README for Buffer Lab

I 实验原理

本实验为缓冲区溢出实验，其基本原理非常简单，和C语言的运行时和Gets函数有关，下面是一个简单的例子

```
unsigned getbuf() {
    char buf[BUFFER_SIZE];
    Gets(buf);
    return 1;

}
```

在示例代码中，getbuf函数分配了一个大小为BUFFER_SIZE的临时数组buf,那么在运行时，将会在栈上分配BUFFER_SIZE个char那么大的一段空间。 同时，我们的Gets是从标准输入中读取输入，并将读到的一串字符放置到buf指向的缓冲区。当读到的这串字符长度不超过缓冲区，那么平安无事；但是遗憾的是，当读到的这串字符长度超过缓冲区长度，就代表位于缓冲区的更高字节的内存位置的原有数据会被覆盖，这些数据可能是当前函数分配的临时数据，可能是返回地址，也可能是保存的寄存器，甚至是上一个函数的"frame"...(参考教材中的内容，并可以上网查阅"x86函数调用约定")。在本次实验中，我们正是要利用后一种现象，来达成我们的目标--缓冲区溢出攻击。

II 实验步骤

英文的实验说明中给出了非常详细的介绍，其中可以忽略英文说明中有关获取文件，以及提交的说明（我们的文件educoder的虚拟机上，提交使用educoder的控制台）。

实验文件

实验文件位于educoder虚拟机的/data/workspace/myshixun/,目录overflow1-overflow5分别对应5个任务。

以下是实验中用到的文件，以overflow1为例

- bufbomb:一个容易遭受缓冲区溢出的程序。
- hex2raw:一个工具，将16进制数文档转换成字符文档。用法见英文文档。
- makecookie:一个可执行文件，用于生成每个人对应的标识符。用法见英文文档。
- test1.sh:这是test1对应的shell脚本，在我们考虑好了如何进行攻击后，可以将要攻击的16进制数写入input1.txt，test1.sh将会自动帮你构造字符文档(作为输入)，并运行bufbomb，将运行结果写入result1.txt，同时还会帮你判断该次攻击是否成功。请仔细阅读脚本内容，注意运行脚本需要的当前目录。在进行实验的过程中，我们可能会单独使用脚本中给的一些命令，比如说运行程序以及构造字符文档。
- input1.txt: 该文件在脚本中作为bufbomb程序的输入。
- result1.txt: 该文件在脚本中作为bufbomb程序结果的输出。

基本方法

基本方法如下

- 阅读英文说明中实验背景的说明。接下来，英文说明中的level0-4分别对应我们的五个任务。
- 用objdump对bufbomb进行反汇编，并阅读汇编代码，了解程序的逻辑。
- 尝试将你设计的攻击字符作为输入，运行可执行程序，查看运行结果。
- 用gdb调试bufbomb，通过查看运行时栈中的内容，推理栈中的排列方式。在不同次运行之间，栈的排列应该是稳定的。

你对C语言运行时的知识，将会帮助你成功完成这次实验。通过以下几个问题，你可以回顾一下学到的知识。

- 运行时，内存中分为哪几个区块？
- 运行过程中栈是如何增长的？
- 不同的数据是怎么存在栈中的？进栈和出栈的先后顺序如何？（它们是我们通过缓冲区攻击可以改变的数据）
    - 函数中分配的临时数据
    - 发生函数调用时的形参，返回值，返回地址
    - 发生函数调用时，调用者保存的寄存器，被调用者保存的寄存器

III 各任务说明

具体内容请看英文文档level0-4.

task1:smoke

构造一个攻击字符串作为bufbomb的输入，在getbuf()中造成缓冲区溢出，使得getbuf()返回时不是返回到test函数，而是转到smoke()函数执行

task2:fizz

构造一个攻击字符串作为bufbomb的输入，在getbuf()中造成缓冲区溢出，使得getbuf()返回时不是返回到test函数，而是转到`fizz(int)`函数执行,且必须要使该参数为你的cookie

task3:bang

造一个攻击字符串作为bufbomb的输入，在getbuf()中造成缓冲区溢出，使得getbuf()返回时不是返回到test函数，而是转到`bang()`函数执行,且要使得变量global_value的值为你的cookie

task4:bomb

在本任务中我们希望getbuf()结束后回到test()原本的位置，并将你的cookie作为getbuf()的返回值传给test()。为了使攻击更加具有迷惑性我们还希望saved ebp被复原，这样一来原程序就完全不会因为外部攻击而出错崩溃，也就是退出攻击后要保证栈空间还原，使test()察觉不到我们干了什么，就好像我们什么都没做一样。

task5:nitro

本任务要使用./bufbomb的-n参数。bufbomb不会再像从前哪样调用test()，而是调用testn()，testn()调用getbufn()。本任务需要使getn返回你的cookie给testn()。在本任务中，bufbomb将会读5次字符串，并每次都调用getbufn(),因此各次之间的栈地址是不一样的。

IV 思考

- 当我们写C代码时，如何避免被缓冲区溢出攻击。角度有很多，比如什么库函数可以替代Gets,或者可不可以在代码中进行判断，等等。
- 思考为什么Java或是Python从标准输入中读取的函数（各种`Reader`, `input()`）不会发生缓冲区溢出，由此问题我们可以更好地了解C语言的作用。
- 对于我们在实验中接触到的这些"规则"（比如函数调用前要先压栈保存寄存器，函数调用结束要退栈等），只是一些约定，是由编译器设计者决定的。有时候为了使得生成的机器指令更加高效，编译器常常会有一些神奇的操作。可以自行搜索了解尾递归优化（tail-call optimization），内联优化(inlining),ShrinkWrap优化。第一个优化使递归调用时无需不断地压栈，第二个优化使函数调用时不需要传递形参以及返回值，最后一个优化使函数调用可以少保存一些寄存器。

V 实验报告

要求

- 要求能对你的代码进行分析，格式不做要求，即，我们希望你能将你做题的思路写下来，至于是怎样的思路都可以，只要是你自己做的，你自己能得到的思路，写在哪里，都可以。
- 最好能以画图的方式，来展示运行时栈中的排列，并说明你的输入是如何进行攻击的。

提交

- 命名格式：`学号-姓名-bufferlab实验报告.pdf`

- 发送到邮箱 hyxt_csapp_2020@163.com
- 截止时间：2020 年 11 月 24 日 24 时。

Good Luck!