Web Assembly: Stack Buffer Overflow

Mitigation of Vulnerabilities

Lex King
Department of Computer Science, University of New Hampshire

Introduction

- ❑ Web Assembly also known as WASM is one of the newest forms of programming languages created with the specific purpose of running on virtual machines and client and server based applications in order to facilitate faster and more efficient interactions and connections.
- ❑ With new technology there are always issues that arise. The issues that arise with Web Assembly come in the form of different vulnerabilities.

Background

Web Assembly is announced in 2015

↓

Web Assembly launches in March of 2017

↓

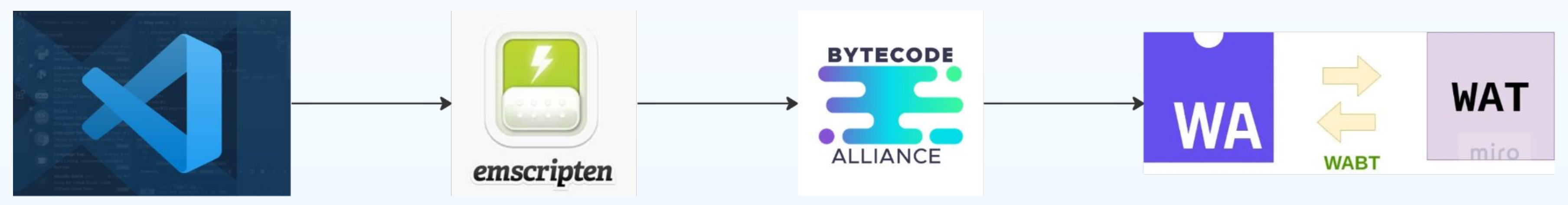
Browsers start updating to support Web Assembly in late 2017

↓

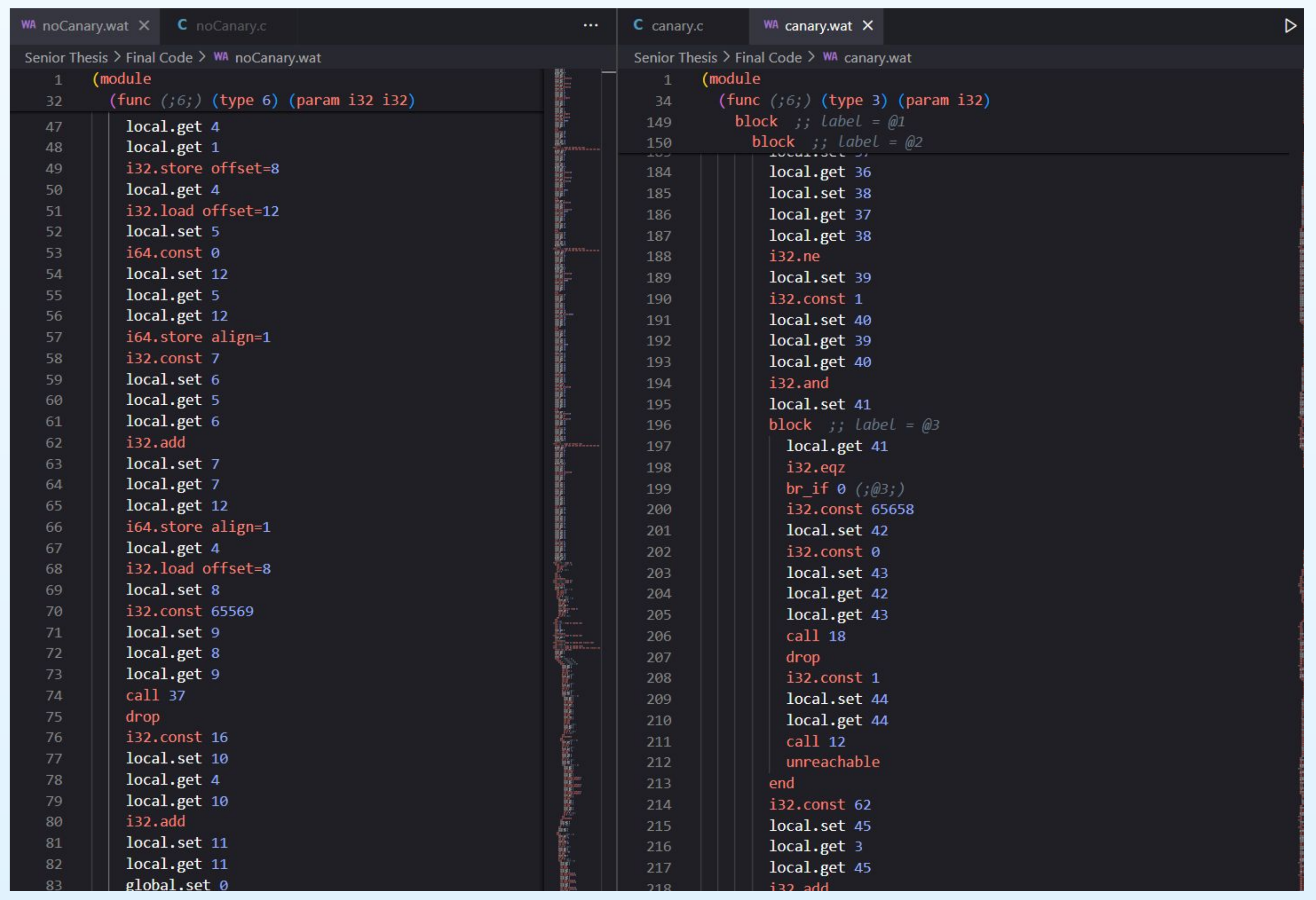
In 2019 Web Assembly vulnerabilities start being exploited

↓

2022 Web Assembly 2.0 starts getting worked on with no focus on vulnerabilities



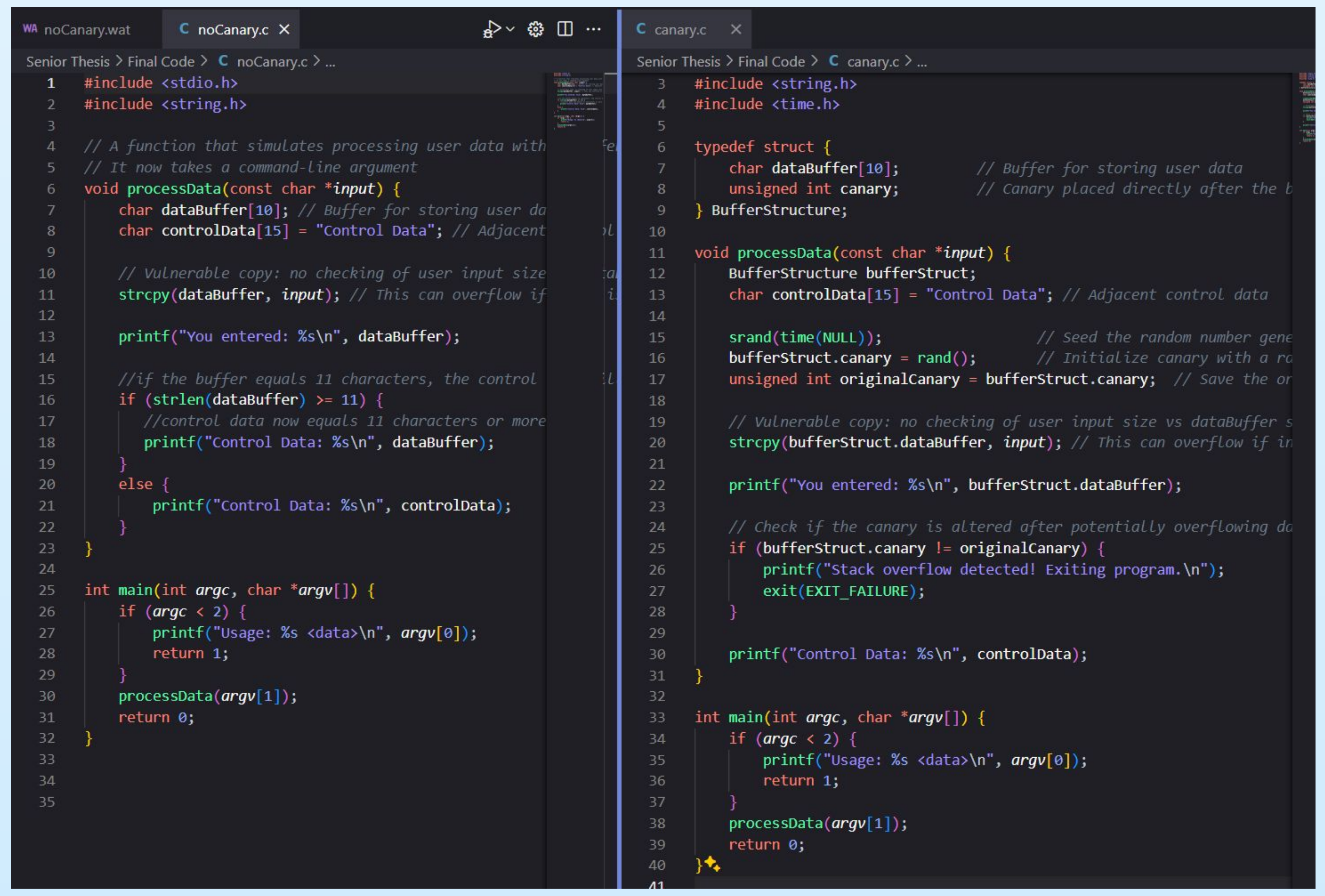
1. Environment setup



```
noCanary.wat  ×   noCanary.c
Senior Thesis > Final Code > noCanary.wat
(module
  (func (;6;) (type 6) (param i32 i32)
      local.get 4
      local.get 1
      i32.store offset=8
      local.get 4
      i32.load offset=12
      local.set 5
      i64.const 0
      local.set 12
      local.get 5
      local.get 12
      i64.store align=1
      i32.const 7
      local.set 6
      local.get 5
      local.get 6
      i32.add
      local.set 7
      local.get 7
      local.get 12
      i64.store align=1
      local.get 4
      i32.load offset=8
      local.set 8
      i32.const 65569
      local.set 9
      local.get 8
      local.get 9
      call 37
      drop
      i32.const 16
      local.set 10
      local.get 4
      local.get 10
      i32.add
      local.set 11
      local.get 11
      global.set 0
```

```
canary.c   canary.wat  ×
Senior Thesis > Final Code > canary.wat
(module
  (func (;6;) (type 3) (param i32)
    block  ;; label = @1
      block  ;; label = @2
        local.get 36
        local.set 38
        local.get 37
        local.get 38
        i32.ne
        local.set 39
        i32.const 1
        local.set 40
        local.get 39
        local.get 40
        i32.and
        local.set 41
        block  ;; label = @3
          local.get 41
          i32.eqz
          br_if 0 (;@3;)
          i32.const 65658
          local.set 42
          i32.const 0
          local.set 43
          local.get 42
          local.get 43
          call 18
          drop
          i32.const 1
          local.set 44
          local.get 44
          call 12
          unreachable
        end
        i32.const 62
        local.set 45
        local.get 3
        local.get 45
        i32.add
```

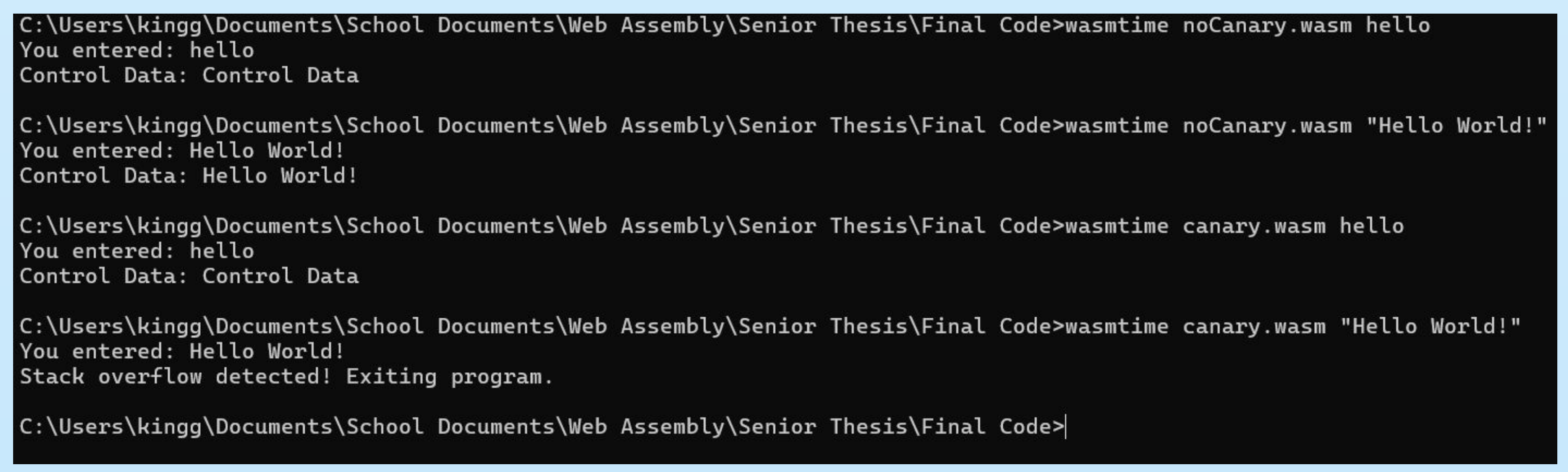
2. WASM code



```c
noCanary.wat   noCanary.c  ×
Senior Thesis > Final Code > noCanary.c > ...
#include <stdio.h>
#include <string.h>

// A function that simulates processing user data with
// It now takes a command-line argument
void processData(const char *input) {
    char dataBuffer[10]; // Buffer for storing user da
    char controlData[15] = "Control Data"; // Adjacent

    // Vulnerable copy: no checking of user input size
    strcpy(dataBuffer, input); // This can overflow if

    printf("You entered: %s\n", dataBuffer);

    //if the buffer equals 11 characters, the control
    if (strlen(dataBuffer) >= 11) {
      //control data now equals 11 characters or more
      printf("Control Data: %s\n", dataBuffer);
    }
    else {
       printf("Control Data: %s\n", controlData);
    }
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
       printf("Usage: %s <data>\n", argv[0]);
       return 1;
    }
    processData(argv[1]);
    return 0;
}
```

```c
canary.c  ×
Senior Thesis > Final Code > canary.c > ...
#include <string.h>
#include <time.h>

typedef struct {
    char dataBuffer[10];          // Buffer for storing user data
    unsigned int canary;          // Canary placed directly after the b
} BufferStructure;

void processData(const char *input) {
    BufferStructure bufferStruct;
    char controlData[15] = "Control Data"; // Adjacent control data

    srand(time(NULL));                    // Seed the random number gene
    bufferStruct.canary = rand();         // Initialize canary with a ra
    unsigned int originalCanary = bufferStruct.canary;  // Save the or

    // Vulnerable copy: no checking of user input size vs dataBuffer s
    strcpy(bufferStruct.dataBuffer, input); // This can overflow if in

    printf("You entered: %s\n", bufferStruct.dataBuffer);

    // Check if the canary is altered after potentially overflowing da
    if (bufferStruct.canary != originalCanary) {
        printf("Stack overflow detected! Exiting program.\n");
        exit(EXIT_FAILURE);
    }

    printf("Control Data: %s\n", controlData);
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        printf("Usage: %s <data>\n", argv[0]);
        return 1;
    }
    processData(argv[1]);
    return 0;
}
```

3. C version of code



```
C:\Users\kingg\Documents\School Documents\Web Assembly\Senior Thesis\Final Code>wasmtime noCanary.wasm hello
You entered: hello
Control Data: Control Data

C:\Users\kingg\Documents\School Documents\Web Assembly\Senior Thesis\Final Code>wasmtime noCanary.wasm "Hello World!"
You entered: Hello World!
Control Data: Hello World!

C:\Users\kingg\Documents\School Documents\Web Assembly\Senior Thesis\Final Code>wasmtime canary.wasm hello
You entered: hello
Control Data: Control Data

C:\Users\kingg\Documents\School Documents\Web Assembly\Senior Thesis\Final Code>wasmtime canary.wasm "Hello World!"
You entered: Hello World!
Stack overflow detected! Exiting program.

C:\Users\kingg\Documents\School Documents\Web Assembly\Senior Thesis\Final Code>
```

4. Output of both files

Results

- ❑ A successful environment where WASM code can run was set up.
- ❑ The code runs normally if within the set buffer limit.
- ❑ If outside of buffer limit for the version with no canary control data is overwritten.
- ❑ A randomized canary was created and placed between the buffer and stack.
- ❑ If outside buffer limit for the version with a stack canary then the code exits with a warning that data is being overwritten and a stack buffer overflow has occurred.

Other Common Vulnerabilities

- ❑ Heap MetaData Corruption
- ❑ Ability to overwrite stack data
- ❑ Code injection
- ❑ Remote code execution
- ❑ Stack Overflow
- ❑ Application specific data overwrite

Further Actions

- ❑ Security work to ensure protection of the stack canary.
- ❑ Brought to a larger scale and manipulated to work in multiple environments.
- ❑ All the other vulnerabilities that web assembly contains need to be addressed.

Resources

WebAssembly. https://webassembly.org/. Accessed 13 Feb. 2024.