CSTE：Computer Security Test Environment，计算机安全测试环境

一、设计目标：

1）用于研究学习计算机系统的漏洞、攻击和防御。

2）用于评测计算机系统的安全性，提供标准测试集和整体安全评估测试方案。

二、设计需求：

1）可扩展性。方便添加各种漏洞程序和攻击样例。

2）模块化，独立性。CSTE中的漏洞程序和攻击样例具有独立性和模块化，既能够单独运行，也能够通过CSTE统一调用。

3）易用性。方便用户安装和使用，方便用于学习和研究，能够很方便的运行漏洞程序和攻击样例，观察漏洞程序和攻击样例运行过程，研究具体原理和机制。

4）可分割性，可整合性。可以按照一定的分类，对测试集进行分割和整合。

三、CSTE框架架构

3.1 CSTE的整体架构

1）run\_all.py：主调程序，负责CSTE的整体运行和调用。

2）configure.sh：配置安装程序，安装CSTE并配置运行环境。

3）version：CSTE的版本号。

4）README.md：CSTE的简要介绍。

5）tools：小工具，具备一定功能的小程序，供用户和主调程序run\_all.py使用

6）report：CSTE运行报告，存放主调程序run\_all.py运行的结果。

7）doc：说明文档，CSTE的整体介绍和使用说明。

8）src：CSTE的主体部分，包含所有的漏洞程序和攻击样例。

3.2 src的架构

1）basic：基本测试集

2）ext：扩展测试集

3）study：用于学习的测试

4）research：用于研究的测试

5）unsorted：未分类测试

每个测试集中都包含许多漏洞程序和攻击样例的文件夹。

每一个文件夹中有一个漏洞程序和对应的一些输入测试（包括普通的正常输入样例和恶意的攻击样例。）

不同的文件夹之间应该是完全独立的，没有任何的关系。

3.3 单个漏洞程序及其攻击样例的架构

1）vul：漏洞程序源代码及其可执行文件，漏洞程序全部用C语言编写。

2）attack：攻击程序。

3）input：攻击样例和正常输入，正常输入文件的名称应该加上\_normal后缀。

4）output：输出漏洞程序运行的结果。

5）check：检测程序，判断漏洞程序是否被攻击成功。

6）doc：说明文档，详细介绍漏洞程序的结构和功能，普通输入样例的作用，攻击样例的详细攻击过程。

7）compile.sh：编译命令，生成漏洞程序可执行文件的编译命令及其选项，全部使用gcc进行编译。

8）define.txt：漏洞程序及攻击样例的定义。

3.4 define.txt说明

格式为：属性名=属性值，#为注释。具体内容如下：

#漏洞程序和攻击样例的基本信息

name：漏洞名称，应该和漏洞程序的名字一致，应该尽量体现漏洞程序的特征

designer：设计者

intro：对漏洞程序的简要描述

#配置信息

run\_all：是否支持主调程序调用

check：是否有测试程序check

#漏洞程序基本信息

bench：所属测试集

bit：32位或64位程序

vul\_type：漏洞类型

vul\_num：漏洞程序中漏洞的数量

#攻击样例基本信息

attack\_model：攻击模型，单纯的输入数据攻击，或者利用攻击程序进行攻击。输入数据可以有多个，而攻击程序只应该有一个。

input\_num：输入样例的数量，包括攻击样例和正常输入

normal\_num：正常输入的数量

attack\_num：攻击样例的数量

normal\_class [0 ~ normal\_num-1] {

normal\_name：普通输入名字，应该和普通样例文件名一致

normal\_intro：对普通输入样例的简要描述

}

attack\_class [0 ~ attack\_num-1] {

attack\_name：攻击样例名字，应该和攻击样例的文件名一致

attack\_type：攻击类型

aslr：alsr打开或关闭

dep：dep打开或关闭

stack\_cookie：栈cookie打开或关闭

attack\_intro：对攻击样例的简要描述

}

四、用户使用说明

本章介绍CSTE的基本使用方法。

4.1 安装：直接运行configure即可。

4.2 使用：直接运行run\_all.py即可。主调程序run\_all.py用Python语言编写。

4.2 主调程序的基本命令说明：

1）help：帮助，打印基本命令及相关介绍

2）

3）

五、开发者添加说明

本章介绍如何向CSTE中添加新的漏洞程序和攻击样例。

5.1 命名规范

所有名称全部使用英文小写，多个单词之间用下划线“\_”隔开。常见的单词可以用缩写，如处理器（cpu），漏洞（vul），文档（doc）等。

5.2 添加漏洞程序和样例

1）在src中建立一个文件夹，命名为：漏洞名称+设计者标识+编号

2）然后在文件夹中，分别建立vul、input、output、attack、check、doc等文件夹和define.txt、compile.sh等文件。

3）按照要求在各个文件夹中建立对应的文件，注意名称。

4）在vul文件夹中用C语言编写漏洞程序，并且用gcc生成可执行文件。漏洞程序的名称应该和define.txt中的name一致。

5）在input中存放漏洞程序所需要的正常输入文件和对应的攻击样例。

6）漏洞程序的输出放入output中。

7）按照规范在check中自定义检测程序。

8）按照要求编写define.txt和compile.sh文件

9）在doc中编写详细漏洞程序说明文档和攻击样例说明文档

主调程序run\_all.py一般情况下不需要修改，按照使用说明使用即可，

5.3 添加检测程序

检测程序check，同样用C语言编写。在程序的开头注释编译选项，程序中的检测函数定义为char\* check(){}，返回值类型为char\*，函数名为check，不需要参数，check文件夹中只存放.c和.so文件。

针对vul中的漏洞程序的攻击，在check中必须存在相应的检测程序，检测攻击是否成功，并将结果反馈给主调程序，如果检测程序需要攻击程序的输出作为判断，则将输出信息存放到output中，供check使用

对于检测程序check的反馈信息有如下规定：

检测程序的反馈信息将作为产生报告的重要依据，所以反馈信息要尽量详细

如果检测到攻击失败，则返回值的起始部分必须为“attack fail,”,再在它的后面加上攻击失败的原因