iJournals: International Journal of Software & Hardware Research in Engineering
ISSN:2347-4890, ijournals.in/ijshre
Volume 12 Issue 12, December 2024

Finding and Analyzing Linux Source Code Errors Using Clang Tools

Authors: [1]Hadeel Tariq Ibrahim; [2]Fatima Hassan Mohamed Ali

[1,2] University of Al-Shatra, College of Education for Women

[1]hadeel.tariq@shu.edu.iq, https://orcid.org/0000-0001-9749-4024

[2]fatimahassan@shu.edu.iq

DOI: 10.26821/IJSHRE.12.12.2024.121204

I. ABSTRACT

In Linux operating system, that based on opened source code, there is a number of faults is roughly constant. While the code size is increasing, these faults will raise, accordingly. While the drivers have improved, the average of fault rate for now is the worst. In addition to that, NULL handling is still a problem [1]. Now there are many great static analyzers, we can find bugs with them automatically .

The main problem that needs solving is: how to detect and find these faults in Linux operating system and analyze them. In terms of the accuracy and range of bug detection, there is a room for improvement. Clang Static Analyzer is Open Source static analyzer [2]. We can control the accuracy and range of bug detection with it. What is more, we are planning to enhance the ability by adding our analysis codes. Applying Clang Static Analyzer to Linux Kernel will reduce review time of codes in making driver or merging patches.

.II. INTRODUCTION

This report will cover the background of how to find and analyze Linux source code errors using Clang tool. It will discuss the problem that need to solve by assessing the accuracy of Clang Static Analyzer in analyzing Linux faults by additional number of checkers. In solution approach section, it shows first the steps of installing Clang tool under Windows environment then discusses the obstacles of this approach and explains the ideal approach by installing Clang under Linux operating system. In the same section, the report browses a number of Linux source codes figures and their executions using Clang tool. In addition to the previous figures, the current report calls Android source codes and compares their execution using Clang with Linux source code (there are another figures with attachment). The report shows the evaluation of Clang tool after using it with Linux and Android source codes. There are some related works to our works and conclusions about Clang tool. Finally, there are some suggestions for future works supported by the steps of adding a new checker.

III. BACKGROUND

As Known, that, Linux is an opened source operating system, this the main reason that make Linux had some errors. We decided to use Clang Analyzer tool to analyze some of these errors. Chou et al, in 2001, studied faults found in Linux by using a static analyzer. An important result of their work that there were 7 times more of certain faults in driver directory than the other directories. These results pushed other researchers to be continued in this field. Finally, they designed a number of tools, one of them was Clang to analyze Linux faults by increasing the number of checkers that used to compile the source code of Linux and C language. [3] The main challenges were, that we are familiar to work with Windows environment, so we tried to install Clang software on Windows where it is essentially working under Linux environment. To overcome this problem, we applied the steps of installing Clang on Windows in November 2013, but every time there was an error occurred. Finally, in December 2013 the designers of Clang announce officially there was a problem of installing Clang under Windows environment within November 2013. At the end, we installed Linux Ubunto 12.4 distribution and use it as our operating system.

IV. PROBLEM STATEMENT

The main problem that needs solving is how to assess the accuracy of bug detection in Linux operating system and analyze them. In terms of the accuracy and range of bug detection, there is a room for improvement [4]. Clang Static Analyzer is Open Source static analyzer. [5] We can control the accuracy and range of bug detection with it. What is more, we are planning to enhance the ability by adding our analysis codes. Applying Clang Static Analyzer [6] to Linux Kernel will reduce review time of codes in making driver or merging patches. After applying Clang on Linux, we applied it on some source codes for Android distributions.

V. SOLUTION APPROACH

At the first step, we tried to install our tool, Clang, on windows 7, 32 bits and 64 bits. We noticed that it is not compatible with 64 bits. So, we applied the following steps many times: Using Visual Studio The following details setting up for and building Clang on Windows using Visual Studio: 1- Get the required tools: Subversion. Source code control program. Get it from: http://subversion.tigris.org/getting.html CMake. This is used for generating Visual Studio solution and project files. Get it from: http://www.cmake.org/cmake/resources/software.html Visual Studio 2008 or 2010 Python. This is needed only if you will be running the tests (which is essential, if you will be developing for clang). Get it from: http://www.python.org/download/ GnuWin32 tools These are also necessary for running the tests. (Note that the grep from MSYS or Cygwin doesn't work with the tests because of embedded double-quotes in the search strings. The GNU grep does work in this case.) Get them from http://getgnuwin32.sourceforge.net/. 2- Checkout LLVM:

3- Checkout Clang:

4- Checkout extra Clang Tools: (optional) cd llvm/tools/clang/tools svn co http://llvm.org/svn/llvm-project/clang-tools-extra/trunk

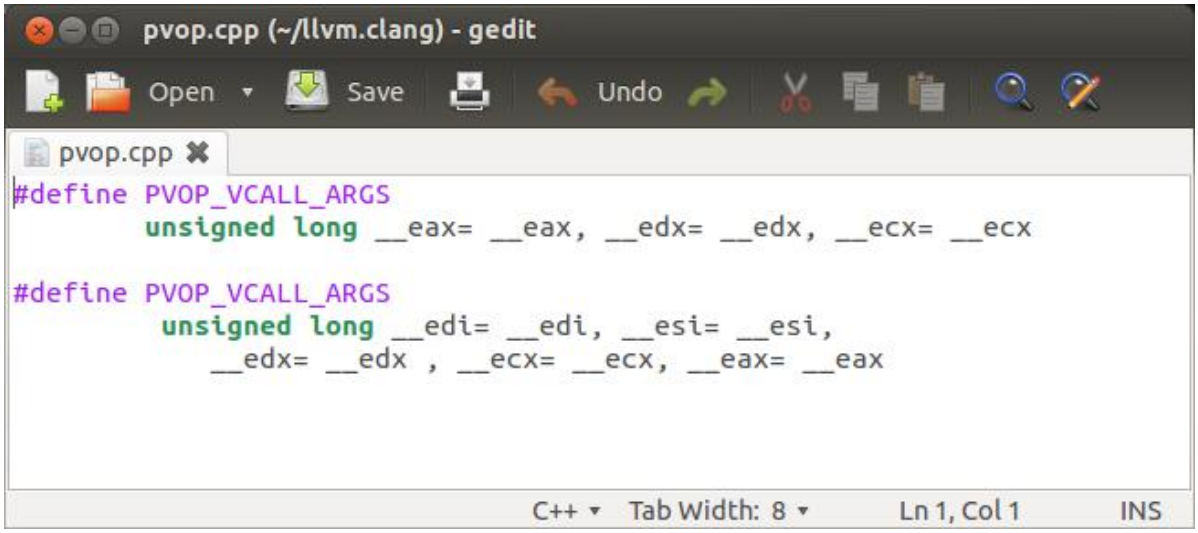

Fig. 1. Memleak.c source code

5- Build Clang: Open LLVM.sln in Visual Studio. Build the "clang" project for just the compiler driver and front end, including tools.

6- Try it out (assuming you added llvm/debug/bin to your path) [6]

After repeating the previous steps many times, for 10 days the LLVM.sln is built... but incomplete. The main problem was that: The following steps work on Mac and Linux only. Note that the clang port is not ready for everyday use yet, so I won't recommend you switching to clang if you are on Windows. As of November 2013,

the trunk of clang is unsuited for building Firefox. [7] And we were working in November 2013 with Firefox!!! So, we installed Linux, Ubunto 12.4 and applied the following steps to install Clang: 1- Get the required tools. (The same requirements for Windows)

2- Checkout LLVM:

Fig. 2. Results of executing Memleak.c using Clang tool

Change directory to where you want the llvm directory placed. svn co http://llvm.org/svn/llvm-project/llvm/trunk llvm 3- Checkout Clang: cd llvm/tools svn co http://llvm.org/svn/llvm-project/cfe/trunk clang cd ../.. 4- Checkout extra Clang Tools: (optional) cd llvm/tools/clang/tools svn co http://llvm.org/svn/llvm-project/clang-tools-extra/trunk extra cd ../../../.. 5- Checkout Compiler-RT: cd llvm/projects svn co http://llvm.org/svn/llvm-project/compiler-rt/trunk compiler-rt cd ../..

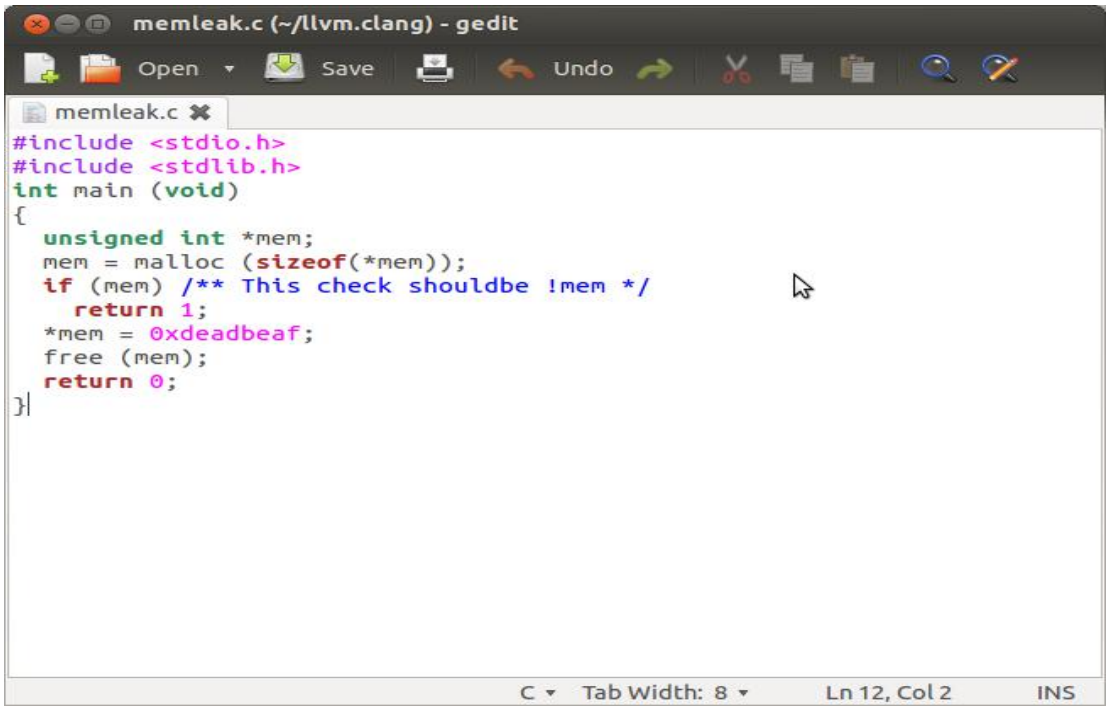

Fig. 3. Pvop.cpp code used the same variables in assignments

As shown in figures 1 and 2, when Clang analyzer has used and the following warning will occurred: memleak.c: 9:3: warning: Dereference of null pointer (loaded from variable 'mem') mem = 0xdeadbeaf; 1 warning generated. That other compilers cannot explore the same error, GCC for example. In Clang, there is an objection when the program used the same variables in assignments, for example the program

Fig. 1. Memleak.c source code

Fig. 2. Results of executing Memleak.c using Clang tool

pvop.cpp:

In figures 3 and 4, there are 5 errors after compilation by

Clang because of using the same definitions.

In this section, we'll discuss if there is difference using

Clang tool between Linux code and Android distributions: [9]

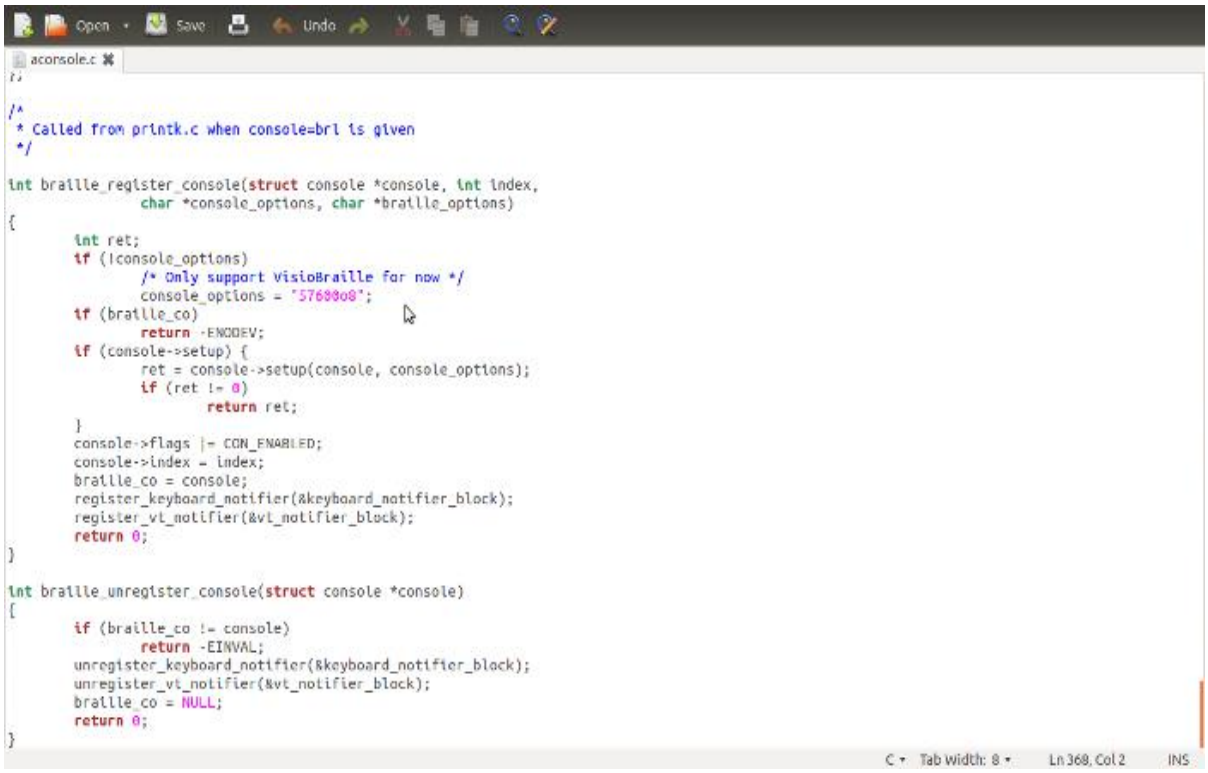

Fig. 4. Results of executing pvop.cpp using Clang tool

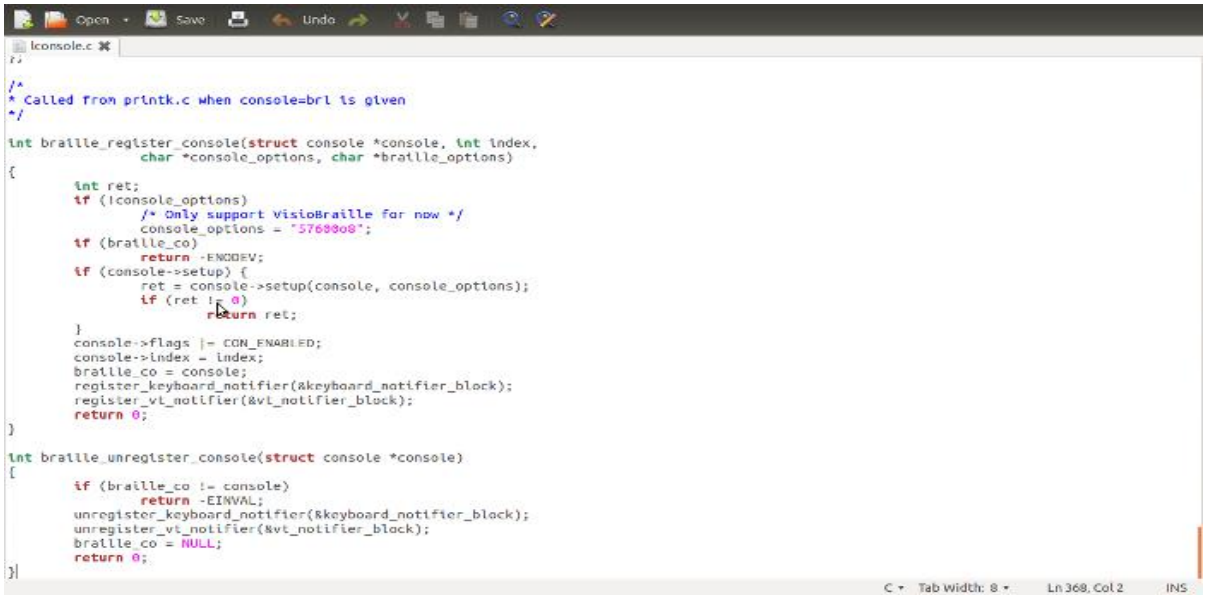

Fig. 5. 1 Console.c code from Linux drivers directory

When applying Clang tool for two codes, Linux and Android in figures 5 ,6 we'll get the same error, as shown in figure 7 Another examples for Linux and Android source code compilation will be attached with the current report.

VI. EVALUATION

The main results from applying Linux and Android source code are: 1- Clang is more accurate than the other compilers like C++

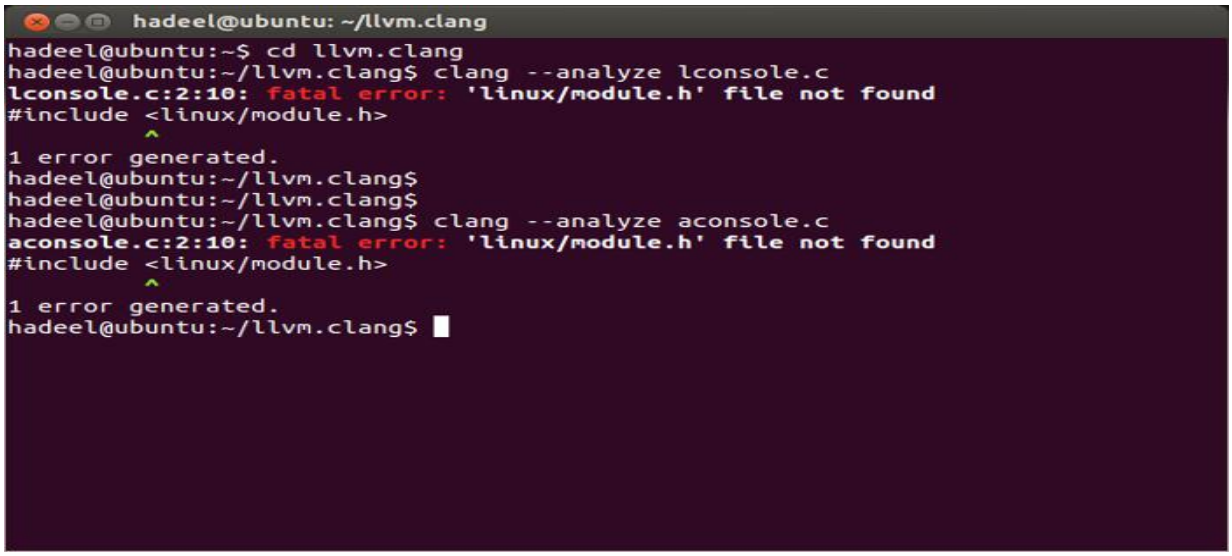

Fig. 6. A console.c code from Android drivers directory

Fig. 7. Results of executing lconsole.c and aconsole.c using Clang tool

because it compiles twice. 2- It is fast in compilation because of using more than 60 checkers. 3- When Clang tools had been used to compile Linux and Android source code, there were the same error occurred in the same files (as seen in attached files).

VII. RELATED WORK

There were some related works related to our project, like: 1- Building FreeBSD with clang/llvm [10] 2- How to Write a Checker in 24 Hours - Clang Static Analyzer. [11] 3- The roadmap towards proper Clang integration into KDevelop. [12]

VIII. CONCLUSIONS

1- Clang tool is not completely compatible to work with Windows environment, so Linux distribution is ideal to work with because Clang code essentially designed for Linux distributions. 2- It is working under Windows with 32 bit system, not 64 bit. 3- Clang is fast and provides more details about errors or warnings. 4- Clang tool is more flexibility and accuracy in treating Linux code errors, than the other compilers.

IX. FUTURE WORKS

For future works, we are recommending adding new checkers to solve some famous errors in Linux source code, like memory. The steps of adding new checker as following: 1- Create a file with checker, e.g. by making a copy of some other. Make sure you leave ClangSACheckers.h include in your checker. 2- Next, you need to add an entry to CMakeLists.txt and Checkers.td. Without that, you will not have your checker built or listed in clang checkers list respectively. 3- You may hit this build error: error: void clang::ento::register... Checker(clang::ento::CheckerManager) should have been declared inside ’clang::ento’ In that case, check that you have ClangSACheckers.h included and that you did not misspelled filename in the two files above. If the compilation is successful, you should see your checker in the clang checker list generated by: clang -cc1 -analyzerchecker-help [13]

X. REFERENCES

REFERENCES

[1] *4 Linux Commands To View Page Faults Statistics*. NIXCRAFT, 2012.

[2] S. S. C. C. J. L. G. M. Nicolas Palix, Gael Thomas, “Faults in linux 10 years later,” *DIKU / INRIA Regal / LIP6*.

[3] *Clang Tools Checkers, https://hal.inria.fr/inria-00509256, Accessed in 2013.*

[4] *Clang Installing Steps, http://clang-analyzer.llvm.org/index.html, Ac- cessed in 2013.*

[5] H. MATSUMOTO, "Applying clang static analyzer to linux kernel," *FUJITSU COMPUTER TECHNOLOGIES LIMITED*, 2012.

[6] *Checking Clang Installation, http://clang.llvm.org, Accessed in 2013.*

[7] *Clang is unsuited for building Firefox in November 2013, http://developer.mozilla.org, Accessed in 2013.*

[8] *Linux Kernel Source Code, https://github.com/hadeelt/kernel, Accessed in 2014.*

[9] *Android source code, http://github.com/hadeel/android, Accessed in 2014.*

[10] *Building FreeBSD with clang/llvm, https://wiki.freebsd.org/BuildingFreeBSDWithClang, Accessed in 2014.*

[11] *How to Write a Checker in 24 Hours, http://techbase.kde.org/Projects/KDevelop4/ClangRoadmap, Accessed in 2013.*

[12] *The roadmap towards proper Clang integration into KDevelop, http://ehsangari.org, Accessed in 2013.*

[13] *http://jirislaby.blogspot.com/2012/03/adding-new-clang-checker.html*.