SE4950 – Fall 2020
Final Project

The Background

In this project we will explore the use of both static and dynamic analysis tools on a larger program than we have investigated in the past.

The program we will be investigating is tree. Tree is a recursive directory listing program that produces a depth indented listing of files. With no arguments, tree lists the files in the current directory. When directory arguments are given, tree lists all the files and/or directories found in the given directories each in turn. Upon completion of listing all files/directories found, tree returns the total number of files and/or directories listed.

Tree was one of the programs NIST selected for their tests of static analysis tools. The version of source code provided is the one NIST first used, so it has some errors that were present in the original program. I have also seeded a few other errors in the program.

The Assignment

Your job is to use the static analysis tools we have used (cppcheck, scan-build [the Clang analyzer] and infer) as well as the dynamic analysis tools we used (valgrind and gcov/lcov) to find the issues that are present in this program. There are 20 unique issues that I found in the program running these tools. There are some compiler warnings that are issued that you do not need to list in your submission, only the errors found with the tools.

What you should turn into Blackboard is a text/Word/PDF document that lists each of the issues you have found. Use the following format for reporting the errors, with the file name and line number of the source in the first line, the type of issue in the 2nd line, the tools which found the error on the 3rd line, and finally a paragraph explaining the error, as well as what might be done to fix it. (In the case of some of the memory leaks, the tools will help you identify where the memory that leaked was allocated. That is all that is needed for the assignment). Put a blank line between each of the issues so it is easy for me to read.

So for example (this is not the location of an actual error):

Tree.c:47
Uninitialized Value
Clang/Valgrind
This is a description of why this error happened…….

Hints:

The following sections give hints for using one or more of the tools on this analysis.

General

To build the Tree program, the make program is used, along with the Makefile that is in the directory.
To build the program without any tools, when in the src directory, you simply enter the following command:

```
make
```

cppcheck

Since cppcheck runs on one file at a time, you will have to analyze each of the C files separately.

Clang analyzer/scan-build

You can use scan-build with the make system, so to run the analyzer program analyzing all the files that get built, you can simply do the following commands to run the analyzer from the source directory:

```
make clean

scan-build make
```

infer

You can run infer using the make system as well.

```
make clean

infer run -- make
```

There are a few more lines of errors with this one, so there are a couple of ways to explore the bugs. The file `infer-out/bugs.txt` lists all of the bugs that were found. You also can explore each of the errors separately with the following command after you have ran the first one:

```
infer explore
```

valgrind

valgrind can be ran the same way we did in the projects, this time against the tree executable.

You should run the tree program against a large directory. The student user home directory on the distributed VM would be a good example. Also, you may want to redirect the output of the tree program to a file, so that it's output isn't mixed with the valgrind output. To run tree against this directory, with those options , you would issue the following command:

```
valgrind --tool=memcheck --leak-check=yes ./tree -C ~ > tree_prog.out
```

Another hint, any uninitialized variable errors you find with valgrind can be ran with the following flag to find where the uninitialized variable originated:

```
--track-origins=yes
```

So, adding that option the valgrind command will then become:

```
valgrind --tool=memcheck --leak-check=yes –track-origins= yes ./tree -C
~ > tree_prog.out
```

gcov

You can collect the coverage information by turning on coverage in the Makefile. Edit the Makefile and uncomment the last two lines shown below:

```
# Uncomment for Coverage
#CFLAGS+=--coverage
#LDFLAGS+=--coverage
```

There is one of the issues, a memory leak, which does not occur using the default tree command line options. To find it, you may be able to use the coverage analysis to help figure out options to specify for code that hasn't been run.

Grading

The project will be worth 100 points. To get full credit, you should find 14 of the possible 16 unique bugs that are in the code. The tools, if you use all of them correctly, will identify 22 issues. Of those 22, 4 issues are duplicates of issues found by other tools, and 2 are false positives. If you are able to identify the false positives, you will get 5 points of extra credit. If you are able to find the memory leak that only occurs with additional options, you will get 10 points of extra credit.