Eötvös Loránd University

Faculty of Informatics

Dept. of Software Technology and Methodology

Enhancing pattern matching-based static analysis of C-family software projects with project-level knowledge

Supervisor:
Richárd Szalay
Assistant Lecturer

Author:
Benedek Attila Bahrami
Computer Science BSc

Budapest, 2022

This page should be the original Thesis Topic Declaration.

Contents

Chapter 1

Introduction

1.1 Motivation

Static analysis is a method to analyse the source code of software projects without performing a real execution of the application. It is widely used in industry to find bugs and code smells during development, to aid in the prevention of bad code that misbehaves in production. Among various methods, the most important techniques are the ones that are based on pattern matching on a syntactic representation of the software project. Unfortunately, for programming languages in the C family, such as C++, the concept of "separate translation" causes issues for static analysis.

As most static analysers are built upon compilers, and in C++, each compiler only sees the local information in the source file (also known as the Translation Unit) it is to compile or analyse (as opposed to a more project-level knowledge), crucial details might be hidden, which lowers the accuracy of the analysis. Clang-Tidy is a static analysis rule collection that is built upon the LLVM Compiler Infrastructure's C-family compiler, Clang. It performs pattern matching on Clang's "Abstract Syntax Tree" (AST) representation, and generating diagnostics based on which analysis modules – called "checks" – the user turns on. We will address both the LLVM library and AST matchers in later chapters of the thesis.

Let us imagine a checker, that keeps a statistics on how many times a return value of a non-void function is used (in the remainder of the thesis I will refer to the usage of the value as being checked). This property is important, because there exist a great amount ot functions, whose return value should be checked in most situations but remain unchecked in quite a lot.

Some examples:

- POSIX read: returns the number of files read; this return value can also indicate errors with it being -1.
- POSIX scanf: returns the number of items in the argument list successfully filled; also indicates errors with EOF return.
- (cstdio) std::remove: return indicates success or error.
- (algorithm) std::remove: Does not remove. It returns an iterator, and we still need to use erase for all elements after this iterator.
- std::remove_if: Same as remove.
- container specific erase: Returns an iterator to the next element after the removed.
- container specific insert: Returns an iterator to the first of the new elements inserted.

It is important to note, that the nodiscard attribute was introduced for this exact reason, for functions to give warning if their return value is unchecked. A function can obviously exist outside of the translation unit of its declaration. If such analysis with our imagined checker is done separately on each translation unit, it is easy to see how that can affect the outcome. A function might be called 100 times and checked 95 times through. We would want to give warnings for the unchecked 5% of those calls, but if, for example, these are in a separate translation unit, then the analysis would return with 0 checks out of 5 calls. We would not want to give warnings to a function that is unchecked in all 100% of its calls. The detail of the statistics, that it was only unchecked in 5% is lost, unless we use project level knowledge during our analysis. Consider this code example:

```
1   // First translation unit
2 std::vector<int> vector = {7, 9, 10};
3
4 vector.insert(vector.begin(), 6);
5 vector.insert(vector.begin(), 5);
6 vector.insert(vector.begin(), 2);
7 vector.insert(vector.begin(), 1);
```

```
8
9  // Second translation unit
10
11 auto it = vector.insert(vector.begin() + 2, 4);
12 vector.insert(it, 3);
```

With separate analysis, we would get 0% checked in the first TU and 50% in the second one. With project level knowledge, however we would get 12.5%. We used insert without a care towards its return value, since we did not intend to use it for future insertions, except for one occasion.

Unfortunately, the current versions of Clang-Tidy checks only access what is visible to the compiler, which is a local information. Several classes of security issues and bad coding patterns might be diagnosed if the implemented checks would be capable of creating percompilation knowledge, and reusing the full knowledge about the project during diagnosis. The work of the thesis is to enhance Clang-Tidy on the infrastructure level to support multi-pass analyses in a generic manner, by utilizing the ideas similar to that of MapReduce.

This is achieved by allowing individual checks to store check-specific data on a thread-safe location. A subsequent execution of the analysis will be able to do the pattern matching fine-tuned with the data stored in the previous step also available. To prove the usability of the solution, a new safety and security related check, currently not provided by Clang-Tidy, will be developed utilizing the new infrastructure created in this work. In the end, the results of the thesis will allow the international community behind LLVM to develop and make available a wider potential of checks.

Chapter 2

User documentation

Both the changes in the Clang-Tidy infrastructure and our new checker obviously focuses heavily on LLVM's Clang-Tidy. Clang-Tidy is a clang-based C++ "linter" tool. Its purpose is to provide an extensible framework for diagnosing and fixing typical programming errors, like style violations, interface misuse, or bugs that can be deduced via static analysis. Clang-Tidy is modular and provides a convenient interface for writing new checks.

This tool can be found in the LLVM project repository.

2.1 Install guide

2.1.1 System Requirements

Table 2.1 shows the system requirements and supported compilers for building LLVM. The checkers were developed with Ubuntu 20.04 and tested on Ubuntu 18.04, macOS, and WSL Ubuntu 20.04.

Building and using LLVM's Clang-Tidy takes a lot of time on weaker computers. The minimum recommended memory size is 64 GB, the optimal amount is 128 GB or even 256 GB of memory.

Operating System	Processor Architecture	Compiler
Linux	x861	gcc, clang
Linux	amd64	gcc, clang
Linux	arm	gcc, clang
Linux	Mips	gcc, clang
Linux	PowerPC	gcc, clang
Solaris	V9	gcc
FreeBSD	x861	gcc, clang
FreeBSD	amd64	gcc, clang
NetBSD	x861	gcc, clang
NetBSD	amd64	gcc, clang
macOS2	PowerPC	gcc
macOS	x86	gcc, clang
Cygwin	x86	gcc
Windows	x86	Visual Studio
Windows64	x86-64	Visual Studio

Table 2.1: System requirements and supported compilers for building LLVM

2.1.2 Building from source

These commands will compile LLVM from source. The building process with parameters can be found on the README.md of LLVM project Github repository, or the Getting Started page of Clang documentation. These are the commands I used for the compilation.

```
1  # On windows, git clone --config core.autocrlf=false https://
       github.com/llvm/llvm-project.git
2  git clone https://github.com/llvm/llvm-project.git
3  cd llvm-project
4  mkdir Build
5
6  # cmake -S llvm -B build -G <generator> [options]
7  cd Build/
8  cmake \
9    -DCMAKE_EXPORT_COMPILE_COMMANDS=ON -DLLVM_ENABLE_PROJECTS="llvm
       ;clang;clang-tools-extra" \
10   -DLLVM_TARGETS_TO_BUILD="X86" -DLLVM_APPEND_VC_REV=OFF -
       DLLVM_ENABLE_BINDINGS=OFF \
11   -DLLVM_USE_RELATIVE_PATHS_IN_FILES=OFF -DBUILD_SHARED_LIBS=ON -
       DLLVM_USE_LINKER="lld" \
12   -DLLVM_PARALLEL_LINK_JOBS=2 -DCMAKE_BUILD_TYPE=Release -
       DLLVM_ENABLE_DUMP=ON \
```

```
13      -DLLVM_ENABLE_ASSERTIONS=ON \
14      -G Ninja ../llvm
15
16  # cmake --build build [-- [options] <target>] or your build
        system specified above directly.
17  ninja -j12 clang clang-tidy clang-query clang-format llvm-
        symbolizer clang-extdef-mapping
```

2.2 Running Clang-Tidy

The updated infrastructure contains two new flags for running Clang-Tidy, multipass-phase and multipass-dir. Multipass-phase is an enum flag, that has three values, "collect", "compact" and "diagnose" with the latter as default. Multipass-dir needs a path to a directory where the checkers that support the collect feature can dump their collection datas that they are going to compact and use later.

- Collect phase, as the name suggest, will have the checkers collect data on each translation unit and write them into yaml files.
- Compact phase will have the checkers read, use and compact all the data collected into one yaml file.
- Diagnose phase has two types of behaviour:
  - If compact had been run for the checker, it will read and use the compacted data and diagnose accordingly.
  - Otherwise the checker will act as it would have originally, before the feature and the infrastructure was introduced.

Here is an example of a project with two files (of the same name) in different folders, and how I run this from terminal. The original procedure of diagnosis would have been to simply write `clang-tidy -checks='-*,misc-discarded-return-value' -p ./Build a/main.cpp b/main.cpp`, where the "checks" first disables all checkers with -*, then enables our checker, the flag "p" gets the build path and finally we give the paths to our code. Here we would have gotten two separate diagnosis for our two separate files or translation units. Instead of this, we will have do it in different phases:

```
# Pre-collection diagnosis (The phase and directory flags are unneeded in this scenario)
clang-tidy \
-checks='-*,misc-discarded-return-value' --multipass-phase=diagnose \
--multipass-dir='MyCollectionDirectory' -p ./Build \
a/main.cpp b/main.cpp

# Collection phase
clang-tidy \
-checks='-*,misc-discarded-return-value' --multipass-phase=collect \
--multipass-dir='MyCollectionDirectory' -p ./Build \
a/main.cpp b/main.cpp

# Compact phase
clang-tidy \
-checks='-*,misc-discarded-return-value' --multipass-phase=compact \
--multipass-dir='MyCollectionDirectory' -p ./Build \
a/main.cpp b/main.cpp

# Diagnosis with project level knowledge
clang-tidy \
-checks='-*,misc-discarded-return-value' --multipass-phase=diagnose \
--multipass-dir='MyCollectionDirectory' -p ./Build \
a/main.cpp b/main.cpp
```

The first diagnosis will give us what `clang-tidy -checks='-*,misc-discarded-return-value' -p ./Build a/main.cpp b/main.cpp` would have. The second diagnosis after collecting and compacting, howerever might have different results.

Chapter 3

Developer documentation

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis nibh leo, dapibus in elementum nec, aliquet id sem. Suspendisse potenti. Nullam sit amet consectetur nibh. Donec scelerisque varius turpis at tincidunt.

3.1 Theorem-like environments

Definition 1. Mauris tristique sollicitudin ultrices. Etiam tristique quam sit amet metus dictum imperdiet. Nunc id lorem sed nisl pulvinar aliquet vitae quis arcu. Morbi iaculis eleifend porttitor.

Maecenas rutrum eros sem, pharetra interdum nulla porttitor sit amet. In vitae viverra ante. Maecenas sit amet placerat orci, sed tincidunt velit. Vivamus mattis, enim vel suscipit elementum, quam odio venenatis elit, et mollis nulla nunc a risus. Praesent purus magna, tristique sed lacus sit amet, convallis malesuada magna. Phasellus faucibus varius purus, nec tristique enim porta vitae.

Theorem 1. *Nulla finibus ante vel arcu tincidunt, ut consectetur ligula finibus. Mauris mollis lectus sed ipsum bibendum, ac ultrices erat dictum. Suspendisse faucibus euismod lacinia. Etiam vel odio ante.*

Proof. Etiam pulvinar nibh quis massa auctor congue. Pellentesque quis odio vitae sapien molestie vestibulum sit amet et quam. Pellentesque vel dui eget enim hendrerit finibus at sit amet libero. Quisque sollicitudin ultrices enim, nec porta magna imperdiet vitae. Cras condimentum nunc dui. □

Donec dapibus sodales ante, at scelerisque nunc laoreet sit amet. Mauris porttitor tincidunt neque, vel ullamcorper neque pulvinar et. Integer eu lorem euismod, faucibus lectus sed, accumsan felis.

Remark. Nunc ornare mi at augue vulputate, eu venenatis magna mollis. Nunc sed posuere dui, et varius nulla. Sed mollis nibh augue, eget scelerisque eros ornare nec. Praesent porta, metus eget eleifend consequat, eros ligula eleifend ex, a pellentesque mi est vitae urna. Vivamus turpis nunc, iaculis non leo eget, mattis vulputate tellus.

Fusce in aliquet neque, in pretium sem. Donec tincidunt tellus id lectus pretium fringilla. Nunc faucibus, erat pretium tempus tempor, tortor mi fringilla neque, ac congue ex dui vitae mauris. Donec pretium et quam a cursus.

Note. Aliquam vehicula luctus mi a pretium. Nulla quam neque, maximus nec velit in, aliquam mollis tortor. Aliquam erat volutpat. Curabitur vitae laoreet turpis. Integer id diam ligula.

Ut sollicitudin tempus urna et mollis. Aliquam et aliquam turpis, sed fermentum mauris. Nulla eget ex diam. Donec eget tellus pharetra, semper neque eget, rutrum diam.

3.1.1 Equations, formulas

Duis suscipit ipsum nec urna blandit, $2 + 2 = 4$ pellentesque vehicula quam fringilla. Vivamus euismod, lectus sit amet euismod viverra, dolor metus consequat sapien, ut hendrerit nisl nulla id nisi. Nam in leo eu quam sollicitudin semper a quis velit.

$$a^2 + b^2 = c^2$$

Phasellus mollis, elit sed convallis feugiat, dolor quam dapibus nibh, suscipit consectetur lacus risus quis sem. Vivamus scelerisque porta odio, vitae euismod dolor accumsan ut.

In mathematica, identitatem Euleri (equation est scriptor vti etiam notum) sit aequalitatem Equation **??**:

$$e^{i \times \pi} + 1 = 0 \tag{3.1}$$

3.2 Source code samples

Nulla sodales purus id mi consequat, eu venenatis odio pharetra. Cras a arcu quam. Suspendisse augue risus, pulvinar a turpis et, commodo aliquet turpis. Nulla aliquam scelerisque mi eget pharetra. Mauris sed posuere elit, ac lobortis metus. Proin lacinia sit amet diam sed auctor. Nam viverra orci id sapien sollicitudin, a aliquam lacus suscipit. Quisque ac tincidunt leo Code **??** and **??**:

```
#include <stdio>

int main()
{
  int c;
  std::cout << "Hello World!" << std::endl;

  std::cout << "Press any key to exit." << std::endl;
  std::cin >> c;

  return 0;
}
```

Code 3.1: Hello World in C++

```
using System;
namespace HelloWorld
{
  class Hello
  {
    static void Main()
    {
      Console.WriteLine("Hello World!");

      Console.WriteLine("Press any key to exit.");
      Console.ReadKey();
    }
  }
}
```

Code 3.2: Hello World in C#

3.2.1 Algorithms

A general Interval Branch and Bound algorithm is shown in Algorithm **??**. An appropriate selection rule is applied in Step **??**.

Source of example: Acta Cybernetica (this is a hyperlink).

Algorithm 1 A general interval B&B algorithm

Funct IBB(S, f)

1: Set the working list $\mathcal{L}_W := \{S\}$ and the final list $\mathcal{L}_Q := \{\}$
2: **while** ($\mathcal{L}_W \neq \emptyset$) **do**
3: Select an interval X from $\mathcal{L}_W$ ▷ Selection rule
4: Compute $lbf(X)$ ▷ Bounding rule
5: **if** X cannot be eliminated **then** ▷ Elimination rule
6: Divide X into $X^j,\ j = 1, \ldots, p$, subintervals ▷ Division rule
7: **for** $j = 1, \ldots, p$ **do**
8: **if** X^j satisfies the termination criterion **then** ▷ Termination rule
9: Store X^j in $\mathcal{L}_W$
10: **else**
11: Store X^j in $\mathcal{L}_W$
12: **end if**
13: **end for**
14: **end if**
15: **end while**
16: **return** $\mathcal{L}_Q$

Chapter 4

Conclusion

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In eu egestas mauris. Quisque nisl elit, varius in erat eu, dictum commodo lorem. Sed commodo libero et sem laoreet consectetur. Fusce ligula arcu, vestibulum et sodales vel, venenatis at velit. Aliquam erat volutpat. Proin condimentum accumsan velit id hendrerit. Cras egestas arcu quis felis placerat, ut sodales velit malesuada. Maecenas et turpis eu turpis placerat euismod. Maecenas a urna viverra, scelerisque nibh ut, malesuada ex.

Aliquam suscipit dignissim tempor. Praesent tortor libero, feugiat et tellus porttitor, malesuada eleifend felis. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nullam eleifend imperdiet lorem, sit amet imperdiet metus pellentesque vitae. Donec nec ligula urna. Aliquam bibendum tempor diam, sed lacinia eros dapibus id. Donec sed vehicula turpis. Aliquam hendrerit sed nulla vitae convallis. Etiam libero quam, pharetra ac est nec, sodales placerat augue. Praesent eu consequat purus.

Appendix A

Simulation results

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque facilisis in nibh auctor molestie. Donec porta tortor mauris. Cras in lacus in purus ultricies blandit. Proin dolor erat, pulvinar posuere orci ac, eleifend ultrices libero. Donec elementum et elit a ullamcorper. Nunc tincidunt, lorem et consectetur tincidunt, ante sapien scelerisque neque, eu bibendum felis augue non est. Maecenas nibh arcu, ultrices et libero id, egestas tempus mauris. Etiam iaculis dui nec augue venenatis, fermentum posuere justo congue. Nullam sit amet porttitor sem, at porttitor augue. Proin bibendum justo at ornare efficitur. Donec tempor turpis ligula, vitae viverra felis finibus eu. Curabitur sed libero ac urna condimentum gravida. Donec tincidunt neque sit amet neque luctus auctor vel eget tortor. Integer dignissim, urna ut lobortis volutpat, justo nunc convallis diam, sit amet vulputate erat eros eu velit. Mauris porttitor dictum ante, commodo facilisis ex suscipit sed.

Sed egestas dapibus nisl, vitae fringilla justo. Donec eget condimentum lectus, molestie mattis nunc. Nulla ac faucibus dui. Nullam a congue erat. Ut accumsan sed sapien quis porttitor. Ut pellentesque, est ac posuere pulvinar, tortor mauris fermentum nulla, sit amet fringilla sapien sapien quis velit. Integer accumsan placerat lorem, eu aliquam urna consectetur eget. In ligula orci, dignissim sed consequat ac, porta at metus. Phasellus ipsum tellus, molestie ut lacus tempus, rutrum convallis elit. Suspendisse arcu orci, luctus vitae ultricies quis, bibendum sed elit. Vivamus at sem maximus leo placerat gravida semper vel mi. Etiam hendrerit sed massa ut lacinia. Morbi varius libero odio, sit amet auctor nunc interdum sit amet.

Aenean non mauris accumsan, rutrum nisi non, porttitor enim. Maecenas vel tortor ex. Proin vulputate tellus luctus egestas fermentum. In nec lobortis risus,

sit amet tincidunt purus. Nam id turpis venenatis, vehicula nisl sed, ultricies nibh. Suspendisse in libero nec nisi tempor vestibulum. Integer eu dui congue enim venenatis lobortis. Donec sed elementum nunc. Nulla facilisi. Maecenas cursus id lorem et finibus. Sed fermentum molestie erat, nec tempor lorem facilisis cursus. In vel nulla id orci fringilla facilisis. Cras non bibendum odio, ac vestibulum ex. Donec turpis urna, tincidunt ut mi eu, finibus facilisis lorem. Praesent posuere nisl nec dui accumsan, sed interdum odio malesuada.

List of Figures

List of Tables

List of Algorithms

List of Codes