信息安全综合实习
课程报告

姓　　名：乔晓晓

专　　业：信息安全

学　　号：20191001815

日　　期：2022.10.13

目录

1. 问题简述

本题目要求学生开发一个对源代码中的函数调用参数进行分析的插件，具体地，要求学生开发一个 checker，完成如下一些基本要求：

- 定义自己的 checker（完善 checker 的名称、说明）；
- 实现对源代码中函数调用中的参数进行检查的代码，并应实现支持配置文件，由配置文件决定检查哪些函数（根据名称进行过滤），以及检查哪些参数及该参数允许或者禁止的范围；配置文件格式可由学生自定义；考虑分析的局限性，有一些参数可能无法在静态分析中确定其取值，对于此类情况，需要输出信息提示用户；
- 完成对 checker 的测试。

2. 环境搭建

参考已给的 Clang 编译手册，将我们编译的 Clang 代替 Ubuntu 系统默认的版本 10 的 Clang，使用命令查看 Clang 版本，搭建成功后，版本应为 15。

```
student-20191001815@93f70c55224c:~$ clang --version
clang version 15.0.2 (https://github.com/llvm/llvm-project.git 77ff99c10bee11a202dbe5fd1a08d8394d7828af)
Target: x86_64-unknown-linux-gnu
Thread model: posix
InstalledDir: /home/student-20191001815/llvm-project/build/bin
```

2.1 clang 版本展示

3. 实验流程

3.1 定义自己的 checker

① 根据服务器中给出的样例，创建自己的目录及文件(包含 txt、export 和一个 cpp)，并将内容进行复制；

② 按照自己的需求对样例代码进行改动，按照已给出的教程进行修改；

③ 编译并安装过 Clang，然后在构建目录（build）中执行：ninja；可以获得构建好的插件：CheckerIndexCall.so

```
CMakeFiles
CheckerDependencyHandlingAnalyzerPlugin.so
CheckerIndexCall.so
CheckerOptionHandlingAnalyzerPlugin.so
CodeGen
```

3.1 插件编译成功

④ 以插件的方式运行检查器(需要自己编写测试 cpp 代码、插件代码)。

3.2　实现参数检查代码

3.2.1 需要解决的问题

综合题目要求，我们需要解决以下问题：

① 配置文件格式的设计；

② 检查器函数编写规则；

③ 如何获取到关于函数的名称及参数信息；

④ 自定义检查器的整体逻辑。

3.2.2 问题解决

1. 配置文件及过滤格式的设计

根据题目要求，在涉及到需要检查参数的函数调用时，我们需根据配置文件中对其设定的范围等规则进行过滤，若不符合规则，则需要输出信息提示用户。但针对参数设定的规则并不仅仅是一个范围，还有可能是某个具体的值。例如，针对 malloc 函数的参数进行检查，分析其是否为 0（不允许这种情况出现）。所以，在代码实现中设计结构体时，我们不能只定义范围，还需考虑到不能为 0 的情况。

因此，在编写代码时，为规则设计一个结构体，如下图所示。结构体的内容包括：函数的名称、需要检查的参数位置、检查的类型、范围下限、范围上限。函数的名称即需要进行检查的函数名称；需要检查的参数位置定义为从 0 开始；检查的类型包括两类：限定某个具体的值(1)和需要范围限定两种类型(2)。在代码实现过程中需要对类型进行判断，进而实现进一步的判断。

```
typedef struct
{
    char name[0x20];
    int argloc;

    int type;
    int L;
    int R;
} FR;
```

3.2 规则结构体

在配置文件格式及调用方面，采用 csv 格式的文件，在过滤之前，将配置文件按照结构体的格式进行引入用于之后的判断。

2. 检查器函数编写规则

检查器函数编写不同于常见的 c 语言或其他语言代码编写，他所使用的变量类型、编写规则等与其他不同，主要包括四个部分：申明一个 checker 类，实现 checker 的回调函数、编写 checker 的 bugreport、注册 checker。

在 CSA 中的 Checker 类要想订阅程序点，只需要继承类模板 Checker<...>，并将...替换为要订阅的程序点。然后，实现由所订阅程序点规定的回调函数。常用的程序点有 PreCall、PostCall 等；

我们应该在由所订阅程序点规定的回调函数中实现错误诊断代码逻辑。这些回调函数既决定了我们可以利用哪些信息用于错误诊断代码逻辑的实现，也决定了这些错误诊断代码何时被调用。

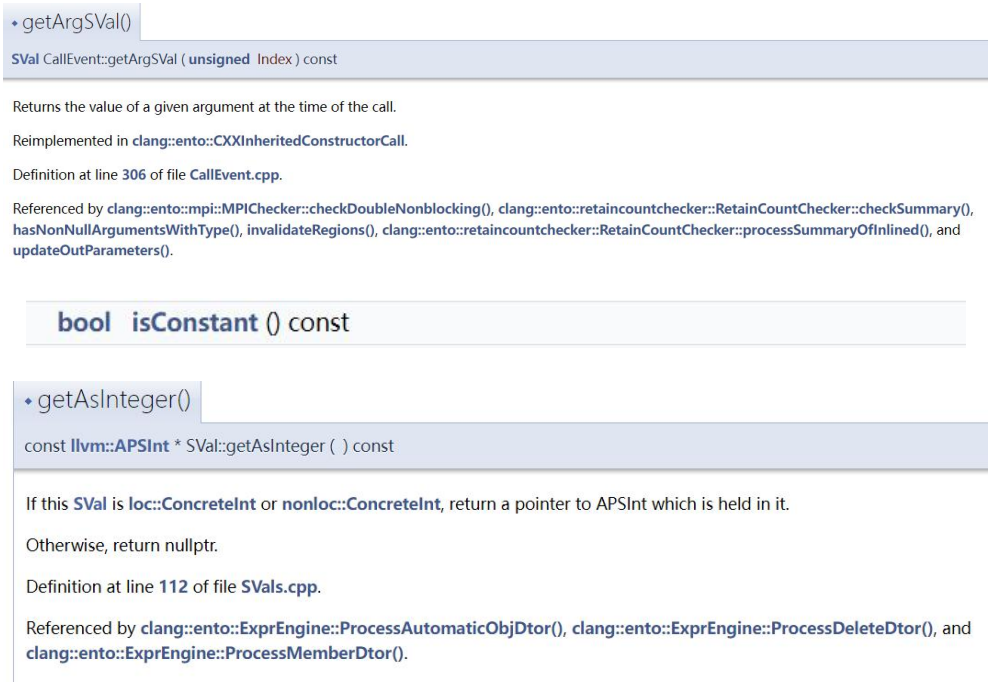
最后还需加上检查器的注册函数，其中需要注意名称的一致。

3. 函数名称及参数信息获取

定义好规则后，我们需要考虑，如何在函数调用中获取到关于函数名称、参数类型等信息，从而才能进行判断。

回调函数中的 CallEvent 结构体在回调函数中有效地包含了所有 analyzer 核心为我们收集的函数调用事件的数据，以及这个结构体包含了有关被调函数的所有信息和参数的值。

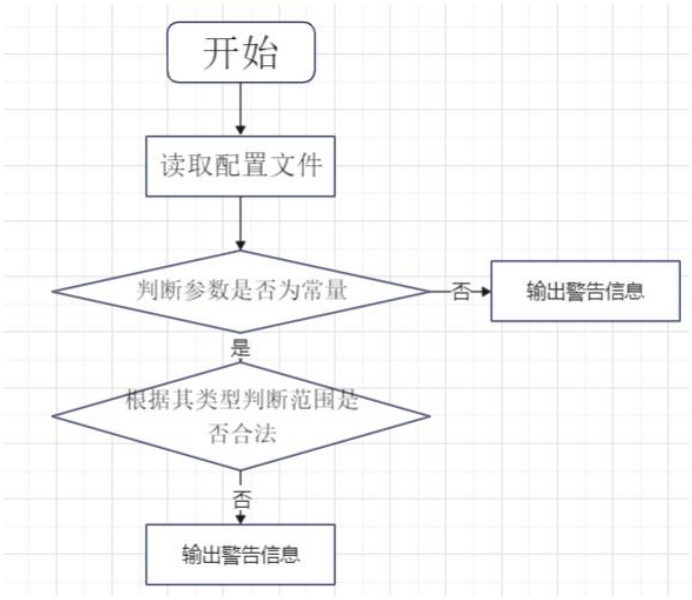
观察给出样例及 clang 官网给出的函数，会发现函数名称的获取包含在 IdentifierInfo 结构体中，它一般用来保存 CallEvent 结构体的被调函数的返回值；而关于某个具体位置的函数信息包含在 getArgSVal 函数中，它可以获取关于某个具体位置的参数信息，返回类型为 SVal，在具体应用中需要进行进一步的转换。关于函数的详细信息如下图所示：



3.3 具体使用函数

4. 自定义检查器的整体逻辑

整体代码流程如下：



3.3　　错误诊断代码流程图

4. 实验效果

编写对应测试 cpp，以插件的方式运行检查器，并将结果输出到文件中。

4.1 针对于不能为 0 的函数

```
student-20191001815@93f70c55224c:~$ clang -cc1 "-internal-isystem" "/usr/lib/gcc/x86_64-linux-gnu/9/../../../../incl
ude/c++/9" "-internal-isystem" "/usr/lib/gcc/x86_64-linux-gnu/9/../../../../include/x86_64-linux-gnu/c++/9" "-intern
al-isystem" "/usr/lib/gcc/x86_64-linux-gnu/9/../../../../include/c++/9/backward" "-internal-isystem" "/12T/home/stud
ent-20191001815/llvm-project/build/lib/clang/15.0.2/include" "-internal-isystem" "/usr/local/include" "-internal-isy
stem" "/usr/lib/gcc/x86_64-linux-gnu/9/../../../../x86_64-linux-gnu/include" "-internal-externc-isystem" "/usr/inclu
de/x86_64-linux-gnu" "-internal-externc-isystem" "/include" "-internal-externc-isystem" "/usr/include" "-fcolor-diag
nostics"  -load ~/llvm-project/build/lib/CheckerIndexCall.so -analyze -analyzer-checker=plugin.CheckerIndexCall test
.cpp -analyzer-output plist -o ~/result
test.cpp:11:16: warning: Arg 0 of function 'malloc' cannot be zero [plugin.CheckerIndexCall]
        data = (char*)malloc(0);
                      ^~~~~~~~
1 warning generated.
```

4.1.1 结果输出

结果输入到 result 文件后，使用 cat 命令可以看到 result 文件中，包含测试结果的警告信息。

```
student-20191001815@93f70c55224c:~$ cat result
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
 <key>clang_version</key>
<string>clang version 15.0.2 (https://github.com/llvm/llvm-project.git 77ff99c10bee11a202dbe5fd1a08d8394d7828af)</st
ring>
 <key>diagnostics</key>
 <array>
  <dict>
   <key>path</key>
   <array>
    <dict>
     <key>kind</key><string>control</string>
     <key>edges</key>
      <array>
```

```
      <key>depth</key><integer>0</integer>
      <key>extended_message</key>
      <string>Arg 0 of function &amp;apos;malloc&amp;apos; cannot be zero</string>
      <key>message</key>
      <string>Arg 0 of function &amp;apos;malloc&amp;apos; cannot be zero</string>
     </dict>
   </array>
   <key>description</key><string>Arg 0 of function &amp;apos;malloc&amp;apos; cannot be zero</string>
```

4.1.2 结果文件展示

4.2 针对于具体范围内的函数

```
student-20191001815@93f70c55224c:~$  clang -cc1 "-internal-isystem" "/usr/lib/gcc/x86_64-linux-gnu/9/../../../../inc
lude/c++/9" "-internal-isystem" "/usr/lib/gcc/x86_64-linux-gnu/9/../../../../include/x86_64-linux-gnu/c++/9" "-inter
nal-isystem" "/usr/lib/gcc/x86_64-linux-gnu/9/../../../../include/c++/9/backward" "-internal-isystem" "/12T/home/stu
dent-20191001815/llvm-project/build/lib/clang/15.0.2/include" "-internal-isystem" "/usr/local/include" "-internal-is
ystem" "/usr/lib/gcc/x86_64-linux-gnu/9/../../../../x86_64-linux-gnu/include" "-internal-externc-isystem" "/usr/incl
ude/x86_64-linux-gnu" "-internal-externc-isystem" "/include" "-internal-externc-isystem" "/usr/include" "-fcolor-dia
gnostics"  -load ~/llvm-project/build/lib/CheckerIndexCall.so -analyze -analyzer-checker=plugin.CheckerIndexCall ran
ge.cpp
range.cpp:15:5: warning: Arg 0 of function 'func' should be in range [0,100] [plugin.CheckerIndexCall]
    func(101);
    ^~~~~~~~~
1 warning generated.
```

4.2.1 结果输出

4.3 针对于不是常量的函数参数(此处为自定义函数)

```
student-20191001815@93f70c55224c:~$ clang -cc1 "-internal-isystem" "/usr/lib/gcc/x86_64-linux-gnu/9/../../../../incl
ude/c++/9" "-internal-isystem" "/usr/lib/gcc/x86_64-linux-gnu/9/../../../../include/x86_64-linux-gnu/c++/9" "-intern
al-isystem" "/usr/lib/gcc/x86_64-linux-gnu/9/../../../../include/c++/9/backward" "-internal-isystem" "/12T/home/stud
ent-20191001815/llvm-project/build/lib/clang/15.0.2/include" "-internal-isystem" "/usr/local/include" "-internal-isy
stem" "/usr/lib/gcc/x86_64-linux-gnu/9/../../../../x86_64-linux-gnu/include" "-internal-externc-isystem" "/usr/inclu
de/x86_64-linux-gnu" "-internal-externc-isystem" "/include" "-internal-externc-isystem" "/usr/include" "-fcolor-diag
nostics"  -load ~/llvm-project/build/lib/CheckerIndexCall.so -analyze -analyzer-checker=plugin.CheckerIndexCall cons
tant_test.cpp
constant_test.cpp:16:5: warning: (Ignored) Arg 0 of function 'func' is non-constant [plugin.CheckerIndexCall]
    func(m);
    ^~~~~~~
1 warning generated.
```

4.3.1 结果输出

5. 遇到的问题及解决方法

1. 直接以插件的方式运行检查器，头文件无法通过；

```
student-20191001815@93f70c55224c:~$ clang -cc1 -load ~/llvm-project/build/lib/CheckerIndexCall.so -analyze -analy
zer-checker=plugin.CheckerIndexCall test.cpp
test.cpp:3:10: fatal error: 'stdio.h' file not found
#include "stdio.h"
         ^~~~~~~~~
1 error generated.
```

   解决：将头文件地址包含在命令中即可通过运行。

2. 在引入配置文件时需引入配置文件的绝对地址，写相对地址无法通过。

6. 代码附件

```
#include "clang/StaticAnalyzer/Core/BugReporter/BugType.h"
#include "clang/StaticAnalyzer/Core/Checker.h"
#include "clang/StaticAnalyzer/Core/PathSensitive/CallEvent.h"
#include "clang/StaticAnalyzer/Core/PathSensitive/CheckerContext.h"
#include "clang/StaticAnalyzer/Frontend/CheckerRegistry.h"
#include "clang/StaticAnalyzer/Core/Checker.h"
#include "clang/StaticAnalyzer/Core/CheckerManager.h"

using namespace clang;
using namespace ento;

int loaded = 0;

typedef struct
```

```
{
    char name[0x20];     //函数名称
    int argloc;          //待检测的参数位置
    int type;         //类别
    int L;           //下限
    int R;            //上限
} FR;

typedef std::vector<FR> FRList;
FRList frl;

using namespace clang;
using namespace ento;

namespace {
class CheckerIndexCall : public Checker<check::PostCall> {
  mutable std::unique_ptr<BugType> BT;
  void initRules(CheckerContext &C) const;
public:
  void checkPostCall(const CallEvent &Call, CheckerContext &C) const;

};
} // end anonymous namespace


void CheckerIndexCall::checkPostCall(const CallEvent &Call,
                                     CheckerContext &C) const {

  if (!Call.isGlobalCFunction()) {
    return;
  }
  initRules(C);
  const Expr *Callee = Call.getOriginExpr();
  const IdentifierInfo *II = Call.getCalleeIdentifier();
  const CallExpr *CE = dyn_cast<CallExpr>(Callee);

  if (!II) {
    return;
  }
  if (!CE) {
    return;
  }
  int i;
```

```
  for(i=0; i<frl.size(); i++)
  {

        FR si = frl.at(i);

        if (II->isStr(si.name))
        {
            //std::cerr<<Call.getArgSVal(si.argloc).isConstant()<<std::endl;
            if(!Call.getArgSVal(si.argloc).isConstant())
            {
                std::string rep = "(Ignored) Arg " + std::to_string(si.argloc) + " of
function \'" + si.name + "\' is non-constant.";
                ExplodedNode *N = C.generateNonFatalErrorNode(C.getState());
                if (!N)
                    return;
                if(!BT)
                    BT.reset(new BugType(this, rep, rep));
                auto report =
                    std::make_unique<PathSensitiveBugReport>(*BT,
BT->getDescription(), N);
                report->addRange(Callee->getSourceRange());
                C.emitReport(std::move(report));
                BT.release();
                return;
            }

            const llvm::APSInt* arg1 = Call.getArgSVal(si.argloc).getAsInteger();

            if(si.type == 1 && arg1->getExtValue() == 0)
            {
                std::string rep = "Arg " + std::to_string(si.argloc) + " of function \'"
+ si.name + "\' cannot be zero.";
                ExplodedNode *N = C.generateNonFatalErrorNode(C.getState());
                if (!N)
                    return;
                if(!BT)
                    BT.reset(new BugType(this, rep, rep));
                //BT = BugType(this, rep, rep);
                auto report =
                    std::make_unique<PathSensitiveBugReport>(*BT,
BT->getDescription(), N);
                report->addRange(Callee->getSourceRange());
                C.emitReport(std::move(report));
                BT.release();
```

```
                return;
            }

            if(si.type == 2  && (arg1->getExtValue() < si.L || arg1->getExtValue() >
si.R))
            {
                std::string rep = "Arg " + std::to_string(si.argloc) + " of function \'"
+ si.name + "\' should be in range [" + std::to_string(si.L) + "," + std::to_string(si.R)
+ "].";
                ExplodedNode *N = C.generateNonFatalErrorNode(C.getState());
                if (!N)
                    return;
                if(!BT)
                    BT.reset(new BugType(this, rep, rep));
                //BT = BugType(this, rep, rep);
                auto report =
                    std::make_unique<PathSensitiveBugReport>(*BT,
BT->getDescription(), N);
                report->addRange(Callee->getSourceRange());
                C.emitReport(std::move(report));
                BT.release();
                return;
            }
        }
    }
}

void CheckerIndexCall::initRules(CheckerContext &C) const {
    if(loaded)
        return;
    FILE * fp = fopen("/home/student-20191001815/config.csv","r");
    FR tmp;
    while(fscanf(fp,"%19s,%d,%d,%d,%d",tmp.name,&tmp.argloc,&tmp.type,&tmp.L,&tmp.R)!=0
)
        frl.push_back(tmp);
    loaded = 1;
}


// Register plugin!
extern "C" void clang_registerCheckers(CheckerRegistry &registry) {
  registry.addChecker<CheckerIndexCall>(
      "plugin.CheckerIndexCall", "CheckerIndexCall",
      "");
```

```
}


extern "C" const char clang_analyzerAPIVersionString[] =
    CLANG_ANALYZER_API_VERSION_STRING;
```