```
Consider the following code snippet:
```

```
#include <string.h>
#include <stdlib.h>
#define BUFFER_SIZE 10
void processInput(char* input) {
   // This function adds a prefix to the input
   char prefix[] = "Data: ";
   char* combinedInput = (char*)malloc(strlen(prefix) + strlen(input) + 1);
   // Concatenate the prefix and input
   strcpy(combinedInput, prefix);
   strcat(combinedInput, input);
   printf("Combined input: %s\n", combinedInput);
void copyInput(char* userInput) {
   char buffer[BUFFER_SIZE];
   strcpy(buffer, userInput);
   printf("Buffer content: %s\n", buffer);
int main() {
   char input[50];
   printf("Enter some text: ");
    fgets(input, sizeof(input), stdin);
   // Remove the newline character from fgets
   input[strcspn(input, "\n")] = 0;
   \ensuremath{//} Process input before passing it to the other function
   processInput(input);
```

```
// Process input before passing it to the other function
processInput(input);
copyInput(input);
return 0;
```

Identify two memory-related vulnerabilities in this code snippet. (4pts)

The memory allocated by malloc in processInput is never freed, possibly causing memoryleaks and strcpy and strcat are used instead of their more secure countervarients strinpy and strinat, thus we do not know if the copied strings are null-terminated, possibly causing out-of-bouds access

2 pt.

Status: Klaar met nakijken

1. Buffer overflow. Memory not freed

2 van 4 pt.

Explain the role of dynamic memory allocation in posing security risks. Describe at least two risks. (2pts)

With dynamic memory allocation more freedom is given than with its static variant. But as with programming, more freedom also means that there are more possible mistakes. 2 of these mistakes or risks that occur due to dynamic memory management are overwriting initialized memory and reading unintended memory blocks

1 pt.

Status: Klaar met nakijken

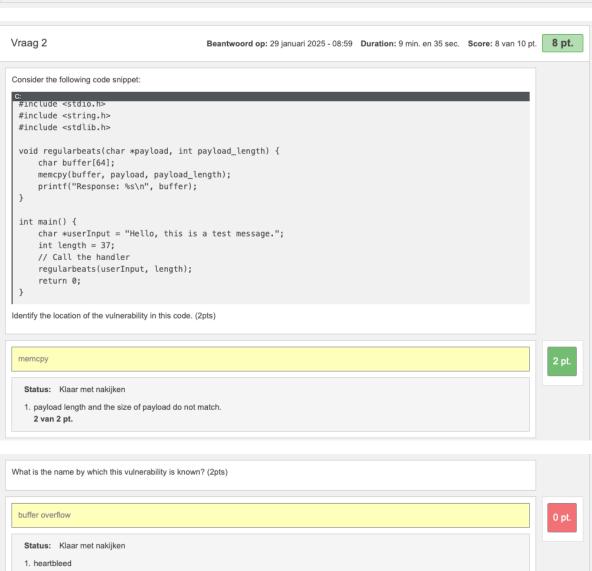
1. Memory leaks; Double free(); Buffer overflow; Memory allocated with malloc is not initialized 1 van 2 pt.

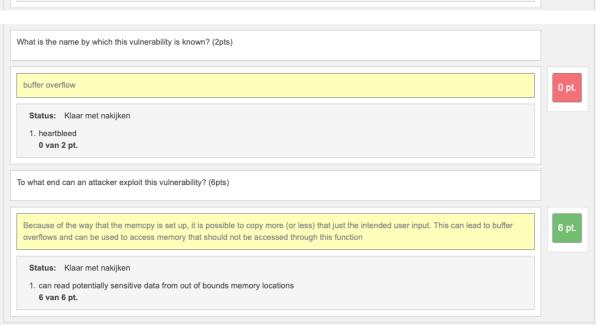
Propose appropriate fixes to the code snippet so it is no longer vulnerable. (4pts)

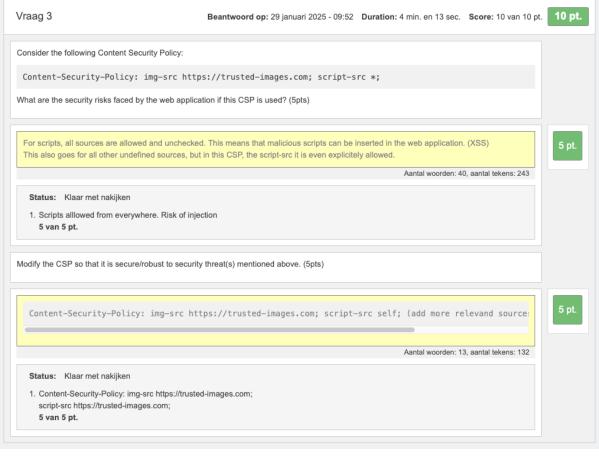
replace every strcpy and strcat by strinpy and strinat and free all memory at the end of main

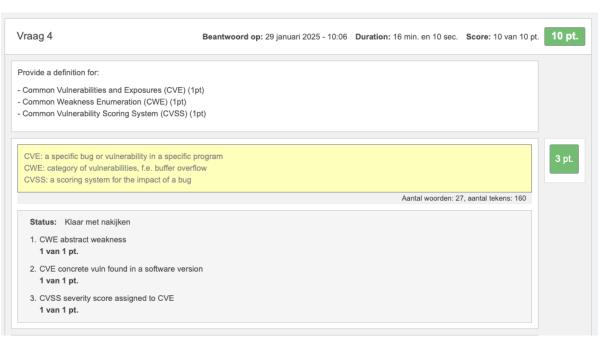
Status: Klaar met nakijken

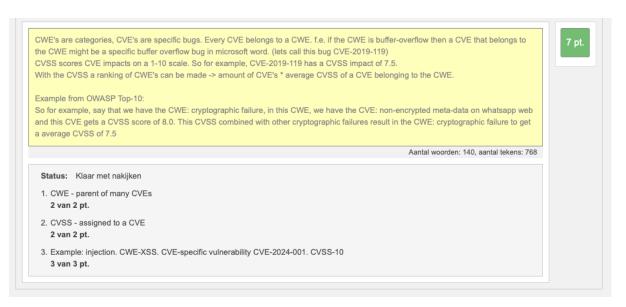
1. Check if combinedInput is not null and that malloc() actually worked. Replace strcpy and strcat with snprintf. Free combinedInput to make sure no memory leak
2 van 4 pt.

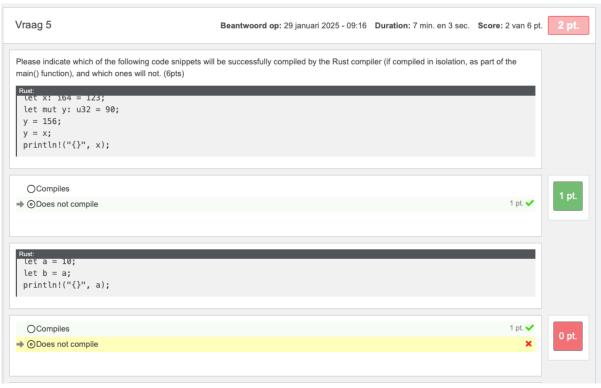
















```
Consider the following incomplete program:
```

```
fn aaa(bbb: BBB, ddd: CCC) -> CCC {
    CCC::D(bbb, Box::new(ddd))
}

fn fff(bbb: BBB, ccc: CCC) -> CCC {
    match ccc {
        CCC::E => return aaa(bbb, CCC::E),
        CCC::D(eee, ddd) => return aaa(eee, fff(bbb, *ddd))
    }
}
```

Try to get an intuition about how it might work before proceeding. Now, you need to make it compilable. In order to do that, you'll need to tackle the following points:

Declare the datatype CCC, based on how the functions deal with values of type CCC. (8pts)

```
struct CCC {
E: int
D: Box
}
```

Aantal woorden: 8. aantal tekens: 39

```
Status: Klaar met nakijken

1. pub enum CCC {
    E,
    D(BBB, Box<CCC>)
  }
2 van 8 pt.
```

Explain which types could be valid for BBB. Also, explain and provide some examples of types that would NOT be valid. (8pts)

It seems that the BBB type is some sort of iterable datatype, we see it used as a first argument with some type of invoke to a datastructure, which sometimes is completely new, suggesting it is data to be used in the structure.

valid examples: integer, string

not valid examples: integer, string

Aantal woorden: 52, aantal tekens: 294

Status: Klaar met nakijken

- i32 works, as does any other predefined numeric type, char, bool, immutable arrays of these, or \&str. More generally, any type implementing the `Copy` trait works
- 0 van 4 pt.
- 2. does not work for references, mutable or not (unless the definition of `CCC` includes a lifetime parameter associated with the `BBB' element of its 'D' case), nor does it work with any type implementing trait 'Drop', as this excludes the implementation of trait 'Copy'. Examples of such types include 'String', 'Vec', and 'Box'.

0 van 4 pt.

Function 'aaa()' makes use of the 'Box' type defined by Rust. If we stop using that type (and its associated functions, e.g., 'new()') in this program, i.e., we simply remove it everywhere it appears, it will stop compiling.

Explain why this happens and why we actually need to use `Box' in this example. (8pts)

If we remove the box-type, the code stops compiling because the structure necessary for CCC to work gets taken away. We need the Box type because we are itiratively using CCC within CCC

Aantal woorden: 33, aantal tekens: 185

Status: Klaar met nakijken

- because the data structure type is recursively defined so Rust will not know how much memory will need to be allocated in advance, which Rust does not allow for stack-allocated data.
 - 2 van 8 pt.

2 pt.

