# Finding Memory Corruption Vulnerabilities

## Description

In this assignment, you will be given five different, simple C programs. Each of the programs may contain one or more memory corruption vulnerabilities belonging to the same type (the types are: stack buffer overflow, heap buffer overflow, integer overflow, use after free, use of uninitialized memory). You are required to find at least one vulnerability from each program (the vulnerability type will be specified). You will need to create a document summarizing the vulnerability you find from each program and submit the document to Gradescope.

## What You Need to Do

1. Download the assignment code/examples/documents from <https://github.com/junxzm1990/CS-4440-2022-Spring/tree/main/assignment-3>
2. Go to “assignment-3”, run the Python script called “progassign.py” with your 7-digit “uid” (without the beginning ‘u’ or ‘0’) as an argument:
   1. E.g., $ python progassign.py 1234567
   2. When you are done, you should be able to see an output like the following:

*Please pick the following programs for your assignment:*

*Stack Buffer Overflow/63.c*

*Heap Buffer Overflow/39.c*

*Use After Free/67.c*

*Integer Overflow/16.c*

*Use of Uninitialized Variable/87.c*

1. The output from step 2 indicates the 5 programs assigned to you.
   1. Where to find the programs
      1. Under the folder of “assignment-3”, you can see subfolders for “Stack Buffer Overflow”, “Heap Buffer Overflow”, “Use After Free”, “Integer Overflow”, and “Use of Uninitialized Variable”. In those subfolders, you will be able to see the program assigned to you, such as “63.c” under “Stack Buffer Overflow”
   2. You will need to find at least one vulnerability from the programs assigned to you. The type of vulnerability is specified in the subfolder name. For instance, “Stack Buffer Overflow/63.c” means you can focus on finding “Stack Buffer Overflow” from this program
2. When you are done with finding the vulnerabilities, you will need to prepare a PDF document summarizing the vulnerabilities you found. In the subfolder “Examples” under “assignment-3”, we have prepared examples of how to summarize the vulnerabilities from different types.

## Note

1. You can assume the programs are supposed to run on Linux systems
2. If you see something that you are not familiar with (e.g., wchar\_t), please do a bit of research on Google. The research should be extremely lightweight.
3. Some code regions are controlled by compilation macros like the following:

#ifdef INCLUDEMAIN

int main(int argc, char \* argv[]){

srand( (unsigned)time(NULL) );

#ifndef OMITGOOD

printLine("Calling good()...");

CWE190\_Integer\_Overflow\_\_int64\_t\_max\_preinc\_11\_good();

printLine("Finished good()");

#endif

#ifndef OMITBAD

printLine("Calling bad()...");

CWE190\_Integer\_Overflow\_\_int64\_t\_max\_preinc\_11\_bad();

printLine("Finished bad()");

#endif

return 0;

}

#endif

*You should assume all those macros evaluate to be true. Simply speaking, you can ignore those macros and assume every piece of code will be included for analysis.* That means the above code will be identical to the following:

int main(int argc, char \* argv[]){

srand( (unsigned)time(NULL) );

printLine("Calling good()...");

CWE190\_Integer\_Overflow\_\_int64\_t\_max\_preinc\_11\_good();

printLine("Finished good()");

printLine("Calling bad()...");

CWE190\_Integer\_Overflow\_\_int64\_t\_max\_preinc\_11\_bad();

printLine("Finished bad()");

return 0;

}

1. When you analyze the programs, you may see functions like “printLongLongLine”. All those functions are defined in “assignment-3/Include/io.c”. You may need to also look at those functions when analyzing the programs.
2. You may see code like “#include "std\_testcase.h". All those header files are included in “assignment-3/Include”. Very likely, your analysis won’t rely on those header files.
3. Every student will receive a different set of programs based on her/his UID, so you will need to do it all on your own.

## Grading policy (100 points in total)

* Correctly identifying one memory corruption vulnerability from a program (+20)
* To get the points for a program, please make sure you make the document clear!

Please submit your PDF document to <https://www.gradescope.com/courses/347861/>

## References

<https://cwe.mitre.org/data/definitions/121.html>

<https://cwe.mitre.org/data/definitions/122.html>

<https://cwe.mitre.org/data/definitions/190.html>

<https://cwe.mitre.org/data/definitions/416.html>

<https://cwe.mitre.org/data/definitions/457.html>