# Q3 Reverse engineer

WebAssembly (Wasm) is a binary format that allows code to be executed efficiently in the browser. It is an important topic in the forefront of current Web security.

Source code：**[Program Language Rust](https://github.com/FrogGuaGua/SQRPRGRM/blob/main/CTFanswer%26design/Q3/source/src/lib.rs)**

Backdoor Position：**[Binary backdoor](https://github.com/Shahzeb892/websokect_chat_app/blob/main/SCCSChatRoom/src/renderer/src/utils/decoder/decoder_bg.wasm)**

## POC:

1. **Transform wasm to c code**  
   <https://github.com/WebAssembly/wabt>  
   wasm2c decoder\_bg.wasm >> decoder.c  
   Now we get the deocer.c
2. **Compile decoder.c to decoder.o**

clang -c ./decoder.c -o ./decoder.o -Oz

1. **Statics analysis**

Import into analyze software (We used idapro)

Fix type and structure

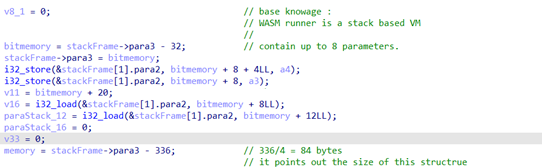


Figure 1 memory use

Now, we know decoder has 5 parameters and para3 has used 84 bytes.

A screen shot of a computer program

Description automatically generated

They are wasm vm stack operation (we will explore it at dynamic analysis

part).

**Function main logic flow:**

Loop:

Switch(para2):

case 'a':goto LABEL\_22 (goto case s);

case 'b':

case 'c':

break;

case 'd':goto LABEL\_27 (goto case s);

case 's': Code(similar to case ‘w’).

case 'w': Code;

default:goto LABEL\_38;

LABEL\_38:  
 A screenshot of a computer code

Description automatically generated

Figure 2 Label\_38

L38: if para2

       jsFunction() // go to label L38 represents an error condition.

Return

Case w:

A computer code on a white background

Description automatically generated

Figure 3 case w

From Figure 3, case w is checking the array bounds and will return to case s (label 31) and we can assume there is a 9x9 array.

Label31:

A white screen with text and numbers

Description automatically generated

Figure 4 backdoor and condition

Label 31 displays a function generated by [wasm-bindgen](https://rustwasm.github.io/wasm-bindgen/), along with its loop condition.

**Conclusion:**

The decoder only accepts 'wasd' as input and performs some operations in a 9x9 array. When the coordinates are [0,6], a vulnerability will be triggered.

1. **Dynamic analysis**

Debugger: Chrome Browser Break at decoder call  
Variable layout:

A screenshot of a computer code

Description automatically generated

Figure 5 WASM var layout

A screenshot of a computer

Description automatically generated

Figure 6 Array addressing

Now, we know the 9x9 array is near at 1048204.

Write a script to show it.

const buffer = new Uint8Array(memories.$memory.buffer);

const offset = 1048204;

const length = 81\*4;

const data = buffer.slice(offset, offset + length);

function printBytesInVM(bytes, groupSize) {

  for (let i = 0; i < bytes.length; i += groupSize) {

    const group = bytes.slice(i, i + groupSize);

    console.log(group.map(byte => byte.toString(16).padStart(2, '0')).join(' '));

  }

}

printBytesInVM(data, 36);

A white background with black numbers

Description automatically generated

Figure 7 memory

Convert it into a more readable arrary.

        [0, 1, 0, 1, 0, 1, 0, 1, 1],

        [0, 1, 0, 1, 0, 1, 0, 0, 0],

        [0, 0, 0, 0, 0, 1, 1, 1, 0],

        [1, 1, 1, 1, 0, 1, 0, 1, 0],

        [0, 0, 0, 1, 0, 1, 0, 0, 0],

        [1, 1, 0, 1, 0, 1, 0, 1, 1],

        [0, 1, 0, 1, 0, 1, 0, 0, 0],

        [0, 1, 0, 0, 0, 1, 1, 1, 0],

        [0, 0, 0, 1, 0, 0, 0, 0, 0],

1. **Analyze summary**

The decoder function searches for the destination [0,6] in a 9x9 maze based on the input. It moves according to the wasd commands and will exit if it goes out of bounds or collides with a wall during the process.

1. **Solution**

ssddddssssssddddwwaawwddwwwaaw + any comment

1. **Verification**

A screenshot of a computer

Description automatically generated

Figure 8 Backdoor triggered

We did it!