SWRhapsody

Using clang to visit AST

于9月 9, 2018由**SWRhapsody**发布

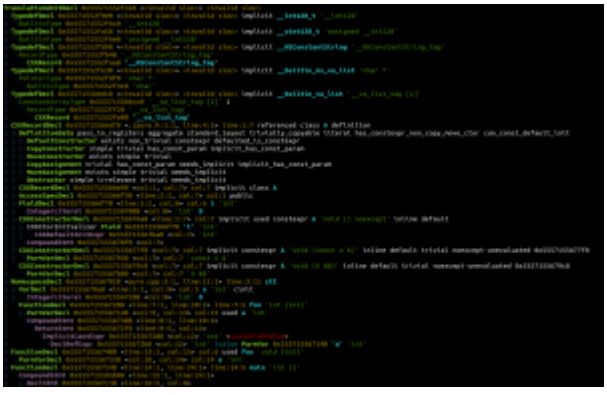
Introduction

If you want write a static analyzer or do something with Abstract syntax tree(AST), instead of implement a AST parser yourself to solve all the corner case, libclang is really a good choice.

Clang is a awesome C,C++ compiler and libclang exports a lot of api which provide you ability to traverse the whole compiled AST easily. However the official tutorial does not introduce every feature in detail, you will need to go through the document and search through google to get better understand about this library. I write this article to record some ways to traverse AST that I once searched.

Code

I supposed you have already read the official tutorial [1], let's start with writing a simple tool traverses the AST and prints it just like command `clang -Xclang -ast-dump -fsyntax-only XXX.cc` , dump the AST to console.



SWRhapsody

As the official tutorial said, one common entry point for writing a clang based tool like a Clang Plugin or a standalone tool based on LibTooling is `FrontendAction`. `FrontAction` allows you to run some user specific actions as part of the compilation. As our goal is to traverse the AST, we will need a `ASTConsumer`.

```
class clangConsumer : public clang::ASTConsumer
{
public:
  explicit clangConsumer(clang::ASTContext *Context)
      : Visitor(Context) {}

  virtual bool HandleTopLevelDecl(clang::DeclGroupRef d) override

  {
    typedef clang::DeclGroupRef::iterator iter;
    for (iter b = d.begin(), e = d.end(); b != e; ++b)
    {
      Visitor.TraverseDecl(*b);
    }
    return true; // keep going
  }

private:
  clangVisitor Visitor;
};

class clangAction : public clang::ASTFrontendAction
{
public:
  clangAction(){}

  virtual std::unique_ptr<clang::ASTConsumer> CreateASTConsumer(
      clang::CompilerInstance &Compiler, llvm::StringRef InFile)
  {
    return std::unique_ptr<clang::ASTConsumer>(
        new clangConsumer(&Compiler.getASTContext()));
  }

private:
};
```

`ASTConsumer` is responsible for how we traverse the AST, it provides many entry points for accessing AST. `HandleTranslationUnit` is a good one. But I will use `HandleTopLevelDecl`, who is called by the parser to process every top-level Decl*, to do the job. We also need a visitor to visit each decl.

```
class clangVisitor
    : public clang::RecursiveASTVisitor<clangVisitor>
{
public:
  explicit clangVisitor(clang::ASTContext *Context):Context(Context){};

  // bool VisitStmt(clang::Stmt *s);

  bool VisitDecl(clang::Decl *s){
clang::FullSourceLoc FullLocation = Context->getFullLoc(decl->getLocStart());
```

SWRhapsody

```
15      {
16        std::cout << "Found Decl at "
17                  << name << " "
18                  << FullLocation.getSpellingLineNumber() << ":"
19                  << FullLocation.getSpellingColumnNumber() << "\n";
20        decl->dumpColor();
21      }
22    }
23    return true;
24  }
25
26    bool TraverseDecl(clang::Decl *D){
27    // std::cout << "TraverseDecl\n";
28    // RecursiveASTVisitor<clangVisitor>::TraverseDecl(D);
29    //the default method will visit the ast tree recursively
30    VisitDecl(D);
31    return true;
32  }
33
34  private:
35    clang::ASTContext *Context;
36  };
```

In the visitor we override the original method `TraverseDecl` which will visit `Decl*` recursively, as we already use `HandleTopLevelDecl` to feed visitor all top `Decl` , we can use our own code to dump them. We can also choose not to override `TraverseDecl` and declare method VisitXXDecl like `VisitCXXRecordDecl` to visit a specific kind of `Decl` to achieve this goal.

> */// These tasks are done by three groups of methods, respectively:*
> */// 1. TraverseDecl(Decl *x) does task #1. It is the entry point*
> */// for traversing an AST rooted at x. This method simply*
> */// dispatches (i.e. forwards) to TraverseFoo(Foo *x) where Foo*
> */// is the dynamic type of *x, which calls WalkUpFromFoo(x) and*
> */// then recursively visits the child nodes of x.*
> */// TraverseStmt(Stmt *x) and TraverseType(QualType x) work*
> */// similarly.*
> */// 2. WalkUpFromFoo(Foo *x) does task #2. It does not try to visit*
> */// any child node of x. Instead, it first calls WalkUpFromBar(x)*
> */// where Bar is the direct parent class of Foo (unless Foo has*
> */// no parent), and then calls VisitFoo(x) (see the next list item).*
> */// 3. VisitFoo(Foo *x) does task #3.*

The last step is to run the tool on the code. The example showed in the official tutorial need to read

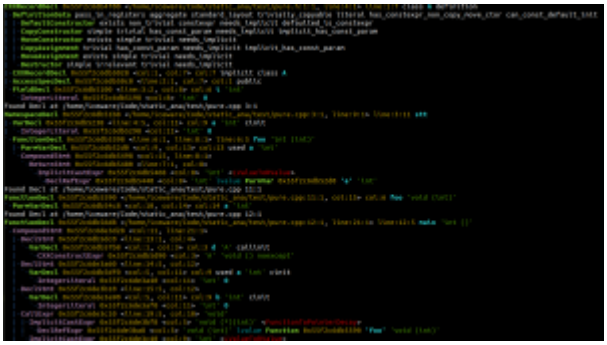
SWRhapsody

```
1   std::string err_message;
2   auto database = clang::tooling::CompilationDatabase::loadFromDirectory(database_path.string()
3
4   clang::tooling::ClangTool Tool(*database, {file_path.string()});
5   assert(database == nullptr);
6   Tool.run(newFactory<clangAction>(this).get());
```

```
1 [
2     {
3         "command": "/usr/bin/clang++  -c /path/test/pure.cpp",
4         "directory": "/path/test",
5         "file": "/path/pure.cpp"
6     }
7 ]
```

`database_path` is where you place the compilation database json file, `file_path` is the path of file you want to be processed “/path/pure.cpp” in this case.

Finally we only need to build and run it, I remember it took me a long time to find how to build it correctly with cmake, I use [3] to help me find where libclang is.



Reference

[1] Official tutorial http://clang.llvm.org/docs/IntroductionToTheClangAST.html

[2] https://shaharmike.com/cpp/libclang/

[3] https://github.com/Iceware/old_code/tree/master/static_ana/cmake

分类: EXERCISE

SWRhapsody

发表评论

名称 *

电子邮件 *

网站

在想些什么?

发表评论

近期文章

- 携程Apollo YAML 反序列化
- CVE-2020-5410
- CodeQL部分源码简读
- 服务器与网关的不一致
- CodeQL 部分使用记录

SWRhapsody

文章归档

2020年8月

2020年6月

2020年5月

2020年3月

2020年1月

2019年12月

2019年11月

2019年8月

2019年7月

2019年5月

2019年4月

2019年1月

2018年11月

2018年10月

2018年9月

2018年4月

2018年3月

2018年2月

2018年1月

分类目录

Article Collection

Cheat Sheet

SWRhapsody

Exploit

HackTheBox

Penetration Test

Uncategorized

相关文章

EXERCISE

CodeQL部分源码简读

Introduction CodeQL 也用了不少时间了，从最初的不会 阅读更多…

CHEAT SHEET

CodeQL 部分使用记录

前言 CodeQL 是一个代码分析引擎，主要原理是通过对代码进行构建并 阅读更多…

EXERCISE

Java 反编译工具

SWRhapsody

ABOUT

Hestia |由ThemeIsle开发