jsfuzz mutation algorithm will be more efficient on binary file.

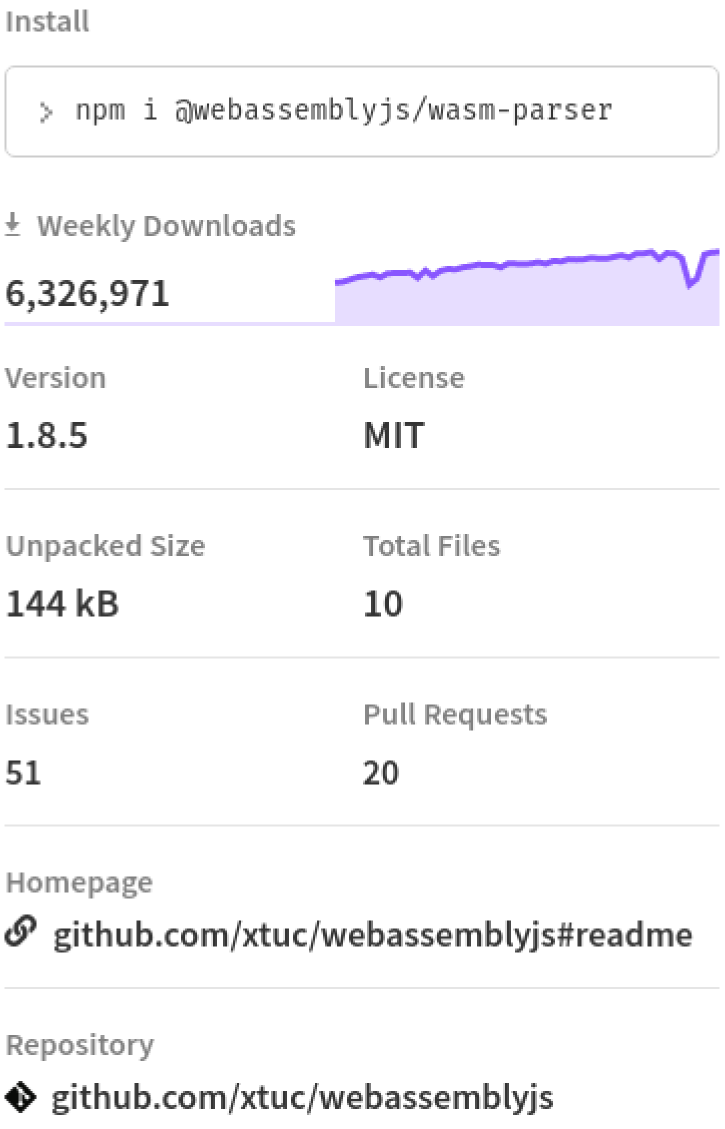


```
import { decode } from "@webassemblyjs/wasm-parser";

const decoderOpts = {};

const ast = decode(binary, decoderOpts);
```

wasm-parser npm package simple API


Install

```
> npm i @webassemblyjs/wasm-parser
```

Weekly Downloads

6,326,971

Version	License
1.8.5	MIT
Unpacked Size	Total Files
144 kB	10
Issues	Pull Requests
51	20

Homepage

github.com/xtuc/webassemblyjs#readme

Repository

github.com/xtuc/webassemblyjs

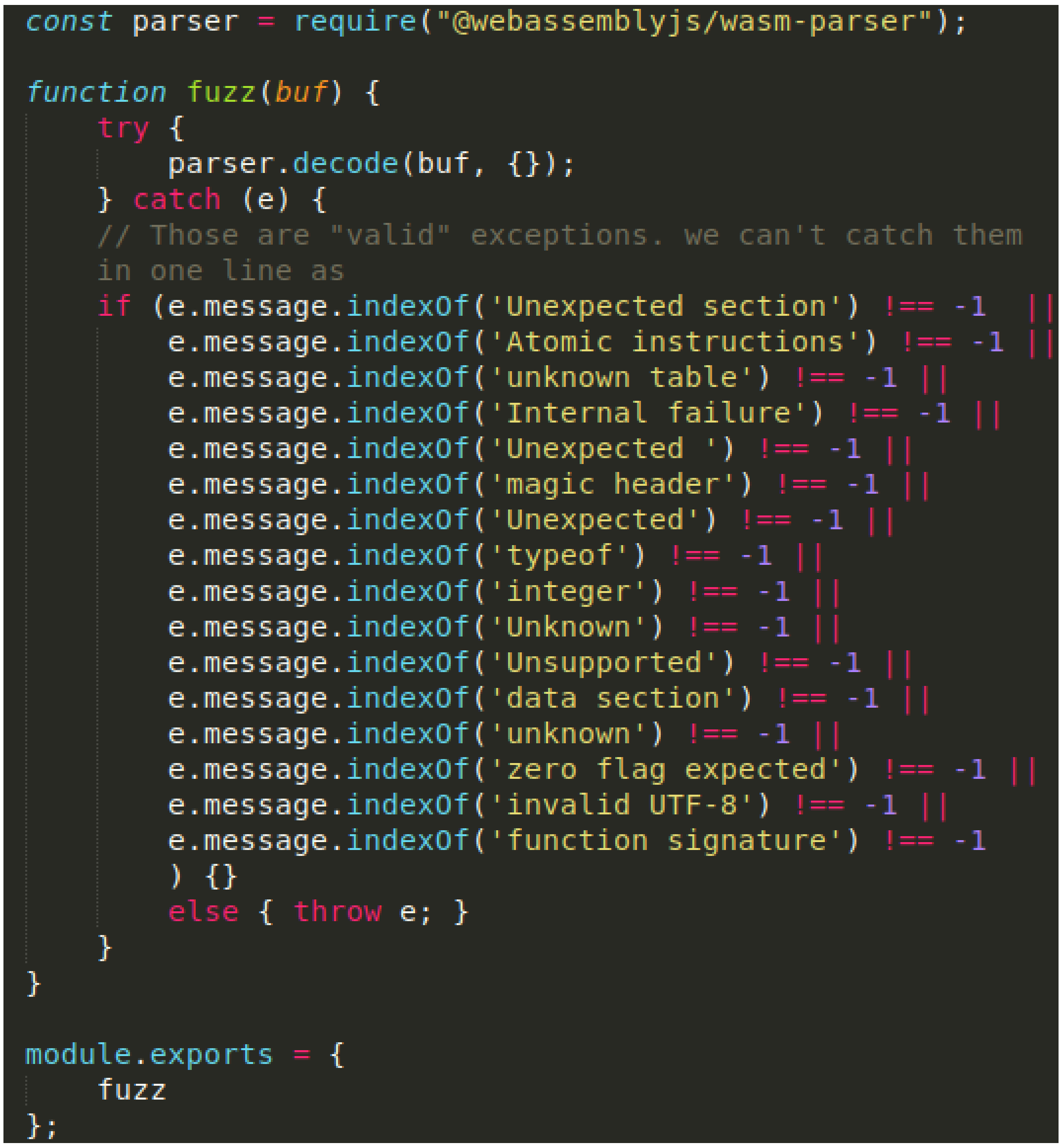
3. Setup the fuzz target script

The process to setup a fuzz target script with jsfuzz is extremely simple. The first step is to create a corpus of valid WebAssembly module like the ones on Mozilla github repository: MDN webassembly-examples.

Then, run the fuzzer with the following command:

```
jsfuzz fuzz-wasm-parser.js corpus-wasm
```

You should immediately **trigger some valid error/exception** and you will need to **customized** the fuzz target to **catch and ignored** them like in the following code (available here).



```
const parser = require("@webassemblyjs/wasm-parser");

function fuzz(buf) {
    try {
        parser.decode(buf, {});
    } catch (e) {
    // Those are "valid" exceptions. we can't catch them
    in one line as
    if (e.message.indexOf('Unexpected section') !== -1  ||
        e.message.indexOf('Atomic instructions') !== -1 ||
        e.message.indexOf('unknown table') !== -1 ||
        e.message.indexOf('Internal failure') !== -1 ||
        e.message.indexOf('Unexpected ') !== -1 ||
        e.message.indexOf('magic header') !== -1 ||
        e.message.indexOf('Unexpected') !== -1 ||
        e.message.indexOf('typeof') !== -1 ||
        e.message.indexOf('integer') !== -1 ||
        e.message.indexOf('Unknown') !== -1 ||
        e.message.indexOf('Unsupported') !== -1 ||
        e.message.indexOf('data section') !== -1 ||
        e.message.indexOf('unknown') !== -1 ||
        e.message.indexOf('zero flag expected') !== -1 ||
        e.message.indexOf('invalid UTF-8') !== -1 ||
        e.message.indexOf('function signature') !== -1
        ) {}
        else { throw e; }
    }
}

module.exports = {
    fuzz
};
```

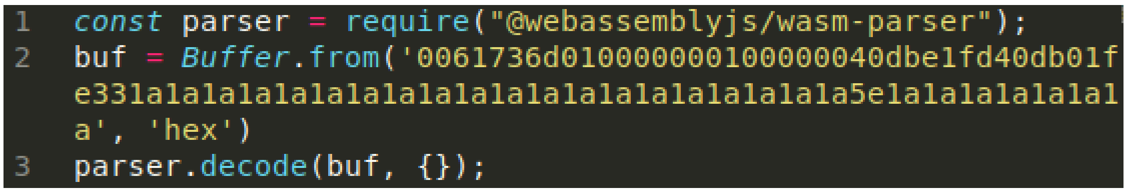
wasm-parser complete fuzz target

---

4. Result: OOM/DoS of nodejs triggered

Once most common exceptions are handled using try/catch, jsfuzz will start to **generate fuzzing status logs** and **eventually crash** like the following picture.

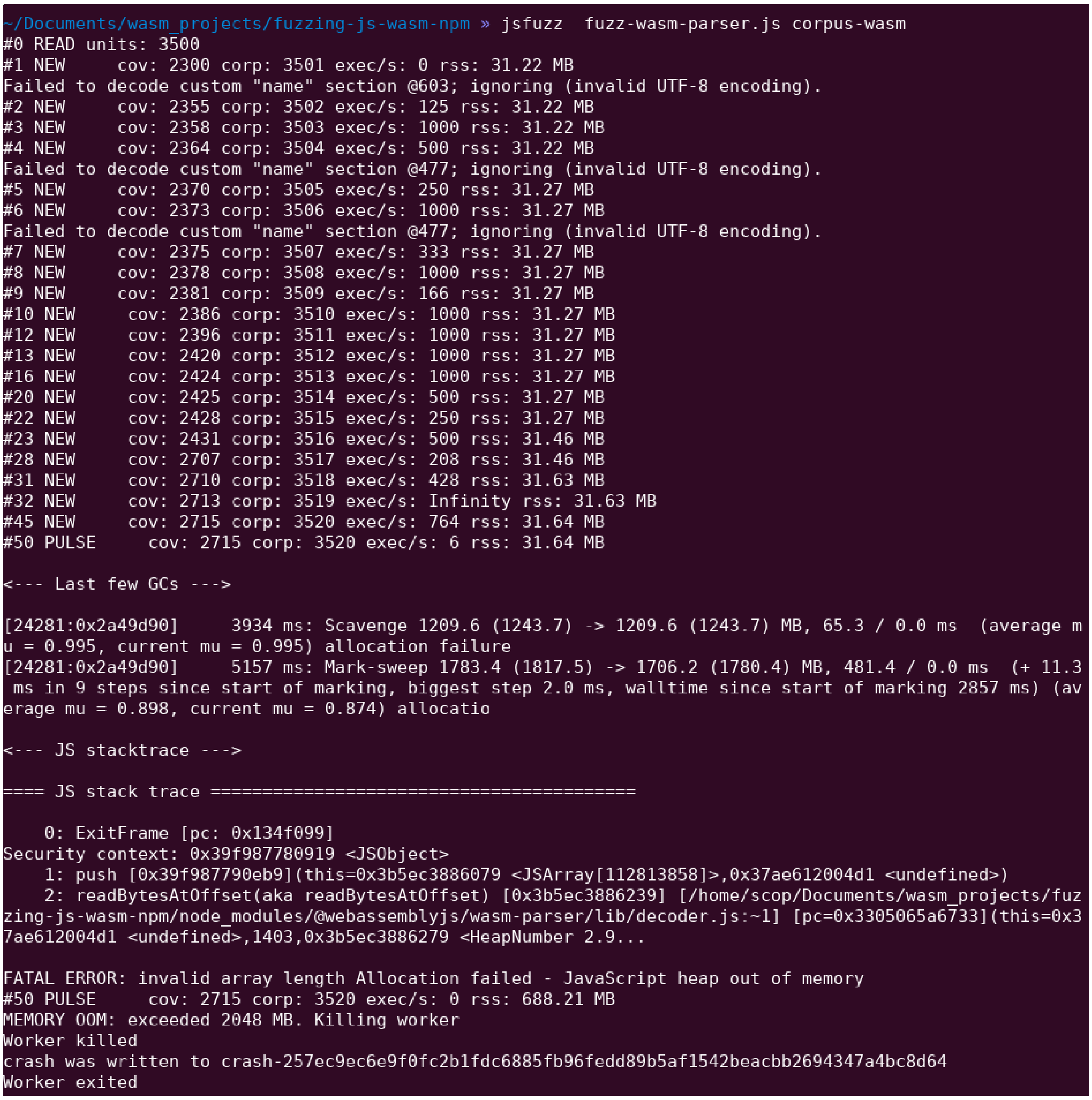
This bug is triggered really quickly (less than a minute) by the fuzzer with my wasm module corpus.
I quickly **minimized the crashing buffer**, create a **reproducer JS** file and directly **open an issue** on webassemblyjs github repository.



```
const parser = require("@webassemblyjs/wasm-parser");
buf = Buffer.from('0061736d010000000100000040dbe1fd40db01f
e331a1a1a1a1a1a1a1a1a1a1a1a1a1a1a1a1a1a1a1a5e1a1a1a1a1a1a1
a', 'hex')
parser.decode(buf, {});
```

Reproducer for DoS/OOM bugs found

In October 2019, Fuzzit team **found another bug** in this library but not in the same npm package: @webassemblyjs/wast-parser: Crash/TypeError.



```
~/Documents/wasm_projects/fuzzing-js-wasm-npm » jsfuzz  fuzz-wasm-parser.js corpus-wasm
#0 READ units: 3500
#1 NEW     cov: 2300 corp: 3501 exec/s: 0 rss: 31.22 MB
Failed to decode custom "name" section @603; ignoring (invalid UTF-8 encoding).
#2 NEW     cov: 2355 corp: 3502 exec/s: 125 rss: 31.22 MB
#3 NEW     cov: 2358 corp: 3503 exec/s: 1000 rss: 31.22 MB
#4 NEW     cov: 2364 corp: 3504 exec/s: 500 rss: 31.22 MB
Failed to decode custom "name" section @477; ignoring (invalid UTF-8 encoding).
#5 NEW     cov: 2370 corp: 3505 exec/s: 250 rss: 31.27 MB
#6 NEW     cov: 2373 corp: 3506 exec/s: 1000 rss: 31.27 MB
Failed to decode custom "name" section @477; ignoring (invalid UTF-8 encoding).
#7 NEW     cov: 2375 corp: 3507 exec/s: 333 rss: 31.27 MB
#8 NEW     cov: 2378 corp: 3508 exec/s: 1000 rss: 31.27 MB
#9 NEW     cov: 2381 corp: 3509 exec/s: 166 rss: 31.27 MB
#10 NEW     cov: 2386 corp: 3510 exec/s: 1000 rss: 31.27 MB
#12 NEW     cov: 2396 corp: 3511 exec/s: 1000 rss: 31.27 MB
#13 NEW     cov: 2420 corp: 3512 exec/s: 1000 rss: 31.27 MB
#16 NEW     cov: 2424 corp: 3513 exec/s: 1000 rss: 31.27 MB
#20 NEW     cov: 2425 corp: 3514 exec/s: 500 rss: 31.27 MB
#22 NEW     cov: 2428 corp: 3515 exec/s: 250 rss: 31.27 MB
#23 NEW     cov: 2431 corp: 3516 exec/s: 500 rss: 31.46 MB
#28 NEW     cov: 2707 corp: 3517 exec/s: 208 rss: 31.46 MB
#31 NEW     cov: 2710 corp: 3518 exec/s: 428 rss: 31.63 MB
#32 NEW     cov: 2713 corp: 3519 exec/s: Infinity rss: 31.63 MB
#45 NEW     cov: 2715 corp: 3520 exec/s: 764 rss: 31.64 MB
#50 PULSE     cov: 2715 corp: 3520 exec/s: 6 rss: 31.64 MB

<--- Last few GCs --->

[24281:0x2a49d90]     3934 ms: Scavenge 1209.6 (1243.7) -> 1209.6 (1243.7) MB, 65.3 / 0.0 ms  (average m
u = 0.995, current mu = 0.995) allocation failure
[24281:0x2a49d90]     5157 ms: Mark-sweep 1783.4 (1817.5) -> 1706.2 (1780.4) MB, 481.4 / 0.0 ms  (+ 11.3
 ms in 9 steps since start of marking, biggest step 2.0 ms, walltime since start of marking 2857 ms) (av
erage mu = 0.898, current mu = 0.874) allocatio

<--- JS stacktrace --->

==== JS stack trace =========================================

    0: ExitFrame [pc: 0x134f099]
Security context: 0x39f987780919 <JSObject>
    1: push [0x39f987790eb9](this=0x3b5ec3886079 <JSArray[112813858]>,0x37ae612004d1 <undefined>)
    2: readBytesAtOffset(aka readBytesAtOffset) [0x3b5ec3886239] [/home/scop/Documents/wasm_projects/fuz
zing-js-wasm-npm/node_modules/@webassemblyjs/wasm-parser/lib/decoder.js:~1] [pc=0x3305065a6733](this=0x3
7ae612004d1 <undefined>,1403,0x3b5ec3886279 <HeapNumber 2.9...

FATAL ERROR: invalid array length Allocation failed - JavaScript heap out of memory
#50 PULSE     cov: 2715 corp: 3520 exec/s: 0 rss: 688.21 MB
MEMORY OOM: exceeded 2048 MB. Killing worker
Worker killed
crash was written to crash-257ec9ec6e9f0fc2b1fdc6885fb96fedd89b5af1542beacbb2694347a4bc8d64
Worker exited
```

jsfuzz crash after less than a minute

5. Conclusion

I strongly invite every **JavaScript/nodejs developer to give a try to Jsfuzz**. It will help you **finding bugs**, create **edge-cases inputs** and easily get **code coverage visibility**. Kudos again to Fuzzit for sharing this fuzzer publicly.

Final reminder, If you want to learn/discover more about **WebAssembly security (both reversing and fuzzing)**, my next public trainings will be in:

- 29 March – 01 April 2020 / 🇸🇬 Singapore ==> SHACK/WhiskeyCon.
- 20 – 22 April 2020 / 🇳🇱 Amsterdam ==> HITB.
- 01 – 04 August 2020 / 🇺🇸 Las Vegas ==> Ringzer0.
- Onsite trainings ==> here.

If you want to contact me for consulting, please DM me on Twitter/LinkedIn or use the following contact form.

Patrick Ventuzelo / @Pat_Ventuzelo

UPCOMING TRAININGS

REQUEST ON-SITE QUOTE