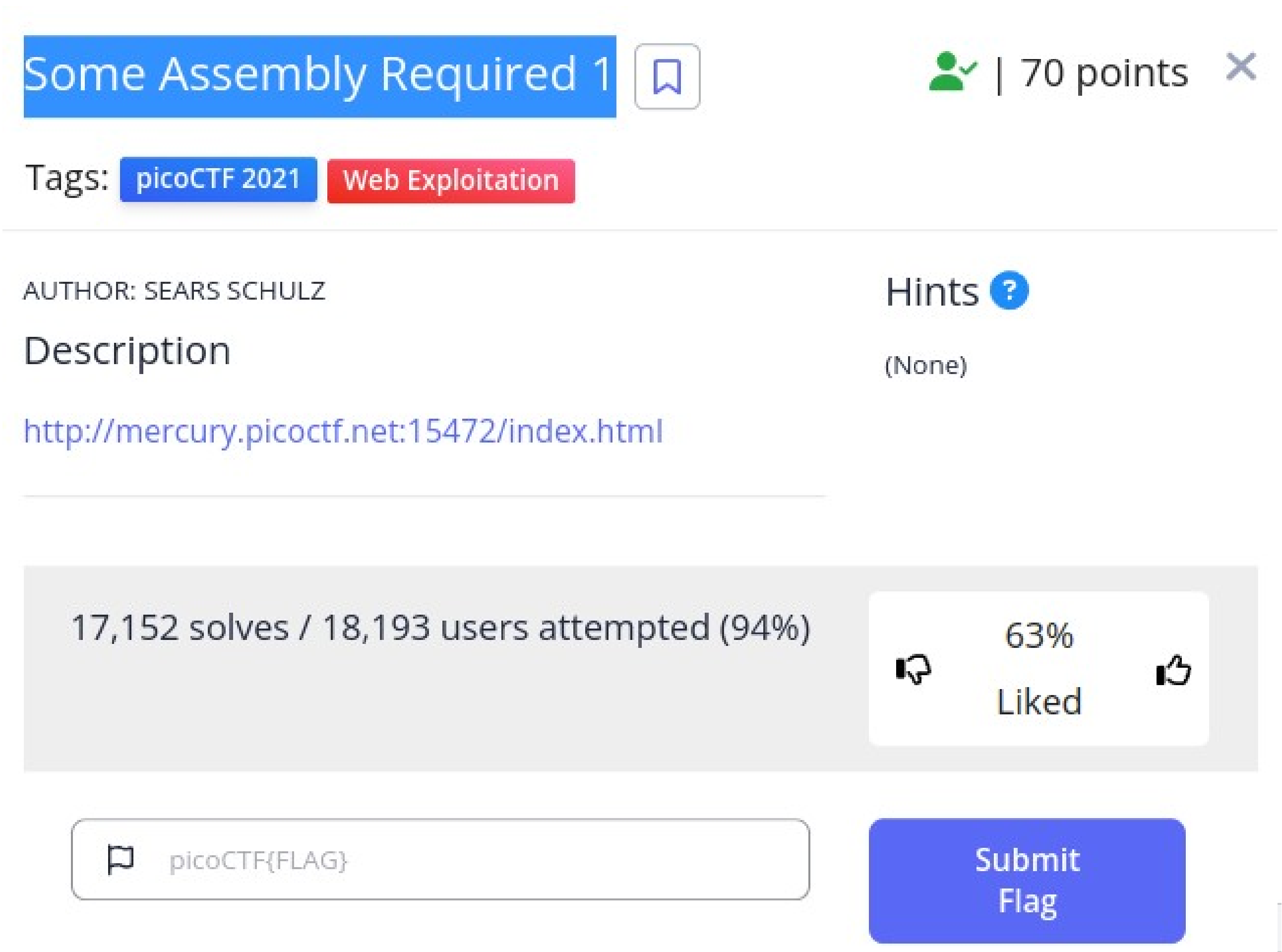
# Some Assembly Required 1



Some Assembly Required 1

| 70 points

Tags: picoCTF 2021 Web Exploitation

AUTHOR: SEARS SCHULZ

Description

http://mercury.picoctf.net:15472/index.html

Hints

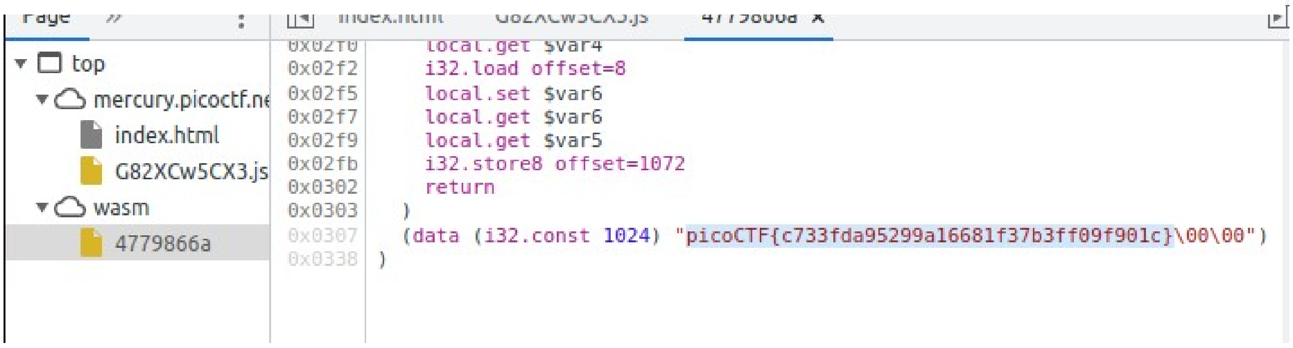
(None)

17,152 solves / 18,193 users attempted (94%)

63% Liked

picoCTF{FLAG}

Submit Flag



```
0x02f0      local.get $var4
0x02f2      i32.load offset=8
0x02f5      local.set $var6
0x02f7      local.get $var6
0x02f9      local.get $var5
0x02fb      i32.store8 offset=1072
0x0302      return
0x0303    )
0x0307    (data (i32.const 1024) "picoCTF{c733fda95299a16681f37b3ff09f901c}\00\00")
0x0338  )
```

-

WebAssembly is a binary instruction format for a stack-based virtual machine that is designed to be a portable target for the compilation of high-level languages such as C, C++, and Rust, among others. It is supported by all major web browsers and allows code written in languages other than JavaScript to be executed in web applications with near-native performance.

WebAssembly is a low-level, assembly-like language that is designed to run on a virtual machine, making it platform-independent and portable. It is designed to be a complement to JavaScript, not a replacement, and can be used in conjunction with JavaScript to improve performance and reduce the amount of code that needs to be written.

WebAssembly code is executed within a sandboxed environment, which provides a high level of security and prevents the execution of malicious code. It also allows for efficient code optimization and supports multiple threading, which can improve performance for certain types of applications.

Overall, WebAssembly is a powerful tool that enables developers to write high-performance web applications in a variety of languages, while still maintaining a high level of security and portability.

Suppose you have a web application that calculates the factorial of a number. You write the application in JavaScript, but you notice that it takes a long time to calculate the factorial of large numbers. You can use WebAssembly to speed up the calculation.

First, you write a function to calculate the factorial of a number in C:

```
int factorial(int n) {
  if (n == 0) {
    return 1;
  } else {
    return n * factorial(n - 1);
  }}
```

This function takes an integer `n` and recursively calculates its factorial. You then compile this function into WebAssembly using a tool like Emscripten:

```
emcc factorial.c -o factorial.wasm -s WASM=1
```

This generates a file named `factorial.wasm` that contains the compiled WebAssembly code.

Next, you can use this WebAssembly module in your web application by loading it using the `WebAssembly.instantiateStreaming()` function:

```
fetch('factorial.wasm')
  .then(response => response.arrayBuffer())
  .then(buffer => WebAssembly.instantiate(buffer))
  .then(module => {
    const factorial = module.instance.exports.factorial;
    console.log(factorial(5)); // Output: 120
  });
```

In this code, we use `fetch()` to load the `factorial.wasm` file, convert it to an array buffer, and then instantiate it using `WebAssembly.instantiate()`. We can then call the `factorial()` function from the WebAssembly module, passing in the value `5` to calculate the factorial of 5.

Using WebAssembly in this way can greatly improve the performance of your web application, as you can write high-performance code in languages like C and Rust and execute it directly in the browser.