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**
Backdoor Position: **Binary backdoor**

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

3. **Statics analysis**
   Import into analyze software (We used idapro)
   Fix type and structure

```
v8_1 = 0;                               // base knowage :
                                        // WASM runner is a stack based VM
                                        //
bitmemory = stackFrame->para3 - 32;     // contain up to 8 parameters.
stackFrame->para3 = bitmemory;
i32_store(&stackFrame[1].para2, bitmemory + 8 + 4LL, a4);
i32_store(&stackFrame[1].para2, bitmemory + 8, a3);
v11 = bitmemory + 20;
v16 = i32_load(&stackFrame[1].para2, bitmemory + 8LL);
paraStack_12 = i32_load(&stackFrame[1].para2, bitmemory + 12LL);
paraStack_16 = 0;
v33 = 0;
memory = stackFrame->para3 - 336;       // 336/4 = 84 bytes
                                        // it points out the size of this structrue
```

Figure 1 memory use

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

```
i64_store(&stackFrame[1].para2, memory + 72, 0LL);
i32_store(&stackFrame[1].para2, memory + 68, 1LL);
i64_store(&stackFrame[1].para2, memory + 60, 1LL);
i64_store(&stackFrame[1].para2, memory + 52, 1LL);
i64_store(&stackFrame[1].para2, memory + 80, 0LL);
i64_store(&stackFrame[1].para2, memory + 88, 0LL);
i64_store(&stackFrame[1].para2, memory + 96, 0LL);
i64_store(&stackFrame[1].para2, memory + 104, 0x100000001LL);
i64_store(&stackFrame[1].para2, memory + 112, 1LL);
i64_store(&stackFrame[1].para2, memory + 128, 0x100000001LL);
i64_store(&stackFrame[1].para2, memory + 136, 0x100000000LL);
i64_store(&stackFrame[1].para2, memory + 144, 0x100000000LL);
i64_store(&stackFrame[1].para2, memory + 44LL, 1LL);
i64_store(&stackFrame[1].para2, memory + 36LL, 0x100000000LL);
i64_store(&stackFrame[1].para2, memory + 28LL, 0x100000000LL);
i64_store(&stackFrame[1].para2, memory + 20LL, 0x100000000LL);
i64_store(&stackFrame[1].para2, memory + 12LL, 0x100000000LL);
i64_store(&stackFrame[1].para2, memory + 120LL, 0x100000001LL);
i64_store(&stackFrame[1].para2, memory + 160, 0LL);
i64_store(&stackFrame[1].para2, memory + 152, 0LL);
i64_store(&stackFrame[1].para2, memory + 240, 1LL);
i64_store(&stackFrame[1].para2, memory + 232, 1LL);
i64_store(&stackFrame[1].para2, memory + 224, 1LL);
i64_store(&stackFrame[1].para2, memory + 216, 0x100000000LL);
i64_store(&stackFrame[1].para2, memory + 208, 0x100000000LL);
i64_store(&stackFrame[1].para2, memory + 200, 0x100000000LL);
i64_store(&stackFrame[1].para2, memory + 184, 0LL);
```

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:

```
LABEL_38:
  if ( para2 )
  {
    v34 = 0;
    if ( (para2 & 0x80000000) != 0
      || (i32_load8_u(&stackFrame[1].para2, 1052565LL), v34 = 1, (paraStack_16e = w2c__f37(stackFrame, para2, 1LL)) == 0) )
    {
      w2c__f33(stackFrame, v34, para2);
      wasm_rt_trap(5LL);
      goto LABEL_44;
    }
  }
  else
  {
    paraStack_16e = 1;
  }
  v39 = w2c__f51(stackFrame, paraStack_16e, v16, para2);
  i32_store(&stackFrame[1].para2, v11 + 8LL, para2);
  i32_store(&stackFrame[1].para2, v11 + 4LL, v39);
  i32_store(&stackFrame[1].para2, v11, para2);
  stackFrame->para3 = memory + 336;
  if ( para2 )
    jsFunction(stackFrame, v16, para2);
  v40 = bitmemory + 20;
  v35 = i32_load(&stackFrame[1].para2, bitmemory + 20 + 8LL);
```

Figure 2 Label_38

```
L38: if para2
       jsFunction()        // go to label L38 represents an error condition.
Return
```

Case w:

```
EL_22:
          if ( !j )
            goto LABEL_38;
          paraStack_16d = j - 1;
          if ( j > 9 )
          {
            jsFunctionError(stackFrame, paraStack_16d, 9LL, 1048620LL);// //arrary out of bound
            wasm_rt_trap(5LL);
EL_27:
            if ( j >= 8 )
              goto LABEL_38;
            ++j;                            // It is 9x9 arrary
                                            // if array[i][j]:
                                            //    error
            if ( i32_load(&stackFrame[1].para2, v31 + 12 + 36 * i + 4 * j) )
              goto LABEL_38;
          }
```

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:

```
        para3 = v14 - paraStack_4 + v10;
        if ( i | (j != 6) )                 // if i==0 j == 6
                                            // it will stop and execute wbg_alert
          continue;
        v38 = v14 + 1;
        if ( v14 == -1 )
          goto LABEL_37;
        if ( v38 < para2 )
        {
          if ( i32_load8_s(&stackFrame[1].para2, v38 + v16) <= 4294967231 )
            goto LABEL_44;
        }
        else if ( v38 != para2 )
        {
44:
          jsFunctionError2(stackFrame, v16, para2, v38, para2, 0x10003Cu);
        }
37:
        wbg_alert(*&stackFrame->para1, v38 + v16, para2 - v38);// js alert
        break;
```

Figure 4 backdoor and condition

Label 31 displays a function generated by 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.

4. Dynamic analysis

Debugger: Chrome Browser Break at decoder call

Variable layout:

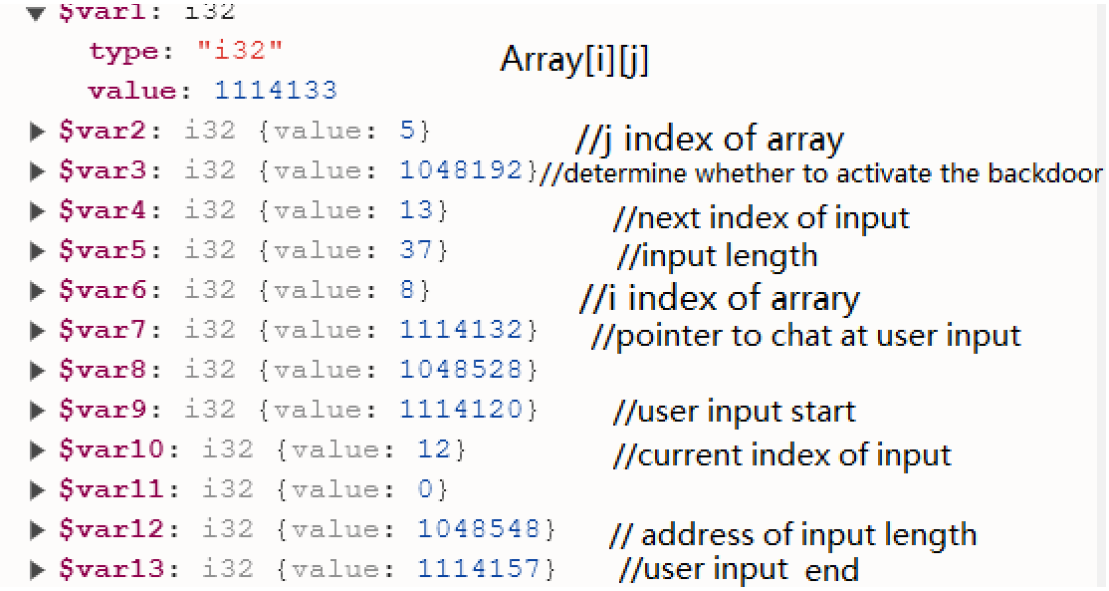

Figure 5 WASM var layout

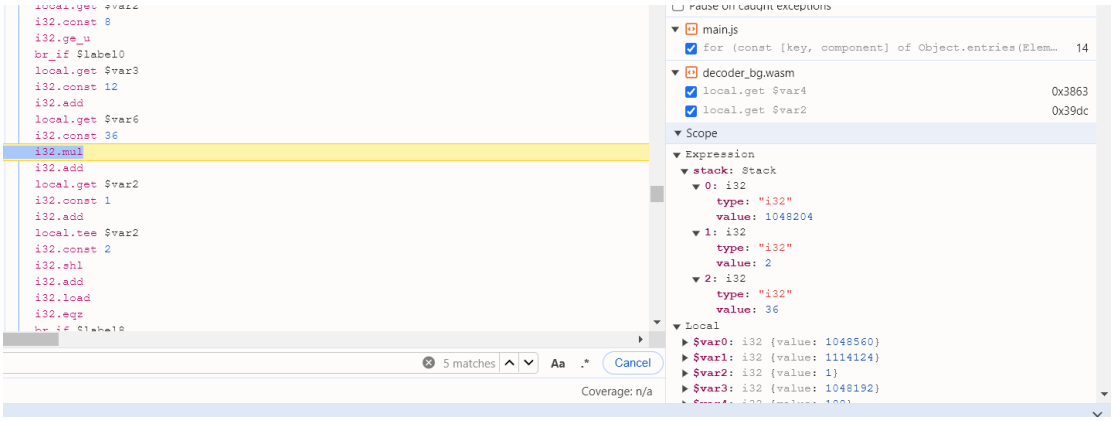

Figure 6 Array addressing

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);
```

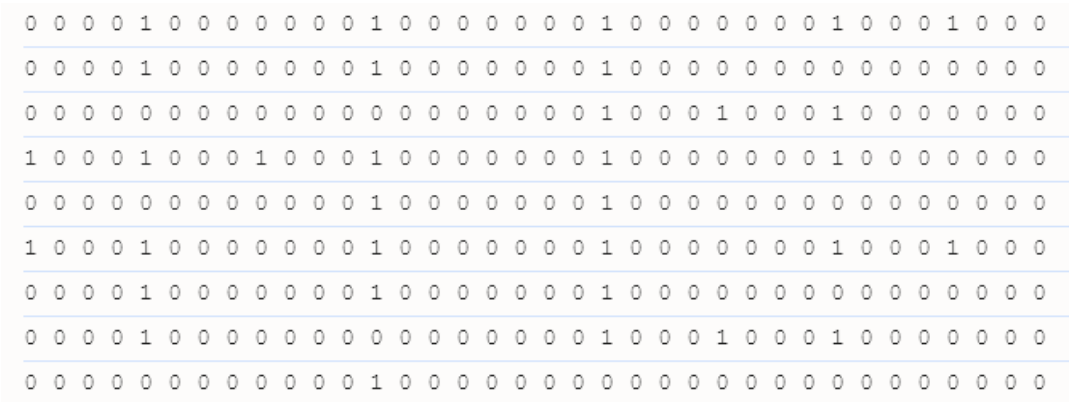

Figure 7 memory

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],
```

5. **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.

6. **Solution**

   ssddddsssssddddwwaawwddwwwaaw + any comment

7. **Verification**

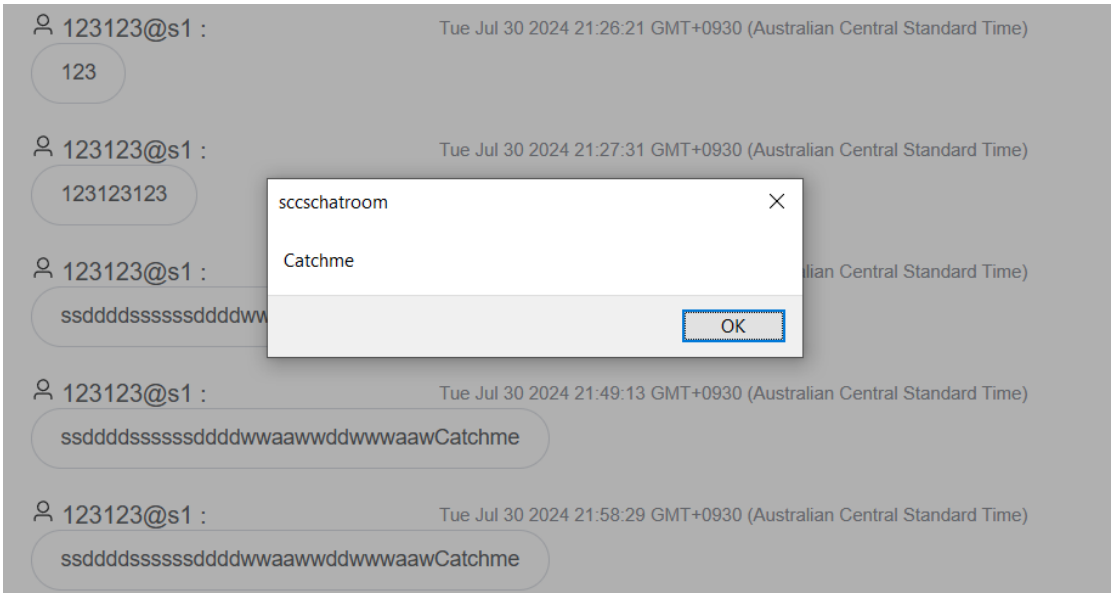

Figure 8 Backdoor triggered

We did it!