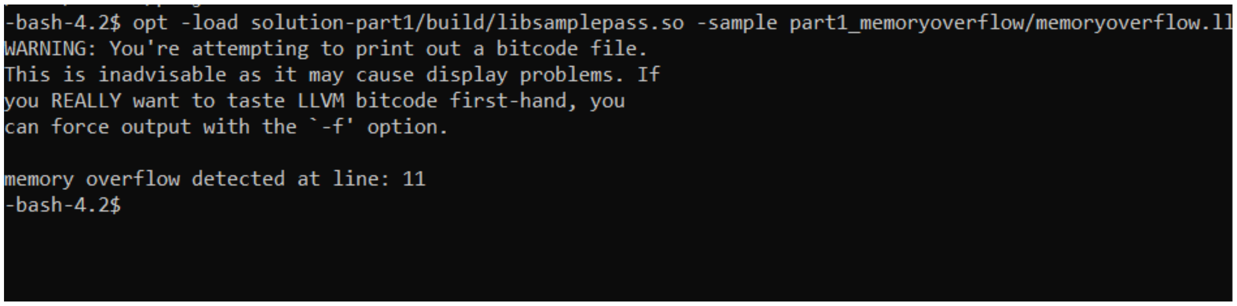
Team 19

For part1 and part2, the source code can be found in samplepass.cpp in directories solution-part1 and solution-part2.

Part 1

cd /home/team19/project4
opt -load solution-part1/build/libsamplepass.so -sample part1_memoryoverflow/memoryoverflow.ll



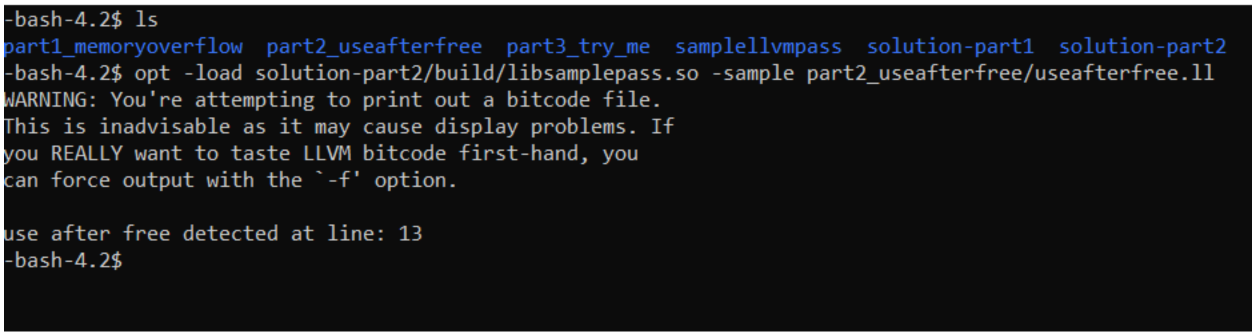
```
-bash-4.2$ opt -load solution-part1/build/libsamplepass.so -sample part1_memoryoverflow/memoryoverflow.ll
WARNING: You're attempting to print out a bitcode file.
This is inadvisable as it may cause display problems. If
you REALLY want to taste LLVM bitcode first-hand, you
can force output with the `-f' option.

memory overflow detected at line: 11
-bash-4.2$
```

For the implementation, the llvm pass loops through all the arguments in memset. The first argument is a pointer for the array, and the size of that array is saved in a local variable when it is encountered. The third argument is the number of bytes to set in memset. When this argument is encountered, the program checks if the number of bytes in the third argument is bigger than the number of bytes in the first argument. If so, there is a memory overflow and the program outputs the line where it was detected.

Part 2

cd /home/team19/project4
opt -load solution-part2/build/libsamplepass.so -sample part2_useafterfree/useafterfree.ll



```
-bash-4.2$ ls
part1_memoryoverflow  part2_useafterfree  part3_try_me  samplellvmpass  solution-part1  solution-part2
-bash-4.2$ opt -load solution-part2/build/libsamplepass.so -sample part2_useafterfree/useafterfree.ll
WARNING: You're attempting to print out a bitcode file.
This is inadvisable as it may cause display problems. If
you REALLY want to taste LLVM bitcode first-hand, you
can force output with the `-f' option.

use after free detected at line: 13
-bash-4.2$
```

For the implementation, the program creates two local vectors for each basic block. The first vector stores strings representing the name of the malloc objects. The second vector stores binary digits representing a state, where 0=object not freed and 1=object freed.

Next, each call instruction is checked in the basic block.

If the call instruction is malloc, then the name of the object created by malloc gets pushed into the first vector. And the number 0 gets pushed into the second vector to represent that the object isn't freed yet.

If the call instruction is free, the name of the object gets searched in the first vector to gets its index. Then the index is used to set the corresponding binary digit to 1 in the second vector to say that it’s freed.

If the call instruction is use, the program checks if the object passed into use was already freed by checking the vector. If the object was already freed, the program outputs the line where there is a memory use after free.