“软件安全”课程设计报告

**.**

**班 级：　　　　IS1701　　　　　　.**

**姓名：　　　　罗南清　　　　　　.**

**学号：　　　　U201714868　　　　.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **项目** | **撰写规范** | **课程设计过程(系统设计、系统实现)** | **问题分析与小结** | **总分** | **教师签字** |
| 分值 | 20 | 50 | 30 | 100 |  |
| 评分 |  |  |  |  |  |

# 目　录

[1 课程设计任务书 1](#_Toc20515541)

[1.1 课程设计目的 1](#_Toc20515542)

[1.2 课程设计要求 1](#_Toc20515543)

[1.3 系统环境 2](#_Toc20515544)

[1.4 实验过程记录 3](#_Toc20515545)

[2 绪言 4](#_Toc20515546)

[2.1 字符串匹配同源性检测 4](#_Toc20515547)

[2.2 控制流程图CFG源代码同源性检测 4](#_Toc20515548)

[2.3 漏洞检测 5](#_Toc20515549)

[3 系统方案设计 6](#_Toc20515550)

[3.1 系统方案设计 6](#_Toc20515551)

[3.2 字符串匹配同源性检测 6](#_Toc20515552)

[3.3 CFG匹配源代码同源性检测 6](#_Toc20515553)

[3.4 栈缓冲区溢出 7](#_Toc20515554)

[3.5 格式化字符串漏洞检测 7](#_Toc20515555)

[3.6 分布式任务调度 8](#_Toc20515556)

[3.7 堆缓冲区检测 8](#_Toc20515557)

[3.8 整数宽度溢出检测 8](#_Toc20515558)

[3.9 整数运算溢出检测 8](#_Toc20515559)

[3.10 空指针引用 8](#_Toc20515560)

[4 系统实现 9](#_Toc20515561)

[4.1 字符串匹配的同源性检测 9](#_Toc20515562)

[4.2 CFG控制流程图的同源性检测 13](#_Toc20515563)

[4.3 栈缓冲区溢出 14](#_Toc20515564)

[4.4 堆缓冲区溢出检测 15](#_Toc20515565)

[4.5 整数宽度溢出 15](#_Toc20515566)

[4.6 格式化字符串漏洞检测 16](#_Toc20515567)

[4.7 空指针引用 16](#_Toc20515568)

[4.8 分布式任务调度 16](#_Toc20515569)

[4.9 整数运算溢出检测 17](#_Toc20515570)

[4.10 UI模块 17](#_Toc20515571)

[5 系统测试 18](#_Toc20515572)

[5.1 字符串匹配同源性检测 18](#_Toc20515573)

[5.2 CFG同源性检测 19](#_Toc20515574)

[5.3 缓冲区溢出 20](#_Toc20515575)

[5.4 格式化字符串输出漏洞 20](#_Toc20515576)

[5.5 整数溢出检测 21](#_Toc20515577)

[5.6 整数宽度溢出 22](#_Toc20515578)

[5.7 多进程调度 22](#_Toc20515579)

[5.8 空指针引用 23](#_Toc20515580)

[5.9 栈溢出检测 24](#_Toc20515581)

[6 总结与展望 25](#_Toc20515582)

[6.1 总结 25](#_Toc20515583)

[6.2 展望 25](#_Toc20515584)

[7 参考文献 26](#_Toc20515585)

# 课程设计任务书

## 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。

## 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。 |

## 系统环境

操作系统：Windows 10 pro

运行环境：Python 3.7.2, ply, NetworkX，Qt5，PyQt5

编程语言：Python 3.7

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

## 实验过程记录

09/02/2019 – 09/08/2019：1) 完成字符串的匹配的同源性检测

2) 控制流程图CFG的同源性检测

09/08/2019 – 09/12/2019：1）完成漏洞检测部分的代码

2）完成分布式任务调度

09/12/2019 – 09/15/2019：1）完成UI界面

2）完善了整个代码体系

3）完成漏洞检测样本库以及同源性检测样本库

# 绪言

## 字符串匹配同源性检测

当前，软件同源性检测技术主要分为二进制程序和源代码。计算机语言成分少，针对每门具体的语言都只有有限多个关键字，结构简单，因此基于代码层次的软件同源性检测技术具有实现简单、实行效率高等优势。源代码同源性检测主要有基于文件属性、字符串、Token、树和语义的方法等。

基于文件属性的方法采用文本信息、文本内容散列处理的方法，通过散列值比较，评估源文件（待检测文件）与目标文件的相似性。虽然检测性能高，但对于字符串替换模式的检测有效性较低。基于字符串的方法将源文件转换为字符串组，通常一个字符串对应于一行源代码，通过比对源代码程序段对应的字符串组，评估源代码间的相似性，但该方法只对于代码的完整拷贝或部分拷贝适用，对应代码拷贝修改的检测不鲁棒。基于树的方法采用语法树或抽象语法树产生源代码描述，文件比对通过语法树或抽象语法树中子树的匹配来评估[1]， 子树的信息可以进一步生产指纹信息，通过指纹信息的比对，形成可能存在的克隆点。基于语义的方法，通过程序依赖图（program dependency graph）中同构子图的匹配识别不同源代码中代码的克隆关系，该方法也被用于软件重构中的程序提取。基于语法和语义在检测能力方面具有较好的表现，但其性能通常无法满足实际要求。

基于Token的方法通过源代码的词法化产生Token（标号）序列，通过Token的比对实现源代码相似性度量。该方法相对于基于文件属性的方法和基于字符串的方法具有较好的鲁棒性，并且相对于基于树的方法与基于语义的方法具有较好的性能。然而，对于代码块结构中内容的修改，该检测方法还缺乏相应的手段，如在代码块结构中插入新语句、在代码块结构中删除语句等。

## 控制流程图CFG源代码同源性检测

基于图的抄袭检测方法[2]相对较少 , GPLAG从软件代码中构造程序依赖图(Program Dependency Graph, PDG), 通过子图同构匹配实现相似性计算; 此外, 为提高检测效率, 文章提出有 损过滤机制和基于 G-test 假设检验的过滤机制约减子图搜索空间。PDG 由于捕获了程序的数据和控制依赖关系、编码了程序逻辑, 抄袭者很难在不理解代码的前提下对 PDG作出修改, 实验证实其相比其他方法能更有效应对多种人工混淆手段。然而自动化混淆工具的混淆能力更强, 抄袭隐藏手段更高明; 此外, PDG的构建过程代价很高, 而且子图匹配是NP问题, 难以分析较大规模的软件。总的来说, GPLAG 是从学术抄袭到软件产品抄袭的初步实践, 它考虑了较为复杂的人工混淆手段, 但它对源码的需求及较高的时空花销, 使其不足以应对软件产品的抄袭。

## 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。

漏洞的危害评估主要依据美国的通用漏洞评分标准CVSS，其度量标准笼统，对漏洞危害性的评估不够准确。目前在大流量环境下，漏洞攻击数据包具有复杂度高，攻击行为具有高隐蔽性、持续性的特点，导致传统流量攻击检测方法精度低。

# 系统方案设计

## 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

## 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

流程图：

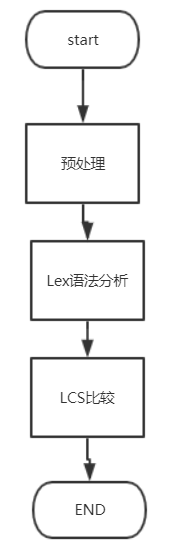


图1 字符串匹配同源性检测流程图

## CFG匹配源代码同源性检测

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

输出：输出的数据是代码相似度的百分比

流程图：

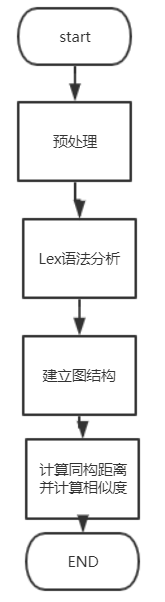


图2 CFG匹配同源性检测流程图

## 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：{行数，可疑函数名}

## 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{行数，函数名}

## 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

## 堆缓冲区检测

输入：输入数据来源是文件内容

功能：首先扫描代码分析函数中是否有可疑函数的使用，再分析函数调用参数的存在堆溢出的可能性。

输出：输出结果为可能发生溢出的行数与函数名，输出形式如此{行数，函数名}

## 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

## 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

## 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

# 系统实现

## 字符串匹配的同源性检测

### 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 格等。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Lex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，并依此形成行链表。

4）计算行摘要值。行数据结构中要记录行的摘要值。实现代码行摘要值的高性能计算及摘要值的唯一性是实现高效源代码比对的关键，行摘要计算将不考虑空格和ＴＡＢ，通常通 过Token自定义值与Token位置信息形成摘要值。

5）代码块比对。通过源文件和目标文件的相同行链表， 比较源文件和目标文件的链表中的信息摘要值，可以依次遍历 多个源文件和目标文件，实现文件的多对一对比，比对结果存 入行链表的相关位置，通常采用LCS方法定位相应代码中的同源性模块。

6）相似度计算。根据行链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。

### 关键函数说明

1）pretreate函数

函数声明：def pretreate(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

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

函数具体实现：

A）定义保留字：识别程序文件中的关键字，需要定义关键字集reserved，并设定关键字在识别过程中转换为什么token。举例如下：'if': '\_IF'，若词法分析器查找到if这个关键词，就会将其转换为\_IF。

以下为部分保留字：

reserved = {

'if': '\_IF',

'while': '\_WHILE',

'else': '\_ELSE',

'return': '\_RETURN',

'unsigned': '\_UNSIGNED'

}

B）定义Tokens：除了程序中会出现关键词之外，程序中会出现变量以及括号、等号、数字类似的符号。举例如下：t\_PLUS = r'\+'，其中t\_的前缀表示其是token，会利用r’’这样的正则表达式去匹配，若为’+’，则会将其变为\_PLUS，其它符号则类似。

以下为部分Token：

tokens = [ '\_NUMBER', '\_PLUS', '\_MINUS', '\_TIMES', '\_DIVIDE', '\_LPAREN', '\_RPAREN', '\_LBRACKET', '\_RBRACKET', '\_LBRACE', '\_RBRACE', '\_EQUAL', '\_ASSIGN', '\_SEMI', '\_COMMA', '\_STRING', '\_TYPE', '\_ID', '\_BACKSLASH', '\_LESSTHAN', '\_MORETHAN', '\_QUESTION', '\_COLON', '\_AND', '\_CHAR']

C）建立lex分析器对象及使用lex分析器：lexer = lex.lex()，lexer是建立的lex分析器对象，利用lexer.input(data)将要处理的数据输入到lex分析器中，然后不断读取每一行的内容按照定义的规则进行转化，最后将所转化的token文件返回。

### LCS函数

LCS函数用于代码相似度的匹配，其通过一个矩阵来记录两个序列中所有位置的元素的匹配情况，矩阵的行与列分别表示不同的字符串，如果位置字符匹配，则将对应的矩阵位置元素标记为符合一定规则的数字，然后找出符合算法约定的标记位置及匹配长度，确定两个序列中最长的 公共子序列，从而判定匹配字符串的位置及长度。

字符串ｖ的子序列可以定义为：源于ｖ的有序的字符子序 列，该序列在ｖ中不一定连续。例如，如果 ｖ＝ＡＴＴＧＣＴＡ，那么 ＡＧＣＡ和ＡＴＴＡ为ｖ的子序列，而 ＴＧＴＴ和 ＴＣＧ不是ｖ的有效子序列。最长公共子串问题可以定义为：在多个字符串中，找出字符外形的顺序一致的最长公共字符子串序列，即满足上述条件的公共子序列。通常通过以下方式定义 ｖ＝Ａ０Ａ１… Ａｍ－１和ｗ＝Ｂ０Ｂ１…Ｂｎ－１的公共子序列：

在ｖ中子序列的位置满足０≤ｉ１＜ｉ２＜…＜ｉｋ≤ｍ－１。

在ｗ中子序列的位置满足０≤ｊ１＜ｊ２＜…＜ｊｋ≤ｎ－１。

ｖ和ｗ中相应的符号在以上相应的位置一致 ｖｉｔ＝ｗｉｔ，１≤ ｔ ≤ｋ 。

行时间复杂度为Ｏ（ｍｎ），空间复杂度为Ｏ（ｍａｘ（ｍ，ｎ））。 ＬＣＳ是实现序列相似性分析的最有效手段之一。其中求解的字符串子序列是源于带比较字符串中的公共最长有序子序列，但不一定是连续的。

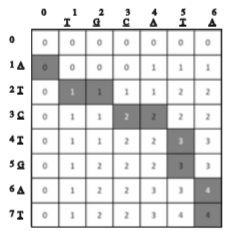
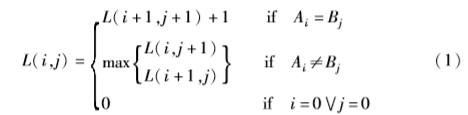
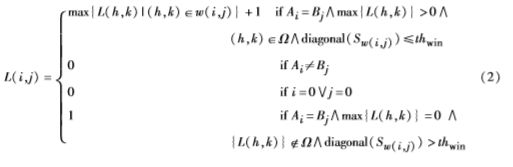
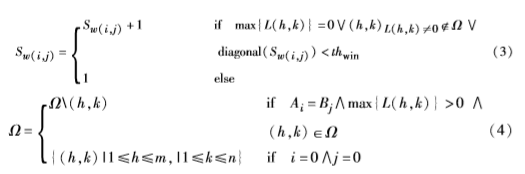


图3 计算w与v的相似性







算法步骤：

ａ）初始化矩阵的第１行与第１列为０。初始化未标记集合Ω为所有矩阵坐标位置。当前行列号为：行号赋值为１，列号赋值为１。设置匹配计数变量 Ｎｍａｔｃｈ＝０。初始化阈值thｗｉｎ值。

ｂ）初始化可扩展正方形窗口尺寸 Sw（ｉ, ｊ）＝１。

ｃ）在当前行列号处，匹配源文件摘要值和目标文件摘要值。

ｄ）如果匹配失败，转ｅ）；如果匹配成功，转f )。

ｅ）在矩阵相应位置填入０，转ｍ）。

ｆ）根据式（３）改变窗口尺寸。

ｇ）判断窗口尺寸是否超出了阈值范围，如果否，转 ｈ）；否 则转ｌ）。

ｈ）搜索窗口中最大非零元素。如果存在非零最大元素， 转ｉ）；否则转ｆ）。

ｉ）判断该矩阵位置索引（ｈ，ｋ）是否在标记集合Ω，如果是， 转ｊ）；否则转ｆ）。

ｊ）根据式（２），将非零最大元素加１，填入矩阵相应位置，记录矩阵的位置到匹配位置集合链表。根据式（４）将矩阵匹配位置索引从Ω中剔除。转ｍ）。

ｋ）如果不在未标记集合，转ｆ）。

ｌ）矩阵当前位置元素置为１，根据式（３）将窗口尺寸置为 １。

ｍ）当前列号加１。如果超出矩阵列数，将行号加１，列号 赋值为１。如果超出矩阵行数，转ｎ）；如果未超出矩阵行数，转ｂ）。

ｎ）统计匹配位置集合链表中长度大于阈值 thｓｅｑ的成功匹 配子序列中不同元素的个数，即相似性行个数，记为 linesimilar。其中thｓｅｑ标记有效匹配子序列的最小长度。

ｏ）计算源文件代码与目标文件代码的相似性。

### 相似性计算

CalculateSimilarity函数，依据公式

similar＝linesimilar／linewaitingcodes，

计算出相似度similar

## CFG控制流程图的同源性检测

### 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、空格等。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Lex完成。

3）建立代码数据结构。对于程序中的重要关键词建立图的数据结构，其中包括while，if-else的分支或循环结构。

4）计算相似度。利用图的同构算法，计算由两个文件生成的两个图的图距离，计算两个图的相似度。

### 关键函数说明

1）pretreate函数

函数声明：def pretreate(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

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

函数具体实现：

A）定义关键字：识别程序文件中的关键字，需要定义关键字集reserved，并设定关键字在识别过程中转换为什么token。举例如下：'if': '\_IF'，若词法分析器查找到if这个关键词，就会将其转换为\_IF。

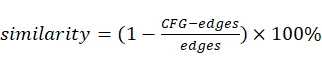
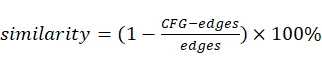
B）定义Tokens：除了程序中会出现关键词之外，程序中会出现变量以及括号、等号、数字类似的符号。举例如下：t\_PLUS = r'\+'，其中t\_的前缀表示其是token，会利用r’’这样的正则表达式去匹配，若为’+’，则会将其变为\_PLUS，其它符号则类似。

C）建立Lex分析器对象及使用Lex分析器：lexer = lex.lex()，lexer是建立的lex分析器对象，利用lexer.input(data)将要处理的数据输入到Lex分析器中，然后不断读取每一行的内容按照定义的规则进行转化，最后将所转化的token文件返回。

3）optimize\_graph\_edit\_distance函数

返回图G1和G2之间的图编辑距离，图编辑距离是用来描述图的相似性程度的，它用来描述两个图之间最小的相差距离，这个相差距离指的是两个图的同构距离。

输出两个图之间最小的同构距离CFG，根据公式



计算两个图的相似性。

4）initgraph函数

先识别程序内容的关键词建立图，主要的关键词是if-else的分支结构和while循环结构。

(a) if-then-else 分支结构 (b) while 循环结构 (c)有两种退出的分支结构 (d) 分支-循环复合结构

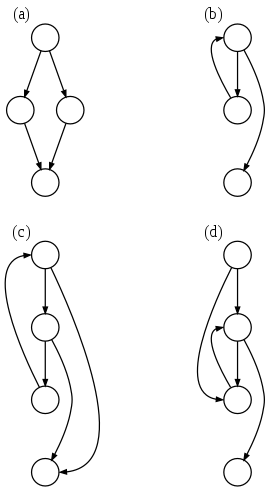


图4 CFG示意图

## 栈缓冲区溢出

### 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）对上述函数的调用参数进行检查，若传入的参数已经超过函数本来的空间大小，则判定溢出可能。若函数传入的为指针，则判定为溢出，因为指针传参没有检查参数长度，很大概率会溢出。若函数中规定了传入参数长度，那判定传入参数长度是否超过了规定长度，若超过，那必定为溢出。将所有确定性溢出和可能性溢出的结果装入字典中返回。

## 堆缓冲区溢出检测

### 检测流程

（1）扫描函数，寻找程序中malloc和calloc函数，记录下变量名和变量大小

（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）对上述的可以函数中，扫描是否存在在字典中的变量；若存在，检测变量大小与传入参数的大小，若大于则判定为溢出。并将溢出的结果存入字典中作为返回值

## 整数宽度溢出

### 检测流程

（1）为char, short，int，long, 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。

## 格式化字符串漏洞检测

### 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

## 空指针引用

### 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

## 分布式任务调度

### 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

## 整数运算溢出检测

### 检测流程

（1）扫描代码中的变量，并将变量的类型、长度和数值记录

（2）扫描该函数每一行代码，检查代码中是否存在整数运算操作

（3）若存在整数运算操作，则判断两数相加是否会超过其定义的机器数字长度；若超过了，则返回该行行数

（4）将可疑的行数装入列表存储，作为函数的返回值

## UI模块

### 制作流程

（1）配置环境：python 3.5及以上；安装pyqt5-tools

（2）运用Designer设计好界面，Designer界面特别容易设计，仅仅是拖动组件就可以将界面做好，再保存后生成.ui文件

（3）将.ui文件转换为.py文件，此处需要安装pyqt5-tools，将.ui文件用python和pyqt5编译成python文件，使得这些接口能被python程序调用

（4）建立GUI.py文件将各个任务的主函数与设置按钮相互链接起来，链接即是利用.py文件中的接口将函数的运行和鼠标的点击相互链接



图5 链接接口

（5）将各个函数的返回值输出到定义好的对话框中，注意返回值需要转化为str类型

# 系统测试

## 字符串匹配同源性检测

### 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%。

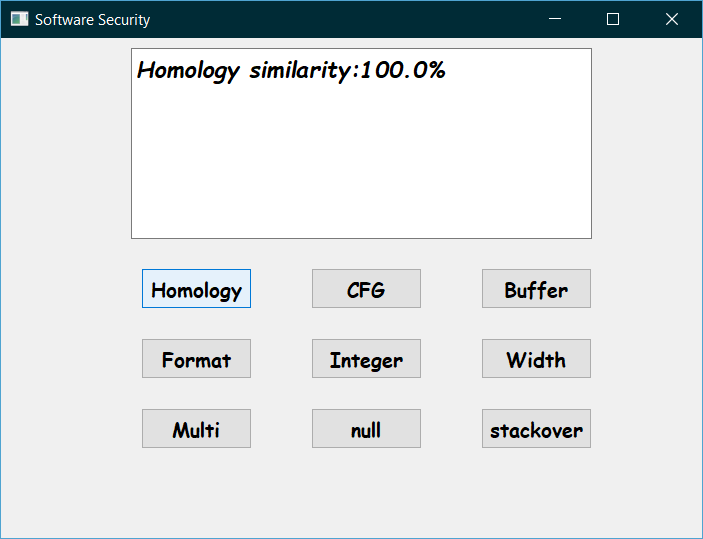


图6 字符串匹配同源性检测测试结果

### 问题解决

1） 在预处理文件中如何正确去除一行的注释？

一开始只是简单识别，如果一行中出现了注释的开始符号//，那么就判定该行为注释。这样做很简单，但是存在一个很严重的问题，若一行的注释放在了最后或者说在输出中存在//这个字符，那么这一行的有用代码信息将全部丢失。

所以，在此之后将此分为两种情况，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。

2）如何用Lex来分析文件？

一开始想到利用Lex来分析文件可能有点超出自己能力范围 ，后来在字符串的匹配上遇到很多困难，于是开始尝试利用Lex来分析文件token，同时发现自己在有正则表达式的基础上理解Lex词法分析器相对更容易。于是，通过阅读官方文档，将其不断调试最终正常运行。

## CFG同源性检测

### 样例检测

样例说明：相对于文件一，文件二程序将一些赋值语句，判断语句的顺序混乱，并将函数名更换，预测二者的相似度计算应该是100%。利用CFG同源性检测程序显示实际检测结果二者程序完全相同。

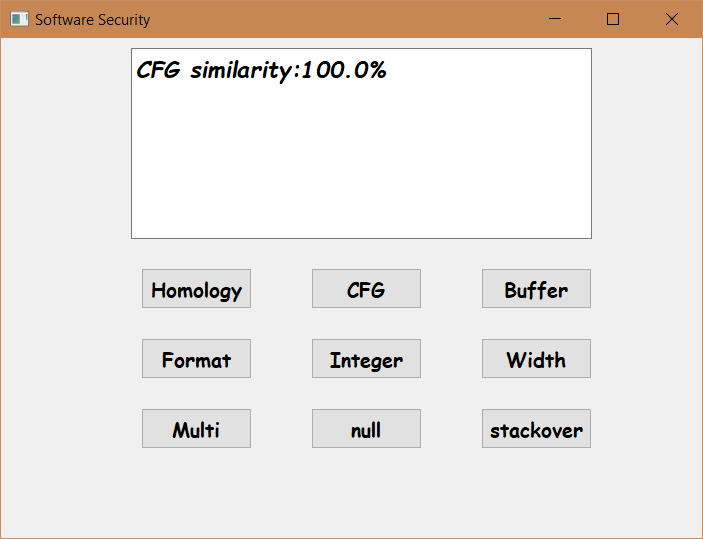


图7 CFG同源性检测测试结果

### 问题解决

1）如何将if-else的分支结构和while的循环结构建立成图？

我选择邻接表来建立图的数据结构，表头是头节点，后面的节点是有向图中该节点出度方向的节点。在if或者while出现时，找到其开始标志，即’{‘，将这一行的序号记下来，存入栈中，随后开始进入循环或者分支结构中。如果内部出现了新的结构，则继续将开始符号’{‘装入栈中，直到’}’出现，不断一个一个将栈中的符号清空，标志着该结构的结束。

2）相似度度量方法

最后将所有的节点序号组合进一个列表，利用python库文件NetworkX将节点和边关系打包进入函数，再然后利用图的同构算法计算图的同构距离。再利用二图的同构距离计算二者的相似度。

3）将邻接表数据打包进NetworkX库文件函数中？

由于NetworkX要求输入的节点关系是列表中的多个元组，所以要将所有的节点关系的二维列表的一维列表全部变为元组。并且还要先导入所有节点，所以要先将所有节点提取出来，再将处理过的列表元组导入节点关系中建立边。

## 缓冲区溢出

### 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。

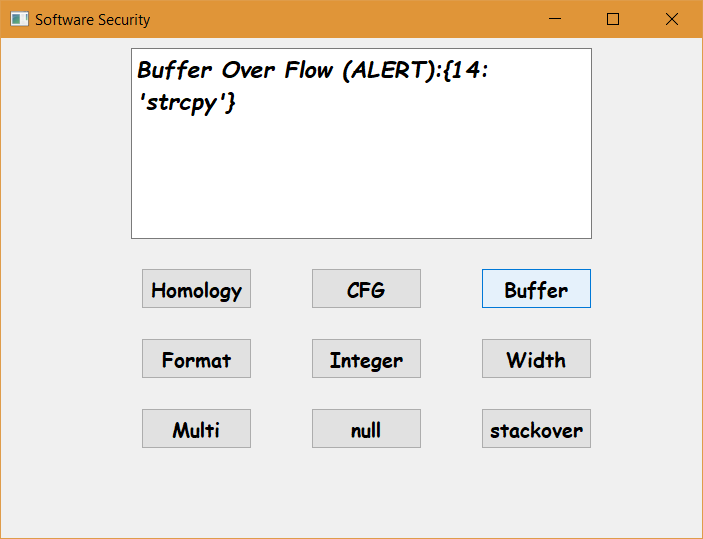


图8 缓冲区溢出测试结果

## 格式化字符串输出漏洞

### 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。

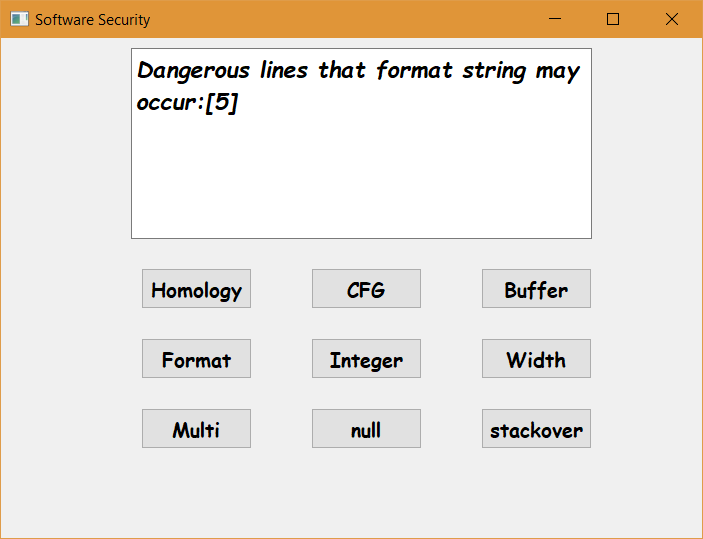


图9 格式化字符串输出漏洞测试结果

## 整数溢出检测

### 样例测试

样例说明：样本中有部分整数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。

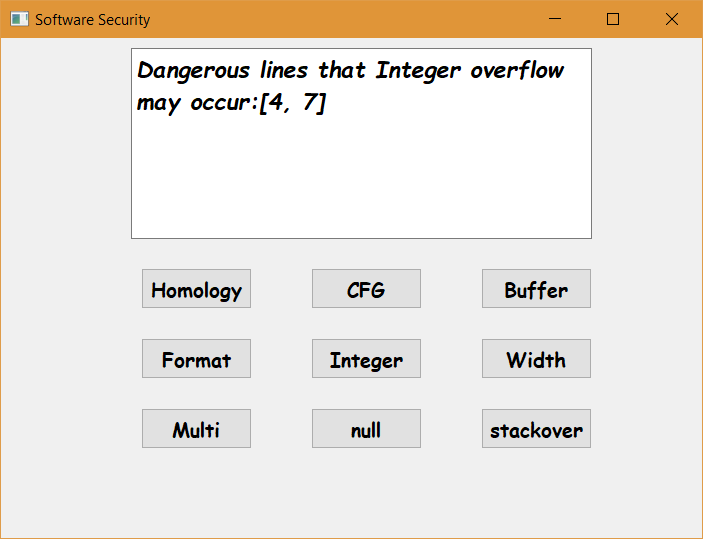


图10 整数溢出漏洞测试结果

## 整数宽度溢出

### 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。

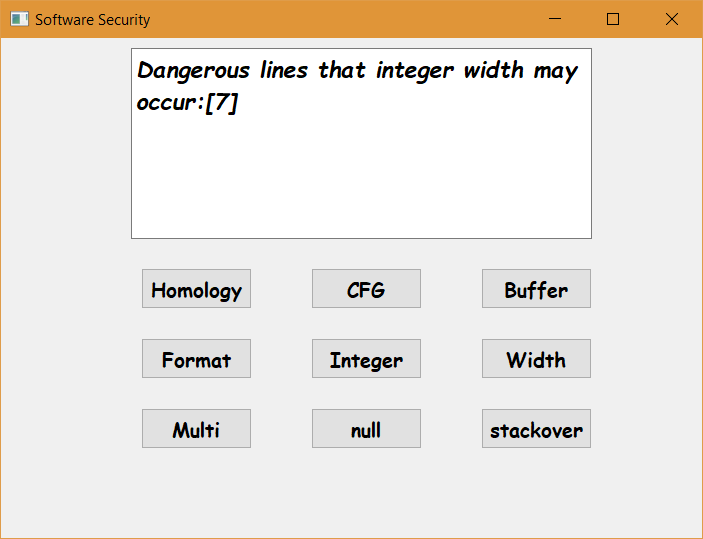


图11 整数宽度溢出测试结果

## 多进程调度

### 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。

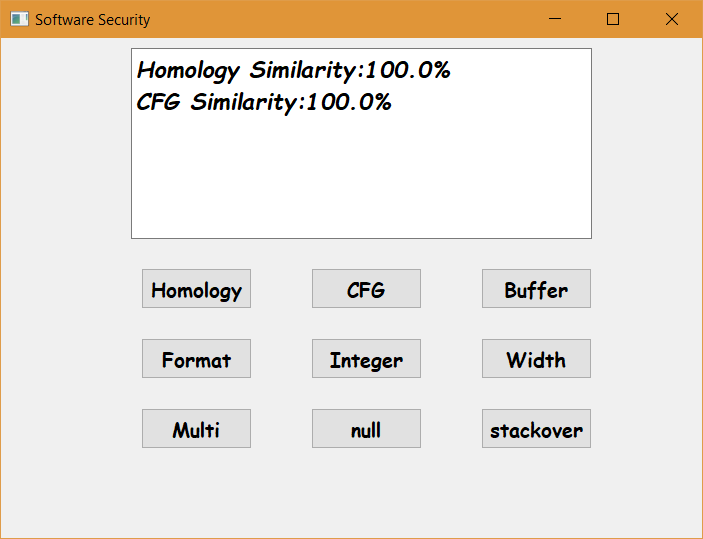


图12 多进程调度测试结果

## 空指针引用

### 样本测试

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。

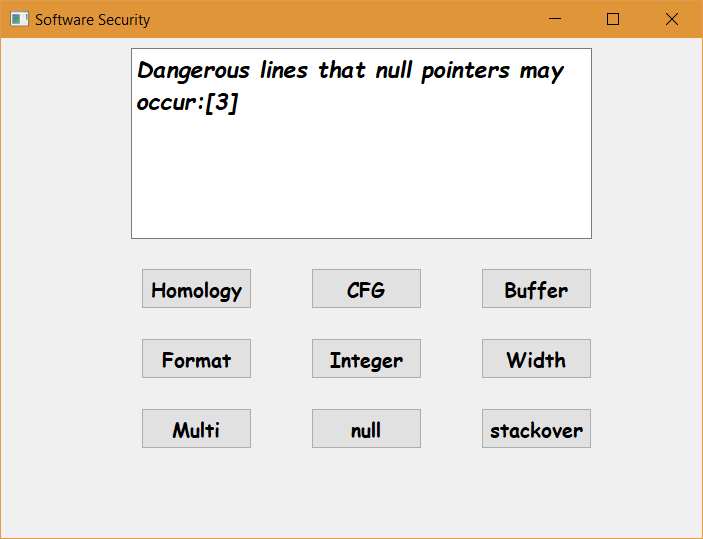


图13 空指针引用测试结果

## 栈溢出检测

### 样本测试

样例说明：样本中有部分栈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名

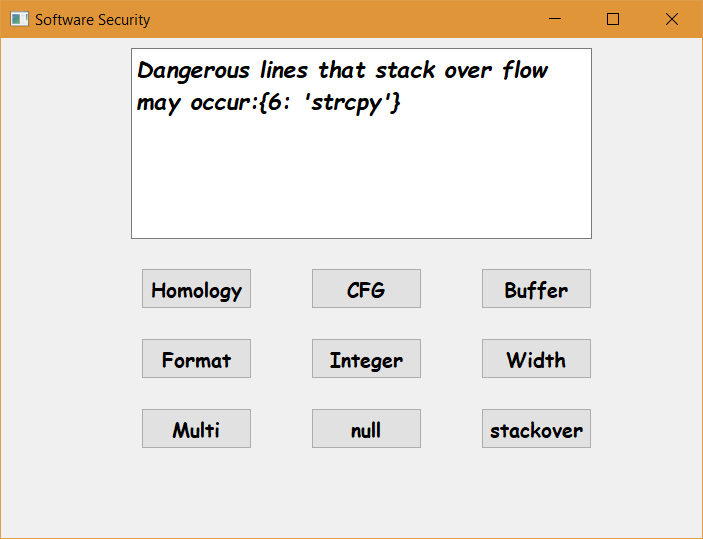


图14 栈溢出检测

## UI界面

### 样例测试

样例说明：UI界面上有九个功能按钮，鼠标点击之后就会执行相应模块的代码，并将返回值输入到对话框中，左上角是该对话框的名字 Software Security

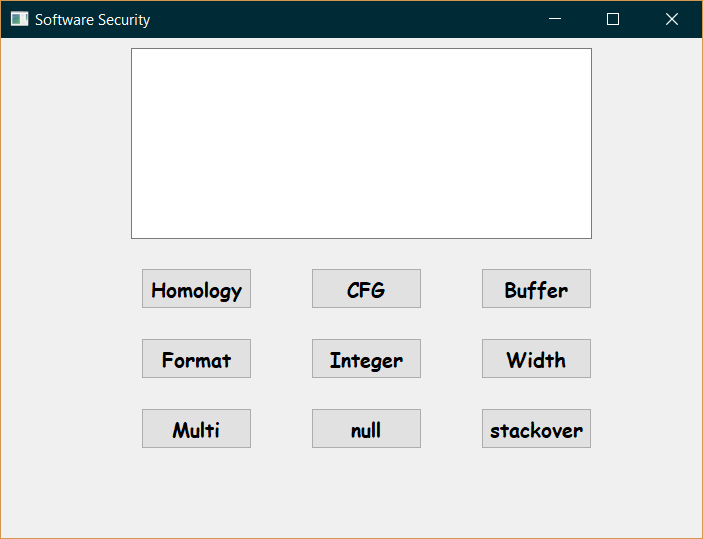


图15 UI界面

# 总结与展望

## 总结

通过本次实验，我对python语言、以及Lex语法分析器、图结构更加熟悉了。字符串的同源性比较是基于Lex语法分析器和最大字串比较算法现，这一次我认识到Lex语法分析器功能的强大，同时也了解了一部分语法分析器的工作原理，对于正则表达式有了进一步的认识。我通过读论文的形式，了解到字符串匹配同源性检测的算法思想，采用改进版本的LCS算法，虽然对于小的样例来讲，改进版本和传统典型算法不会有什么区别，但是我更愿意去探究新的东西。

其次就是CFG同源性的检测了，刚开始拿到完全没有思路，对于整个算法的流程都是模糊的，后来通过阅读一些资料，然后在一张白纸上进行总结，总结出了清晰的思路。并将思路实现到代码中，利用了图结构。建立图的这个过程对我来讲有些困难，因为之前没有怎么用过，但最终通过不断查阅资料把最终的数据结构建立出来。在这过程我认识到了算法和数据结构对于程序的思考的确很重要，是我看到自己在这两方面存在巨大的差距，为我之后的学习指引了方向。

漏洞检测是做的比较基础的一块，因为静态代码的分析得到的具体信息很有限，所以大多数情况下做的比较简单。但在这个过程深入理解了各种漏洞的代码层面的特征，通过分析特征来指定可能发生漏洞的地方。

## 展望

字符串匹配同源性检测仍然做的比较基础，在安全领域也属于一个在不断发展的学科分支，尤其是在LCS算法的比较上。软件同源性检测技术在软件重构性检测、软件知识产权保护、计算机 犯罪司法取证、研发者合法权益维护等方面发挥着积极的作用，已经成为研究热点。CFG控制流程图的检测是基于图的检测，是将代码转换为程序依赖图，对图中结点进行分类获取匹配结点，分析结点的数据依赖和 控制依赖矩阵，查找图中的同构子图并对其进行泛化处 理，计算程序的差异向量距离实现克隆检测。这种检测方法对于断层克隆极为有效，但它对更深层次的功能克隆效果不明显，并且图的计算效率也不高，不适用于大的数据文件。漏洞检测是安全领域发展的一个重要方向，我们仅仅是做的是高级语言层面的静态分析，深入层次的漏洞检测我们还没有接触，希望会有机会接触。

# 参考文献

[1] ＢＡＸＴＥＲＩＤ，ＰＩＤＧＥＯＮＣ，ＭＥＨＬＩ ＣＨＭ．ＤＭＳ: ：program transformations for practical scalable software evolution [C]//Proc of the 26th International Conference on Software Engineering.2004:625-634

[2] C. Liu, C. Chen, J.W. Han, and P.S. Yu, "GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis, " in Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'06), pp. 872-881, 2006.

[3] Behzadan V,Munir A.Vulnerability of deep reinforcement learning to policy induction attacks[M]//Machine Learning and Data Mining in Pattern Recognition.Cham:Springer,2017:262-275.

[4] Dhingra M,Jain M,Jadon R S.Role of artificial intelligence in enterprise information security:A review[C]//Proceeding of the 2016 Fourth International Conference on Parallel,Distributed and Grid Computing.Waknaghat:IEEE,2017:188-191.