

**课 程 设 计 报 告**

**基于源代码的软件同源性分析  
与漏洞检测系统**

课 程 软件安全课程设计

院 系 网络空间安全学院

专业班级 信息安全1803

学 号 U201814735

姓 名 张帅

指导教师 刘铭

2020年 10 月 18 日

软件安全课程设计-功能完成情况

题目：基于源代码的软件同源性分析与漏洞检测系统

班级：信息安全1803班 学号：U201814735 姓名：张帅

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **序号** | **内容及分值** | **得分标准** | **本人完成情况：**  **完成项，填Ｙ；**  **其他优点可备注；** | **分值** | **得分**  **（教师填写）** |
| **必选** | **共50分** | | | |  |
| R1 | 提供系统界面 | 图形化界面GUI：4+；  控制台界面console：3+；设计合理、美观 | Y | 5 |  |
| R2 | 利用字符串匹配进行同源性检测 | 能比对，并输出相似度10；准确度3；能对比展示样本不同行内容2； | Y | 15 |  |
| R3 | 利用控制流程图CFG进行源代码同源性检测 | 生成cfg并保存5；对比得出相似度5；准确度5 | Y | 15 |  |
| R4 | 栈缓冲区检测 | 给出可疑代码行数与列数 | Y | 5 |  |
| R5 | 格式化字符串漏洞检测 | 给出可疑代码行数与列数 | Y | 5 |  |
| R6 | 提供样本库 | 提供漏洞检测与同源性检测样本库，样本数量不少于10个，每个代码行数不少于100行；每种漏洞至少一个。 | Y | 5 |  |
| 2选1 | **共5分** | | | |  |
| A1 | 跨语言同源性检测验证 | 每额外增加一种语言2分（4个样例）。5分封顶 |  | 5 |  |
| A2 | 支持分布式任务调度 | 选定文件夹同时比对3+，结果准确、快速2+。 | Y | 5 |  |
| 6选4 | **共40分**  初步定位可能发生的位置，每个5分 （共20分）  对他人样本能完全、正确确定漏洞位置，每个5分，（共20分） | | | |  |
| B1 | 堆缓冲区检测 | 给出可疑代码行数与列数 | Y | 10 |  |
| B2 | 整数宽度溢出检测 | 给出可疑代码行数与列数 | Y | 10 |  |
| B3 | 整数运算溢出检测 | 给出可疑代码行数与列数 | Y | 10 |  |
| B4 | 整数符号溢出检测 | 给出可疑代码行数与列数 |  | 10 |  |
| B5 | 空指针引用 | 给出可疑代码行数与列数 | Y | 10 |  |
| B6 | 竞争性条件 | 给出可疑代码行数与列数 |  | 10 |  |
| 2选1 | **共5分** | | | |  |
| C1 | 同源性检测样本库 | 50个样本(每个100行以上)每10个为相似的一组，1分，最高5分 | Y | 5 |  |
| C2 | 漏洞检测样本库 | R4+R5 +B1-6选4，每组7个样本，共6组，每组0.5分，共3分；混合类型样本18个，1-2分。 |  | 5 |  |
| **合计** | **满分100** | | | |  |

课程总评成绩

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 平时成绩 | 完成功能情况 | 课程报告 | | 总评成绩 |
| （10%） | （50%） | 格式规范（20%） | 内容完整 （20%） | 100% |
|  |  |  |  |  |

**目 录**

[1 课程设计任务书 7](#_Toc53929809)

[1.1 课程设计目的 7](#_Toc53929810)

[1.2 课程设计要求 7](#_Toc53929811)

[1.3 系统环境 8](#_Toc53929812)

[1.4 实验过程记录 8](#_Toc53929813)

[2 绪言 9](#_Toc53929814)

[2.1字符串匹配同源性检测 9](#_Toc53929815)

[2.2控制流程图CFG源代码同源性检测 9](#_Toc53929816)

[2.3漏洞检测 9](#_Toc53929817)

[3 系统方案设计 11](#_Toc53929818)

[3.1 系统方案设计 11](#_Toc53929819)

[3.2 字符串匹配同源性检测 11](#_Toc53929820)

[3.3 CFG匹配同源性检测 11](#_Toc53929821)

[3.4 栈缓冲区溢出 12](#_Toc53929822)

[3.5 格式化字符串漏洞检测 12](#_Toc53929823)

[3.6 分布式任务调度 12](#_Toc53929824)

[3.7 堆缓冲区检测 13](#_Toc53929825)

[3.8 整数宽度溢出检测 13](#_Toc53929826)

[3.9 整数运算溢出检测 13](#_Toc53929827)

[3.10 空指针引用 13](#_Toc53929828)

[4 系统实现 14](#_Toc53929829)

[4.1 字符串匹配同源性检测 14](#_Toc53929830)

[4.1.1 检测流程 14](#_Toc53929831)

[4.1.2 关键函数说明 14](#_Toc53929832)

[4.1.3相似性计算 16](#_Toc53929833)

[4.2 CFG匹配同源性检测 16](#_Toc53929834)

[4.2.1 检测流程 16](#_Toc53929835)

[4.2.2 关键函数说明 17](#_Toc53929836)

[4.3 栈缓冲区溢出 20](#_Toc53929837)

[4.3.1 检测流程 20](#_Toc53929838)

[4.4 堆缓冲区溢出 21](#_Toc53929839)

[4.4.1 检测流程 21](#_Toc53929840)

[4.5 整数宽度溢出 21](#_Toc53929841)

[4.5.1 检测流程 21](#_Toc53929842)

[4.6 格式化字符串漏洞检测 21](#_Toc53929843)

[4.6.1 检测流程 21](#_Toc53929844)

[4.7 空指针引用 21](#_Toc53929845)

[4.7.1 检测流程 21](#_Toc53929846)

[4.8 整数运算溢出检测 22](#_Toc53929847)

[4.8.1 检测流程 22](#_Toc53929848)

[4.9 分布式任务调度 22](#_Toc53929849)

[4.9.1 运行流程 22](#_Toc53929850)

[4.9.2 代码实现 22](#_Toc53929851)

[4.10 UI模块 22](#_Toc53929852)

[4.10.1 制作流程 22](#_Toc53929853)

[5 系统测试 23](#_Toc53929854)

[5.1 字符串匹配同源性检测 23](#_Toc53929855)

[5.1.1 样例测试 23](#_Toc53929856)

[5.1.2 问题解决 23](#_Toc53929857)

[5.2 CFG匹配同源性检测 24](#_Toc53929858)

[5.2.1 样例测试 24](#_Toc53929859)

[5.2.2 问题解决 24](#_Toc53929860)

[5.3 栈溢出检测 24](#_Toc53929861)

[5.3.1 样例测试 24](#_Toc53929862)

[5.4 字符串匹配同源性检测 25](#_Toc53929863)

[5.4.1 样例测试 25](#_Toc53929864)

[5.5 分布式任务调度检测 26](#_Toc53929865)

[5.5.1样例测试 26](#_Toc53929866)

[5.6 堆溢出检测 26](#_Toc53929867)

[5.6.1 样例测试 26](#_Toc53929868)

[5.7 整数宽度溢出检测 27](#_Toc53929869)

[5.7.1 样例测试 27](#_Toc53929870)

[5.8 整数运算溢出检测 27](#_Toc53929871)

[5.8.1 样例测试 27](#_Toc53929872)

[5.9 空指针引用检测 28](#_Toc53929873)

[5.9.1 样例测试 28](#_Toc53929874)

[5.10 UI界面 28](#_Toc53929875)

[5.10.1 界面说明 28](#_Toc53929876)

[6 总结与展望 30](#_Toc53929877)

[6.1 总结 30](#_Toc53929878)

[6.2 展望与致谢 30](#_Toc53929879)

[7 参考文献 31](#_Toc53929880)

# 1 课程设计任务书

## 1.1 课程设计目的

本次课程设计为了巩固上学期在软件安全课程上所学的安全知识，包括堆栈溢出、整数溢出等等，同时考察了一些课外的新事物，例如字符串匹配与CFG控制流程图的同源性检测。根本目的是让同学们加深对软件安全本门学科的理解和深入的实践。

以下是本次课设的具体目的：

* 对代码的同源性检测，分为字符串匹配和CFG的检测，在于考察学生自我解决问题的能力，包括自己查找文献，根据文献提炼思想，再将思想转化为实际代码的能力。
* 对于漏洞的检测，包括几种不同类型的漏洞种类，这要求学生对于每种漏洞的特征做一个总结，通过静态分析的方式将代码中可能隐藏的漏洞发掘出来。

## 1.2 课程设计要求

本次课程设计要求设计与完成从界面、算法到系统优化等各个环节内容，形成完整的软件系统。在任务中，必做部分为R1-R6,是所有同学必须完成的部分，是本课程设计考核合格所需要的基本构成，选做部分根据自己的擅长做出最好的成果。

具体任务如下：

|  |  |  |
| --- | --- | --- |
| 序号 | 任务（必须完成的） | 要求 |
| R1 | 提供系统界面 | 所有功能要有图形界面展示,形成完整的软件系统.可以使用VS/QT/Python等工具实现。 |
| R2 | 利用字符串匹配进行同源性检测 | 通过代码有效字符串对比匹配，分析样本之间的拷贝比率 |
| R3 | 利用控制流程图CFG进行源代码同源性检测 | 通过提取代码的调用关系图，检测样本之间各个函数调用关系图是否相似，得出相似的概率 |
| R4 | 栈缓冲区检测 | 根据栈缓冲区原理分析分配的栈数据区是否存在溢出的问题，给出可疑代码行数与列数。 |
| R5 | 格式化字符串漏洞检测 | 根据格式化字符串漏洞原理分析使用的格式化函数是否存在溢出的问题，给出可疑代码行数与列数。 |
| R6 | 提供样本库 | 提供漏洞检测与同源性检测样本库，样本数量不少于10个，每个代码行数不少于100行；每种漏洞至少一个。 |
| 以下为2选1: | | |
| A1 | 跨语言同源性检测验证 | 在软件版权保护中，有时候需要检测是否参考了有版权的代码，换用一种语言实现同样的功能，本功能可以通过CFG检测实现，但需要给出4个以上不用语言的同源性分析样本。 |
| A2 | 支持分布式任务调度 | 需要设计一个主控，多个进程/主机并发检测。 |
| 以下为6选4: | | |
| B1 | 堆缓冲区检测 | 根据堆缓冲区原理分析分配的数据区是否存在溢出的问题，给出可疑代码行数与列数。 |
| B2 | 整数宽度溢出检测 | 根据整数宽度溢出原理分析分配的数据是否存在溢出的问题，给出可疑代码行数与列数。 |
| B3 | 整数运算溢出检测 | 根据整数运算溢出原理分析分配的数据是否存在溢出的问题，给出可疑代码行数与列数。 |
| B4 | 整数符号溢出检测 | 根据整数符号溢出原理分析分配的数据是否存在溢出的问题，给出可疑代码行数与列数。 |
| B5 | 空指针引用 | 根据课堂学习其它溢出原理是否存在空指针引用的问题，给出可疑代码行数与列数。 |
| B6 | 竞争性条件 | 给出竞争性条件存在的代码位置。 |
| 以下2选1: | | |
| C1 | 同源性检测样本库 | 样本数大于等于50个，每个代码行数不少于100行,包含1-100行相同代码。 |
| C2 | 漏洞检测样本库 | 样本数大于等于50个，每个代码行数不少于100行；每种漏洞至少一个。 |

## 1.3 系统环境

1. 操作系统：Windows 10
2. 运行环境：Python 3.7.3, ply, NetworkX，Tkinter
3. 编程语言：Python 3.7
4. 备注：ply是python中的Lex词法分析器组件，NetworkX是python中的图算法组件，Tkinter是python自带的UI工具

## 1.4 实验过程记录

09/07/2020 – 09/13/2020：1) 完成字符串的匹配的同源性检测

09/14/2020 – 09/19/2020：1）完成CFG图检测

2）完成分布式任务调度

09/20/2020 – 09/27/2020：1）完成漏洞检测部分

2）完善了整个代码体系

3）完成同源性检测样本库

09/28/2020 – 10/5/2020： 1）完成UI的实现

2）代码进行完善和再调试

# 2 绪言

## 2.1字符串匹配同源性检测

软件的同源性分析是计算机编程语言研究的重要方面之一，主要有三类方向：基于文本的同源性比对、基于单词（token）的同源性比对、基于源代码语法结构的比对。

基于文本的同源性鉴别的目的,是查找出文本上的代码抄袭。常见文本抄袭手段有文本的完全拷贝、增加删除行、顺序无关代码块调换等。典型的基于文本相似性的鉴别技术有：基于子串匹配的方法,其基本思路是从文档中选取一些字符串,被称为“指纹”,然后把指纹映射到hash表中,一个指纹对应一个数字,最后统计hash表中相同的指纹数目或者比率,作为文本相似度依据。这种只对源代码的部分特征进行统计，忽略源代码的结构信息,导致错误率较高；其次，参数化匹配方法（Brenda S.Baker）,解决了变量名替换问题。此外,还有典型的词频统计法用于同源性鉴别。源代码抄袭的过程一般都是整块复制,然后加以改动,比如替换变量名,在不影响程序功能的情况下打乱语句顺序,更改函数名或者函数位置等等。由于基于文本层次的鉴别完全忽略了源代码的语法含义,因而对于上述大部分改动情况都没有理想的效果。

## 2.2控制流程图CFG源代码同源性检测

现在已经有一些从源代码语法结构的层次进行代码克隆检测和同源性比对的研究成果,比如利用欧式向量空间记录语法树信息进行比对（DECKARD）；用Xml记录语法树信息来比对。另外还有在线比对，如Moss，JPlag等（JPlag在一定程度上考虑了语法）,所有的在线比对系统需要提交源码,源代码的保密工作就可能无法保证。Baxter等人提出了一种利用源代码抽象语法树进行代码克隆检测的算法，算法在整个比对过程中都保持了树形结构,需要进行多次树的遍历。

与第一个实验中的单词符号token比对类似，先通过预处理，可以去除程序中的注释、空白符号等，只保留语言有效的单词token，对于获得的单词token，按语言的词法规则将单词分类（参考编译程序的编写），比对中不关心单词的名称，而只注重单词的类型比较（当然，名称对于值传递有作用）。然后按该语言的语法规则，对于各种语法成分，用语法树形式存放并比较。这里我们因为课设时间有限，进行一定的简化，我们只保留函数调用的语法成分，在分析时只构造合适的存储结构（比如树形结构），保留函数调用的相互关系（包括函数名、参数类型等），形成简化的CFG。将原始程序对应的CFG图与修改后的代码对应的CFG图进行比较，计算相似程度。

## 2.3漏洞检测

早期的漏洞挖掘分析一般使用程序分析、模糊测试和符号执行等确定性的程序推理测试方法。随着技术的发展，业界开始尝试多种技术组合的方式以提高分析能力，如KLEE、Mayhem等系统采用基于优化的符号执行技术进行漏洞挖掘，RETracer、CREDAL等系统采用基于内核转储和程序分析定位漏洞点。当前研究方向已开始转向利用人工智能辅助漏洞挖掘分析[3-4]，研究人员试图从程序执行历史中总结出漏洞的特征和发生原因，但因缺少先验知识的指导，无法挖掘分析深层漏洞。

基于模式的可利用性分析是当前漏洞利用验证研究的主要方法。AEG（automatic exploit generation）工具针对源码进行漏洞抽象，利用符号执行和约束求解构造exploit，判定漏洞是否可用。Mayhem和CRAX工具分别在PIN和QEMU的支持下获取漏洞相关信息，优化符号执行效率，较好地对识别的漏洞进行exploit自动化构造。Grieco等利用符号执行和模式匹配的方法可对缓冲区溢出的可利用性进行判定，但是，由于漏洞模式众多，模式提取是一项十分具有挑战的工作，限制了该方法的适用面。Miller等[基于二进制分析平台Bitblaze开发了一套异常判定工具，通过对程序执行路径进行追踪获取和异常相关的信息，并检查异常指令是否能被程序输入影响，提高了异常可利用性判定的精确度，但是它利用虚拟机仿真技术，开销很大，同时，能被输入控制的指令不一定在异常指令附近，并且可能会相距很远，依然会产生很多漏报的可用异常。此外，学者们研制出了AEG、CRAX、Driller等半自动工具，但这些工具严重依赖于对漏洞模式的抽象，应用范围有限。这些方法或工具依赖于人工经验难以规模化，或缺少全路径的数据流分析，无法确定数据与用户输入的关系，较易造成误报。

# 3 系统方案设计

## 3.1 系统方案设计

系统总共分为利用字符串匹配进行同源性检测、利用控制流程图CFG进行源代码同源性检测、栈缓冲区检测、格式化字符串漏洞检测、支持分布式任务调度、堆缓冲区检测、整数宽度溢出检测、整数符号溢出检测、空指针引用，以及UI模块。

## 3.2 字符串匹配同源性检测

输入：输入数据来源是文件内容

功能：对接受的文件文本内容进行预处理，再利用Lex词法分析器对预处理后的文件进行词法分析，然后利用LCS算法进行最大字串的匹配，计算两个文件内容的相似性

输出：输出的数据是代码相似度的百分比。

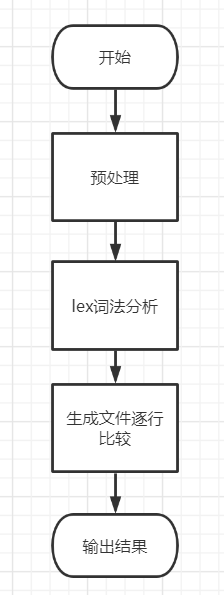


图3.2.1 字符串匹配同源性检测流程

## 3.3 CFG匹配同源性检测

输入：输入数据来源是文件内容

功能：先对输入的文本文件进行预处理，再利用lex词法分析器对预处理的文件进行此法分析，然后对代码中的关键流程部分建立成图，再利用图的同构算法对图的相似度进行计算。（注释：图编辑距离是一种类似于字符串编辑距离的概念，本课设内的图为有向图）

输出：输出的数据是代码相似度的百分比。

流程图如下图3.3.1

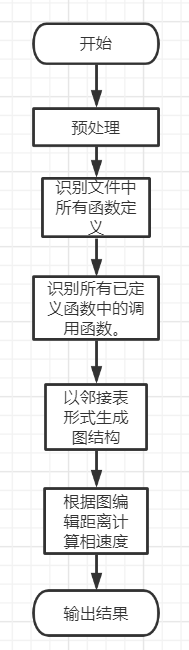


图3.3.1 CFG图匹配同源性检测流程

## 3.4 栈缓冲区溢出

输入：输入数据来源是文件内容

功能：通过对文件扫描，检测其中是否有关于栈缓冲区溢出的可疑函数，再分析函数溢出的可能

输出：输出可能出错的行数和可疑函数的元组，形式如此：

警告:在以下文件行发现使用可疑函数 "函数名" :

行号

## 3.5 格式化字符串漏洞检测

输入：输入的数据来源是文件内容

功能：扫描文件内容，先扫描到可疑函数，再分析可疑函数中的输出格式串个数与后面的参数个数是否匹配。若不匹配，则返回可疑结果。

输出：输出内容是可疑函数的行数和函数名，形式如此：

存在参数不匹配的可疑行数有:

4

## 3.6 分布式任务调度

输入：输入数据来源是文件内容

功能：建立一个内存池，在内存池中建立多个进程，分别独立处理各个子任务，最后返回各个子任务的输出结果。

输出：输出内容是各个子任务的返回结果，在此是字符串同源性检测结果和CFG同源性检测结果

## 3.7 堆缓冲区检测

输入：输入数据来源是文件内容

功能：通过对文件扫描，检测其中是否有关于栈缓冲区溢出的可疑函数，再分析函数溢出的可能

输出：输出可能出错的行数和可疑函数的元组，形式如此：

警告:在以下文件行发现使用可疑函数 "函数名" :

行号

## 3.8 整数宽度溢出检测

输入：输入数据来源是文件内容

功能：扫描一遍代码，将代码中所有变量及变量的类型存储起来，再扫描代码中的是否存在长整型数向短整型数赋值的情况

输出：输出整型转换与溢出的代码位置

## 3.9 整数运算溢出检测

输入：输入数据来源是文件内容

功能：首先分析代码中是否存在加法和乘法，再分析这种转换导致溢出的可能

输出：输出整数运算溢出可能的行数

## 3.10 空指针引用

输入：输入数据来源是文件内容

功能：先扫描文件中的指针，在指针使用之前先对其进行安全检查，确保指针有效，检查指针在使用时是否还是null

输出：输出空指针引用的可疑行数

# 4 系统实现

## 4.1 字符串匹配同源性检测

### 4.1.1 检测流程

（1）预处理。这部分处理不影响程序的功能部分，对于注释、头文件导入语句、宏定义等进行处理。对于不影响语义的字符，预处理过程中设置为空，如宏定义、注释、TAB、回车、空行等。对于类型重定义情况，将源代码中重定义类型还原为原始类型，以降低同源性检测的误检率。

（2）词法分析。进行源代码的词法分析。通过读入源代码信息，依据程序语言词法规则实现每行代码的参数（Token）化 表示，例如返回与输入标志符、关键字等相匹配的符号序列。此项工作可以通过词法分析器pylex完成。

（3）建立代码行数据结构。对于参数化的源代码文件，以行为单位建立源文件和目标文件的行描述数据结构，记录行信息，并依此形成列表（二维）。

（4）相似度计算。根据列表摘要信息进行代码比对评估，计算相似模块总计行数，以相似的行数与总行数的比值计算相似度。

### 4.1.2 关键函数说明

（1）preprocess函数

函数声明：def preprocess(file)

函数功能：将程序文件中的注释、宏定义、头文件去掉，将其中的类型定义装入字典中，再将文件中所有出现的替换类型换成原本的定义

函数实现：

1. **def** preprocess(file):
2. # before all functions, if typedef,put orign&new into a dic.
3. dic = {}
4. newTypelist = []
5. line = file.readline()
6. **while** line:
7. **if** '#' **in** line **or** '/' **in** line:
8. **pass**
9. **elif** 'typedef' **in** line:
10. originType = line.split(' ')[1]
11. newType = line.split(' ')[2]
12. newType = newType.strip(';')
13. dic[newType] = originType
14. newTypelist.append(newType)
15. **elif** '{' **in** line:
16. **break**
17. line = file.readline()
18. **for** key, value **in** dic.items():
19. **if** dic.\_\_contains\_\_(value):
20. dic[key] = dic[value]
22. # Restore source code redefinition type to original type
23. tempfile = ""
24. file.seek(0)
25. line2 = file.readline()
26. **while** line2:
27. **if** '#' **in** line2 **or** '//' **in** line2 **or** 'typedef' **in** line2:
28. """or /\*"""
29. **pass**
30. **else**:
31. place, res = typeexist(newTypelist, line2)
32. **if** res:
33. line2 = line2.replace(place, dic[place])
34. tempfile += line2
35. line2 = file.readline()
36. **return** tempfile

（2）mylex函数：

函数声明：def mylex(data)

函数功能：将文本文件中的所有程序内容通过词法分析器转换为单词Token

函数实现：此函数参照pylex官方文档和民间翻译，没有以上两篇文档我便无法完成此部分，链接分别为：<http://www.dabeaz.com/ply/ply.htm>

https://www.cnblogs.com/P\_Chou/p/python-lex-yacc.html#4\_1

保留字定义与token定义就不在此赘述，可见文档外的实际代码，其他函数定义部分我在以下附上：

1. # A regular expression rule with some action code
2. **def** t\_NUMBER(t):
3. r'\d+'
4. t.value = int(t.value)
5. **return** t
7. # Define a rule so we can track line numbers
8. **def** t\_newline(t):
9. r'\n+'
10. t.lexer.lineno += len(t.value)
12. # A string containing ignored characters (spaces and tabs)
13. t\_ignore = ' \t'
15. # Error handling rule 尝试不输出错误字符而是直接跳过
16. **def** t\_error(t):
17. #print("Illegal character '%s'" % t.value[0])
18. t.lexer.skip(1)
20. **def** t\_ID(t):
21. r'[a-zA-Z\_][a-zA-Z\_0-9]\*'
22. t.type = reserved.get(t.value, 'ID')
23. **return** t

26. # Build the lexer
27. lexer = lex.lex()
28. # input data
29. lexer.input(data)
30. # tokennize
31. pre\_line = 1
32. result = ''
33. # 从下面可以看出来，preprocess的时候其实并没去掉空格，是t\_ignore和下面8行函数实现的，因为空格没被存在token里。
34. **while** True:
35. tok = lexer.token()
36. **if** **not** tok:
37. **break**
38. **if** pre\_line != tok.lineno:
39. result += '\n'
40. pre\_line = tok.lineno
41. result += tok.type
42. **return** result

### 4.1.3相似性计算

函数声明：def calculate(lexfile1, lexfile2)

函数功能：逐行比对计算代码行相似度。

函数实现：依据公式

similar＝linesimilar／linewaitingcodes，

计算出相似度similar

## 4.2 CFG匹配同源性检测

### 4.2.1 检测流程

1. 预处理。这部分处理不影响程序的功能部分，对于注释、头文件导入语句、宏定义等进行处理。对于不影响语义的字符，预处理过程中设置为空，如宏定义、注释、TAB、回车、空行等。对于类型重定义情况，将源代码中重定义类型还原为原始类型，以降低同源性检测的误检率。
2. VertexJudge函数，遍历文件，识别出所有已定义函数，存在列表a中
3. Init\_connections函数，再次遍历文件，识别出所有已定义函数内的调用函数，存在列表b、c、d…中，并生成字典:{函数:[调用函数组成的列表]}，此为邻接表结构。
4. 调用Networkx库，计算图编辑距离。
5. 使用Calculate函数计算相似度。

### 4.2.2 关键函数说明

* 1. VertexJudge函数：

函数声明：def VertexJudge(tmpfile1)

函数功能：根据C语言定义函数的特性找到所有定义函数。

函数实现：见如下代码：

1. **def** VertexJudge(tempfile):
2. vertexs = []
3. **for** line **in** tempfile:
4. **if** '(' **in** line:
5. **if** ('char' **in** line) **or** ('int' **in** line) **or** ('long' **in** line) **or** ('short' **in** line) **or** ('float' **in** line) **or** (
6. 'double' **in** line) **or** ('void' **in** line) **or** ('signed' **in** line) **or** ('bool' **in** line):
7. **if** ';' **not** **in** line:
8. name = get\_function\_name(line)
9. vertexs.append(name)
10. **return** vertexs
    1. Get\_function\_name函数

函数声明：def get\_function\_name(line)

函数功能：获取所有已声明函数的函数名。

函数实现：见如下代码：

1. # 获取函数名称：比如说bool judge\_func(char str[max]), 则list1=['bool' 'judge\_func(char','str[max])'
2. # 挺巧妙地避开了正则表达式。
3. # 找到含有'('则'judge\_func(char',list2=['judge\_func','char']
4. **def** get\_function\_name(line):
5. temp\_list1 = line.split(' ')
6. **for** string **in** temp\_list1:
7. **if** '(' **in** string:
8. temp\_list2 = string.split('(')
9. name = temp\_list2[0]
10. **return** name
    1. Init\_Connections函数

函数声明: def init\_connections(tempfile,vertexs)

函数功能：以邻接表形式生成图结构。

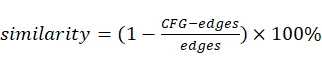
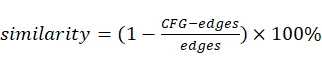
函数实现：见如下代码：

1. # 这个返回的是一个列表，列表里面存储的是元组，元组的第一个元素代表函数名a，第二个元素代表在函数a里被调用的函数。
2. **def** init\_connections(tempfile, vertexs):
3. # stack用来统计大括号，当左右大括号数目相等，表示一个函数体结束。connections存储所有的连接边(有向)
4. # i为一个索引，用来看这是第几个函数体，因为第一次VertexJudge的时候已经按顺序读入了函数结构体
5. # tempfile.seek(0)
6. stack = 0
7. connections = []
8. # print(vertexs)
9. i = -1
10. # 三个if先判断是否为函数体结构
11. **for** line **in** tempfile:
12. flag=1
13. **if** '(' **in** line:
14. **if** ('char' **in** line) **or** ('int' **in** line) **or** ('long' **in** line) **or** ('short' **in** line) **or** ('float' **in** line) **or** (
15. 'double' **in** line) **or** ('void' **in** line) **or** ('signed' **in** line) **or** ('bool' **in** line):
16. **if** ';' **not** **in** line:
17. # 如果满足这三个条件一定上一个函数体结束了，语法错误我们不考虑所以重新来定义name，以及stack归零
18. # 这里是有一个限制的：定义函数的时候{要在下面一行，否则无法根据不同的{的位置来判断stack是否要归零。
19. # 我的理解：因为不同的人编程风格不一样，所以对输入文件的风格要求应该是清晰的，在这里就要求{在下一行。
20. # 这属于细节问题，想尝试得到对于不同位置{都适用的方法，待所有功能完善之后再进行吧。
21. i += 1
22. name = vertexs[i]

25. # 先判断该行中是否存在vertexs里面的元素，如果存在就把(name,v)这个元组加入到connections里
26. # stack != 0是确保进入了函数体内部
27. **if** stack != 0:
28. **if** any(v **in** line **for** v **in** vertexs):
29. **for** v **in** vertexs:
30. **if** (v+'(') **in** line :
31. temp = (name, v)
32. connections.append(temp)
33. **if** '{' **in** line:
34. stack += 1
35. **if** '}' **in** line:
36. stack -= 1
37. # 当函数结构体循环完,重置name，等待新的函数结构体出现
38. **return** connections

4）Calculate函数

函数功能：使用Networkx获得有向图的编辑距离后，进行计算相似度。

函数实现：

5）Graph类的实现

类的功能：实现对图的增删改查。

类的实现：见代码：

1. **class** Graph(object):
2. """ Graph data structure, undirected by default. """
4. **def** \_\_init\_\_(self, connections, directed=False):
5. self.\_graph = defaultdict(set)
6. self.\_directed = directed
7. self.add\_connections(connections)
9. **def** add\_connections(self, connections):
10. """ Add connections (list of tuple pairs) to graph """
12. **for** node1, node2 **in** connections:
13. self.add(node1, node2)
15. **def** add(self, node1, node2):
16. """ Add connection between node1 and node2 """
18. self.\_graph[node1].add(node2)
19. **if** **not** self.\_directed:
20. self.\_graph[node2].add(node1)
22. **def** remove(self, node):
23. """ Remove all references to node """
25. **for** n, cxns **in** self.\_graph.items():  # python3: items(); python2: iteritems()
26. **try**:
27. cxns.remove(node)
28. **except** KeyError:
29. **pass**
30. **try**:
31. **del** self.\_graph[node]
32. **except** KeyError:
33. **pass**
35. **def** is\_connected(self, node1, node2):
36. """ Is node1 directly connected to node2 """
38. **return** node1 **in** self.\_graph **and** node2 **in** self.\_graph[node1]
40. **def** find\_path(self, node1, node2, path=[]):
41. """ Find any path between node1 and node2 (may not be shortest) """
43. path = path + [node1]
44. **if** node1 == node2:
45. **return** path
46. **if** node1 **not** **in** self.\_graph:
47. **return** None
48. **for** node **in** self.\_graph[node1]:
49. **if** node **not** **in** path:
50. new\_path = self.find\_path(node, node2, path)
51. **if** new\_path:
52. **return** new\_path
53. **return** None
55. **def** \_\_str\_\_(self):
56. **return** '{}({})'.format(self.\_\_class\_\_.\_\_name\_\_, dict(self.\_graph))

## 4.3 栈缓冲区溢出

### 4.3.1 检测流程

（1）扫描得到该函数所有的局部变量定义，得到变量名与该变量名所占用的空间大小；如果是数组，那就记录数组所占的空间大小；若为指针，记录其大小为0作为特别标注；将上述所有结果存入一个字典中。

（2）扫描该函数每一行代码，检查代码中是否存在敏感函数调用，包括strcpy(), strncpy(), memcpy(), memncpy(), strcat(), strncat(), sprintf(), vsprintf(), gets(), getchar(), fgetc(), getc(), read(), sscanf(), fscanf(), vfscanf(), vscanf(), vsscanf()等函数，如果存在，则记录其代码所在位置。

（3）若存在敏感函数，则发出警告。

## 4.4 堆缓冲区溢出

### 4.4.1 检测流程

（1）扫描得到该函数所有的局部变量定义，得到变量名与该变量名所占用的空间大小；如果是数组，那就记录数组所占的空间大小；若为指针，记录其大小为0作为特别标注；将上述所有结果存入一个字典中。

（2）扫描该函数每一行代码，检查代码中是否存在敏感函数调用，包括strcpy(), strncpy(), memcpy(), memncpy(), strcat(), strncat(), sprintf(), vsprintf(), gets(), getchar(), fgetc(), getc(), read(), sscanf(), fscanf(), vfscanf(), vscanf(), vsscanf()等函数，如果存在，则记录其代码所在位置。

（3）若存在敏感函数，则发出警告。

## 4.5 整数宽度溢出

### 4.5.1 检测流程

（1）为short，int，long, long五种类型提前赋权值，表示三种不同类型的长度大小

（2）扫描代码，存入所有的变量的类型及长度，此次扫描可忽略指针、数组。

（3）扫描该函数每一行代码，检查代码中是否存在赋值操作，检查赋值操作左右两边变量类型，如果存在右边整型宽度大于左边，则记录其代码所在位置；若存在指针的值赋给一个变量，依然判定为溢出，因为在此情况下大概率会发生溢出。并将位置存入列表中作为函数的返回值。

## 4.6 格式化字符串漏洞检测

### 4.6.1 检测流程

（1）扫描函数中出现printf函数的地方。

（2）检测printf函数中%的出现次数（即格式化输出字符串的个数，此处不考虑%%的情况），检测后面的参数个数，若二者个数不相等则存在格式化字符串漏洞的危险。

（3）出现以下状况均视为格式化字符串漏洞：

1. 格式化字符串中%转换说明个数与printf函数参数个数不符。
2. 格式化字符串中出现了%n转换说明。
3. 格式化字符串的第一个参数不是字面值。

（4）记录漏洞所出现的行数，存入列表中返回。

## 4.7 空指针引用

### 4.7.1 检测流程

（1）逐行检测代码对代码中的指针进行存储，并存储下指针声明或置为空的行数；若该指针后续被初始化就从存储结构中删除该指针。

（2）检测到特定指针在使用的时候是否还是处于未赋值的状态（即NULL状态，是否能从存储结构搜索到该指针），若使用时（如给其它变量赋值）还为空指针状态，则判定为空指针引用状况，记录代码所在的行数，记录到列表中作为函数返回值。

## 4.8 整数运算溢出检测

### 4.8.1 检测流程

（1）逐行检测代码对代码中是否出现乘法运算和加法运算，如果存在就发出警告，但是++或者指针则不会发出。

（2）记录代码所在的行数，记录到列表中作为函数返回值。

## 4.9 分布式任务调度

### 4.9.1 运行流程

（1）建立一个内存池，申明所需要的processor数量

（2）利用异步模式async用多个进程并行运行各个子程序

（3）将各个任务函数的返回值搜集起来，用列表存储并作为函数返回值

### 4.9.2 代码实现

1. **import** multiprocessing
2. **from** r2\_stringmatching **import** stringmatching
3. **from** r3\_FunctionAnalysis **import** FunctionAnalysis

6. **def** multiprocess():
7. pool = multiprocessing.Pool()
8. result = []
10. result.append(pool.apply\_async(FunctionAnalysis).get())
11. result.append(pool.apply\_async(stringmatching).get())
13. pool.close()
14. **return** result

## 4.10 UI模块

### 4.10.1 制作流程

此部分参照以下文档：

<https://likegeeks.com/python-gui-examples-tkinter-tutorial/#Get-input-using-Entry-class-Tkinter-textbox>

<https://tkdocs.com/tutorial/firstexample.html#design>

Tkinter是一个十分简易便捷的GUI制作工具，使用grid模块可以很好的实现布局，再把所有输入、按钮与先前的函数接口即可实现目的。

# 5 系统测试

## 5.1 字符串匹配同源性检测

### 5.1.1 样例测试

样例说明：相比于文件一，文件二将程序中大部分的关键字都用typedef预先定义好的关键字替换掉了。例如将int替换成int16。利用检测程序将程序中所有替换全部还原，预期测试结果二者相似度为100%，实际测试结果显示二者程序相似度为100%。

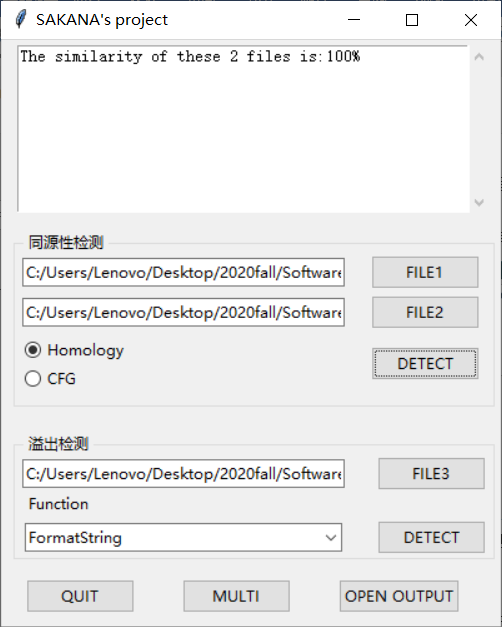


图5.1.1字符串匹配同源性检测测试结果

### 5.1.2 问题解决

1. 是否应该使用lex进行词法分析？

答案是非常应该，如果拷贝者仅仅对变量名等进行了修改和顺序调换，使用词法分析进行token比对能准确地识别出这一修改的原貌并标记为相似行，增大准确度。

1. 如何更有效地进行文本预处理？

首先应该忽略空行、注释、预处理等，并把宏定义恢复为原状。对于注释的识别，分为//和/\* 。对于//：若这一行一开始就为//，则这一行判定为注释；若不是，那接着识别直到出现//，将后面的内容去除。紧接着我们对预处理之后的文件进行分析即可。

1. 进行预处理后遇到哪些问题？

因为去掉了空行、注释、宏定义，在需要输出行数的功能中，倘若也进行了预处理，便会出现输出行数与原文件行数不对应的情况，为此我在UI新设计了一个功能，调用windows的记事本文件，打开预处理文件并便于显示和行数对应。

## 5.2 CFG匹配同源性检测

### 5.2.1 样例测试

样例说明：相对于文件一，文件二程序的函数调用结构更为复杂。这两个文件为我在密码学课设中所写的SPN完整文件，具体可见压缩包中的Untitled3,Untitled4。利用CFG同源性检测程序显示实际检测结果如下：

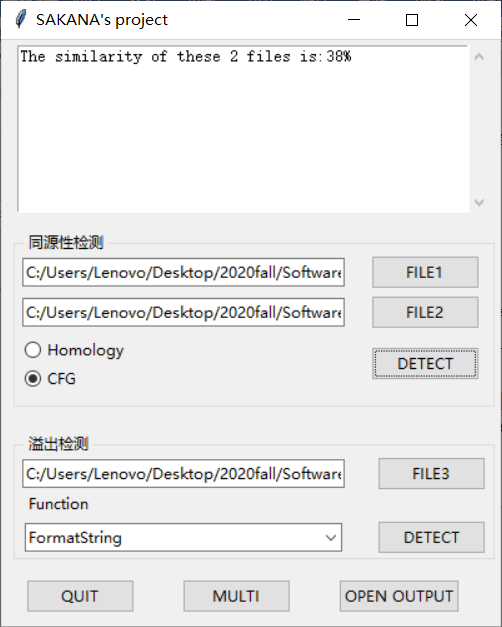
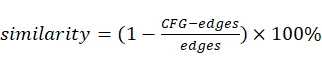


图5.2.1 CFG同源性检测测试结果

### 5.2.2 问题解决

1. 如何建立图？
2. 构造一个图类，完成增删改查功能。
3. 遍历代码，找到所有定义函数
4. 再次遍历代码，找到定义函数里调用的函数，生成字典{函数名:[函数中调用的函数们]}
5. 相似度度量方法：

参照字符串编辑距离算法，调用Networkx库求出编辑距离，最后的结果为：



1. 遇到的问题：

只可以检测完整的代码文件，因为需要获得所有函数的定义。

## 5.3 栈溢出检测

### 5.3.1 样例测试

样例说明：样本中存在部分缓冲区溢出的可疑函数，测试程序通过定义的测试规则指出可能发生溢出的代码行数和函数。

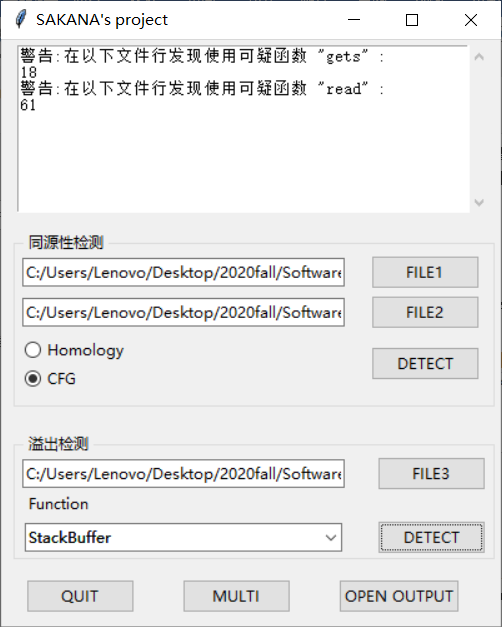


图5.3.1 栈缓冲区溢出测试结果

## 5.4 字符串匹配同源性检测

### 5.4.1 样例测试

样例说明：样本中有部分格式化字符串输出漏洞的可疑函数，测试程序通过定义的测试规则指出可能发生溢出的代码行数和函数。

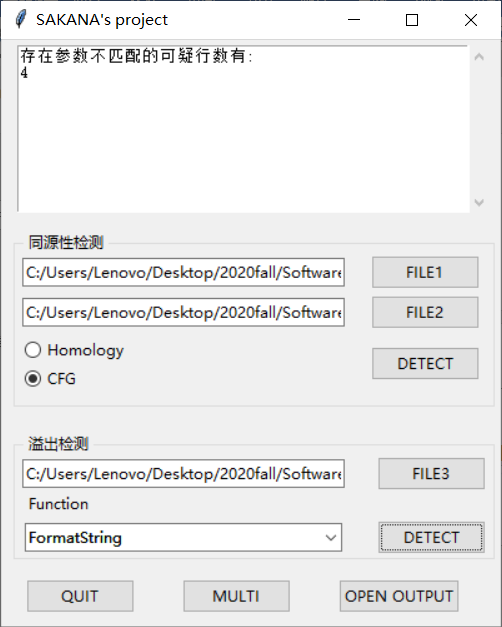


图5.4.1 格式化字符串输出漏洞测试结果

## 5.5 分布式任务调度检测

### 5.5.1样例测试

样例说明：多进程调度建立一个内存池，同时建立多个进程测试多个任务，并将返回的结果输出到界面上。

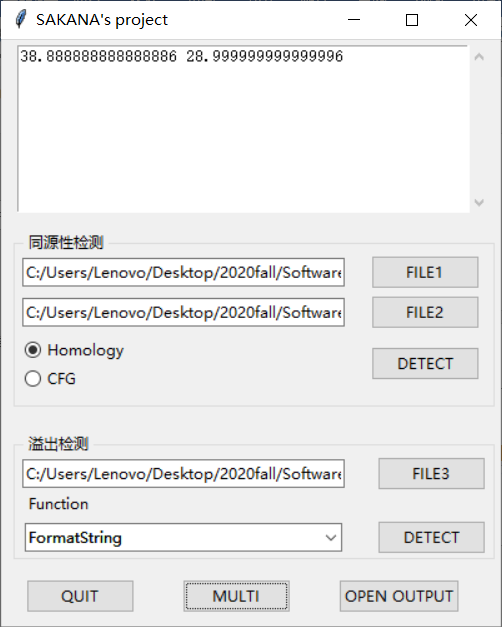


图5.5.1 分布式任务调度测试结果

## 5.6 堆溢出检测

### 5.6.1 样例测试

样例说明：样本中存在部分缓冲区溢出的可疑函数，测试程序通过定义的测试规则指出可能发生溢出的代码行数和函数。

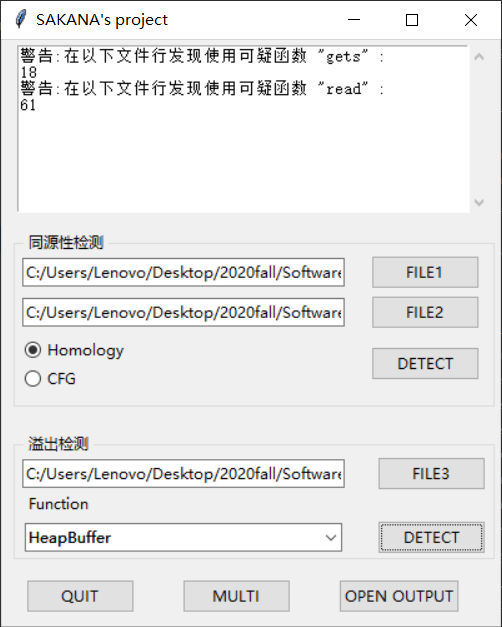


图5.6.1 堆溢出检测测试结果

## 5.7 整数宽度溢出检测

### 5.7.1 样例测试

样例说明：样本中有部分整数宽度溢出的可疑函数，测试程序通过定义的测试规则指出可能发生溢出的代码行数和函数。

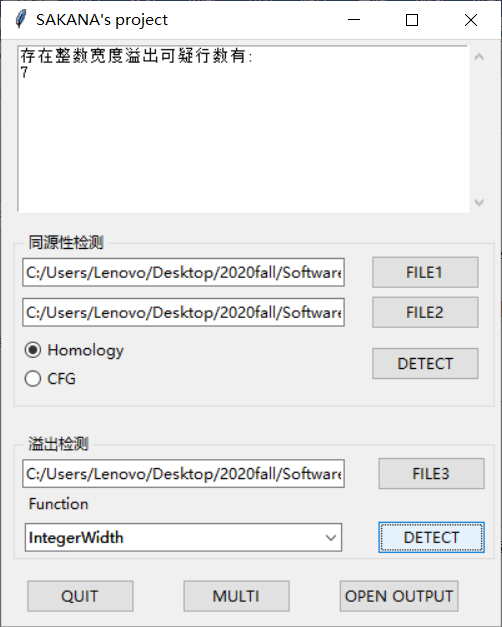


图5.7.1 整数宽度溢出测试结果

## 5.8 整数运算溢出检测

### 5.8.1 样例测试

样例说明：样本中有部分整数运算溢出漏洞的可疑函数，测试程序通过定义的测试规则指出可能发生溢出的代码行数和函数。

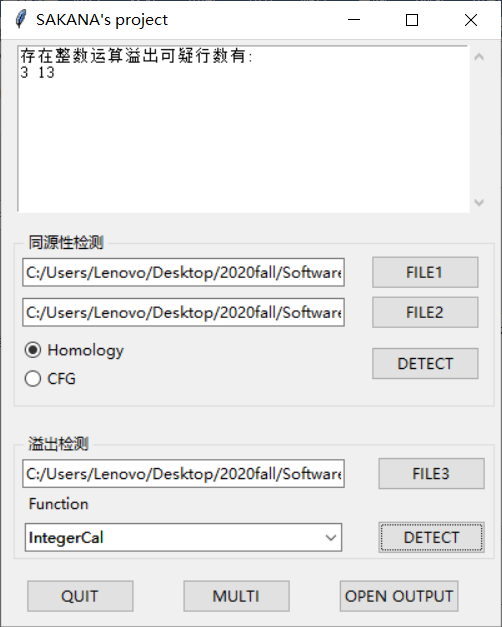


图5.8.1 整数运算溢出测试结果

## 5.9 空指针引用检测

### 5.9.1 样例测试

样例说明：样本中有部分出现空指针引用的可疑地方，测试程序通过定义的测试规则指出可能发生溢出的代码行数和函数。

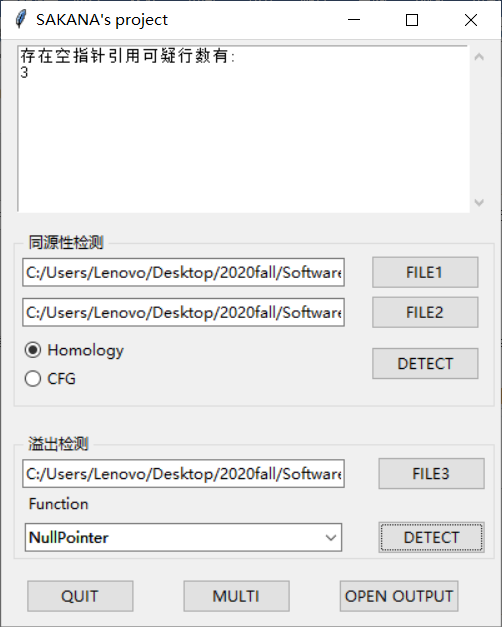


图5.9.1空指针引用检测测试结果

## 5.10 UI界面

### 5.10.1 界面说明



图5.10.1界面展示

UI界面分为四个主要部分：输出区域，同源性检测区域，溢出检测区域和补充功能区域。其中MULTI为多进程功能，OPEN OUTPUT作为解决预处理问题的补充方案，充当本次实验的特性所在。

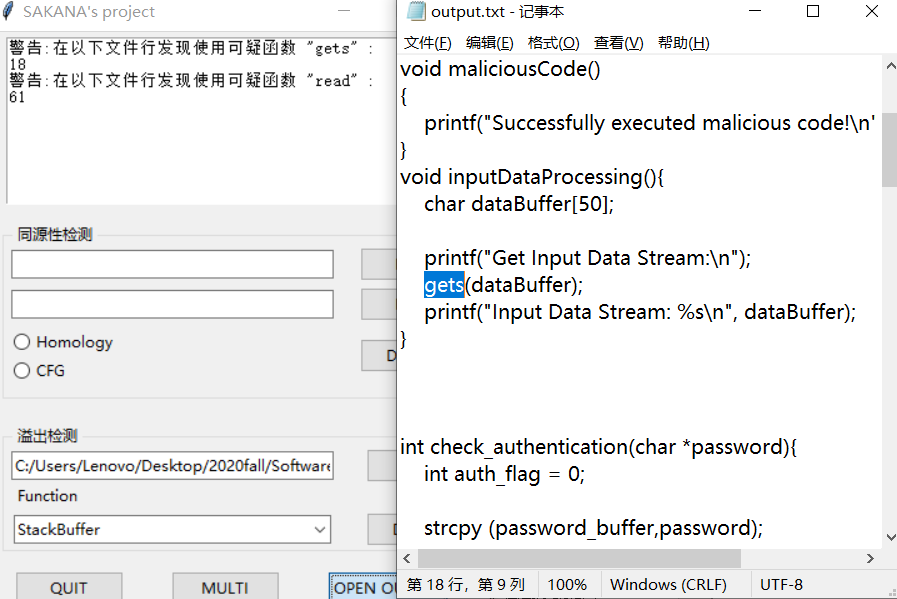


图5.10.2 特性展示

# 6 总结与展望

## 6.1 总结

本次实验是我第一次完成一个整合了GUI与后端代码的项目，本次实验完全用python编写，让我对python的语法特性有了更深入的了解。

对于本次实验中的同源性检测部分，虽然并非是之前课堂上所讲述的内容，但我也进行了较为深入的了解和研究。在课设刚开始的阶段，尝试着参考了一些前辈们的论文，虽然最后的实践过程中并没有完全运用到，仍然感觉受益匪浅。在完成字符串检测时，我根据任务书的提示了解了Lex这一词法分析工具，好在pylex的文档足够详实，样例清晰，耐下性子阅读一遍再复现一下代码，就实现了它在本次课设中的任务。

接着是CFG图检测，参考诸多论文后本来考虑是根据IF-WHILE决定代码具体走向的方式进行，但最后发现实现起来不够直观又相对复杂，转而使用了函数调用的方法，为了实现这一功能，也了解了一些关于图结构相似性的文章，最后根据字符串编辑距离算法，试着实现了图编辑距离算法，定义好相应的公式，达到了期望的目的。

多任务调度直接参考了python文档的multiprocessing部分，十分详细。

漏洞检测是用静态分析实现的，实际上是遇到了很多难点，本来期望达到更好的效果，参阅了一些开源的静态代码分析工具的工作成果，但由于时间有限理解不深，最后的实现只是在简单的程度上保证了相对高的精确性。

完成这次课设，实际上我感到非常享受。在这次课设中，我对于上个学期软件安全课程的内容有了更深入的了解和实践。除此之外，还有很多成长是超过课程内容之外的，比如说：为了更好的理解相关的知识，找到难解问题的解决方案，我开始尝试阅读官方文档、stackoverflow、英文教程、中文或英文的论文，这让我逐渐形成了信息检索和整理的能力，认识到英文社区和中文社区的差异性，也总算理解了为什么不少学长学姐们会说学好英语对计算机专业的学生是很重要的事情。另外就是，就是不断探索的想法得到了满足吧，虽然现在做的是一个toy-project，甚至离开了给定的样本库都无法继续实施功能，但也说明它可以参照前人的成果进行再学习和完善，而且，至少它在全班同学面前展示过。

## 6.2 展望与致谢

本次实验并非每一处都做到尽善尽美，即使每一个功能都实现了要求，但在样例测试部分，我也给出了部分功能上的改进之处，在学期结束的时候，我想我会把它继续做得更好。

希望自己能够参考一些开源项目的优秀结果，把自己的作品改进成一个至少可以供他人使用的项目。比如说对于漏洞检测部分：完善静态分析功能或者让它变得不仅仅是静态检测。对于同源性检测部分：增大预处理的强度，使代码不需要按照某种特定的规范书写，等等。

最后感谢一下本次课设的指导老师，在CFG图部分他给了我莫大的帮助，建议我采用函数调用分析而不是IF-WHILE类型分析的方法，加快了我的项目进度。同时也要感谢一下与我讨论本次课设的同学们，以及开源项目的贡献者，优秀文档的制作者们。

# 7 参考文献

1. 【译】Python Lex Yacc手册：https://www.cnblogs.com/P\_Chou/p/python-lex-yacc.html
2. PLY document：http://www.dabeaz.com/ply/ply.html#ply\_nn4
3. Python GUI Examples：https://likegeeks.com/python-gui-examples-tkinter-tutorial/
4. 《Tkinter GUI Application Development Cookbook - 2018》
5. python官方文档：https://docs.python.org/zh-cn/3.6/library/multiprocessing.html
6. python并发编程之多进程(实践篇)：https://www.cnblogs.com/jiangfan95/p/11439207.html
7. networkx官方文档：https://networkx.github.io/documentation/stable/tutorial.html
8. 图结构生成：https://stackoverflow.com/questions/11869644/
9. 空指针引用：CWE476和CWE690的翻译：
   * 1. https://blog.csdn.net/plstudio1/article/details/89788494
     2. https://blog.csdn.net/plstudio1/article/details/90577771
10. 图结构的相似性：
    * 1. https://stackoverflow.com/questions/9767773/calculating-simrank-using-networkx
11. The edit distance in graphs Ryan Martin：
    * 1. http://orion.math.iastate.edu/rymartin/talks/EditDist/editIITcolloq.pdf