15-213，20xx年秋天
实验室分配L3：缓冲炸弹分配：
XXX，到期：XXX
最后一个可能的时间上交：XXX

哈里·博维克(bovik@cs.cmu.edu)是这项任务的负责人。

导言

此作业将帮助您详细了解IA-32调用约定和堆栈组织。 它涉及对实验室目录中的可执行文件bufbomb应用一系列缓冲区溢出攻击。

注意：在本实验室中，您将获得第一手的经验，使用常用的方法之一来开发操作系统和网络服务器中的安全弱点。 我们的目的是帮助您了解程序的运行时操作，并了解这种形式的安全弱点的性质，以便您在编写系统代码时可以避免它。 我们不容忍使用这种或任何其他形式的攻击来获得未经授权的访问任何系统资源。 有规范此类活动的刑事法规。

后勤

和往常一样，这是一个单独的项目。

我们使用gcc的-m32标志生成了实验室，因此编译器生成的所有代码都遵循IA-32规则，即使主机是x86-64系统。 这应该足以让你相信编译器可以使用它想要的任何调用约定，只要它是一致的。

发出指令

您可以通过将Web浏览器指向：获取缓冲区炸弹：

```
http://$Buflab：：SERVER_NAME：18213/
```

服务器将向浏览器返回一个名为buflab-handout.tar的tar文件。 首先将buflab-handout.tar复制到一个（受保护的）目录中，您计划在该目录中执行您的工作。 然后给出命令“tarxvfbuf-handout.tar”。 这将创建一个名为buflab-handout的目录，其中包含以下三个可执行文件：

炸弹：你要攻击的缓冲炸弹程序。 Makecookie：根据

用户ID生成一个“cookie。 hex2raw：帮助在字符串

格式之间转换的实用程序。

在下面的说明中，我们将假设您已经将这三个程序复制到受保护的本地目录中，并且正在该本地目录中执行它们。

用户和饼干

这个实验室的阶段将需要一个与每个学生略有不同的解决方案。 正确的解决方案将基于您的userid。

cookie是一个由八个十六进制数字组成的字符串，它（具有很高的概率）是用户ID唯一的。 您可以使用makecookie程序生成cookie，并以userid作为参数。 例如：

```
Unix>/MakecookieBovik
0x1005b2b7
```

在五个缓冲区攻击中的四个，你的目标是让你的cookie出现在它通常不会出现的地方。

预算编制计划

BUFBOMB程序从标准输入读取字符串。 它这样做的函数getbuf定义如下：

```
1 /* 缓冲大小为getbuf*/
2 #定义NORMAL_BUFFER_SIZE32
3
4int getbuf()
5 {
6       查布[NORMAL_BUFFER_SIZE];
7       取得(buf)；
8       返回1;
9 }
```

函数Gets类似于标准库函数GET-它从标准输入(以‘\n’或文件结束)读取字符串，并将其（连同空终止符）存储在指定的目的地。 在这个代码中，您可以看到目标是一个数组buf，它有足够的空间来容纳32个字符。

获取(和get)从输入流中获取字符串并将其存储到其目标地址(在本例中为buf)。 但是，获取()无法确定buf是否足够大以存储整个输入。 它简单地复制整个输入字符串，可能超越了在目的地分配的存储的边界。

如果用户输入到getbuf的字符串长度不超过31个字符，那么很明显getbuf将返回1，如下执行示例所示：

```
Unix>/bufbomb-u bovik类型字符串：
我喜欢15-213。达德：getbuf返回0x1
```

如果我们键入一个较长的字符串，通常会发生错误：

```
Unix>/bufbomb-u bovik
类型字符串：当你是助教时，喜欢这个类更容易。
哎哟！ 你造成了分割错误！
```

如错误消息所示，溢出缓冲区通常会导致程序状态损坏，导致内存访问错误。 您的任务是更聪明地使用您提供的字符串，以便它做更有趣的事情。 这些被称为利用字符串。

BUFBOMB使用几个不同的命令行参数：

-u userid：为指定的userid操作炸弹。 你应该总是提供这个论点有几个原因：

- 它需要将您的成功攻击提交给分级服务器。
- BUFBOMB根据用户ID和程序来确定将要使用的cookie
  马科奇。
- 我们已经在BUFBOMB中构建了特性，因此您需要使用的一些关键堆栈地址取决于userid的cookie。

打印可能的命令行参数列表。

-n：以“硝基”模式操作，如下文第4级所使用。

将您的解决方案开发字符串提交给分级服务器。

此时，您应该稍微考虑一下x86堆栈结构，并找出您将针对的堆栈的哪些条目。 您可能还想确切地思考为什么最后一个示例创建了分割错误，尽管这一点不太清楚。

您的开发字符串通常包含与打印字符的ASCII值不对应的字节值。 程序HEX2RAW可以帮助您生成这些原始字符串。 它以一个六格式字符串作为输入。 在这种格式中，每个字节值用两个十六进制数字表示。 例如，字符串

“012345”可以十六进制格式输入为“303132333435”(回想一下十进制数字x的ASCII代码是0x3x。)

您传递HEX2RAW的十六进制字符应该用空格（空格或换行符)分隔）。 我建议在你工作的时候，用换行线将你的开发字符串的不同部分分开。 HEX2RAW还支持C样式块注释，因此您可以标记开发字符串的部分。 例如：

```
bf667b3278/* 移动$0x78327b66，%EDI*/
```

一定要在开始和结束注释字符串(‘/*’， ‘*/’)，因此它们将被适当地忽略。

如果在文件explience.txt中生成一个十六进制格式的explience字符串，则可以将原始字符串应用到
以几种不同的方式购买：

1. 可以设置一系列管道通过HEX2RAW传递字符串。

   Unix>cat explain.txt|./hex2raw|./bufbomb-u bovik

2. 您可以将原始字符串存储在文件中，并使用I/O重定向将其提供给BUFBOMB：

   unix>./hex2raw<explain.txt>explain-raw.txt
   Unix>/bufbomb-u bovik<explain-raw.txt

   在从GDB内部运行BUFBOMB时，也可以使用此方法：

   Unix>GDBBufbomb
   (gdb)run-u bovik<explain-raw.txt

要点：

- 您的开发字符串不能在任何中间位置包含字节值0x0A，因为这是换行(‘\n’)的ASCII代码’)。 当获取遇到此字节时，它将假定您打算终止字符串。
- HEX2RAW期望用空格分隔两位十六进制值。 因此，如果要创建一个十六进制值为0的字节，则需要指定00。 要创建单词0xDEADBEEF，您应该传递EF是ADDEHEX2RAW。

当你正确地解决了其中一个层次时，比如说0级：

```
|./bufbomb-u bovik Userid：bovik./hex2raw<烟雾-bovik.txt
曲奇：0x1005b2b7
类型字符串：烟雾！ 你叫烟雾()瓦利德
干得好！
```

然后，您可以使用-s选项将您的解决方案提交给分级服务器：

```
。/hex2raw<foot-bovik.txt|./bufbomb-u bovik-s Userid：bovik
曲奇：0x1005b2b7
类型字符串：烟雾！ 你叫烟雾()瓦利德
将开发字符串发送到服务器以进行验证。 干得好！
```

服务器将测试您的开发字符串，以确保它真的工作，它将更新缓冲区实验室记分板页面，指示您的userid(由cookie列出的匿名性)已完成此级别。

您可以通过指向Web浏览器来查看计分板

```
http://$Buflab：：SERVER_NAME：18213/scoreboard
```

与炸弹实验室不同，在这个实验室里犯错误是没有惩罚的。 请随意用任何你喜欢的绳子在BUFBOMB上开火。 当然，你也不应该蛮干这个实验室，因为它需要比你必须做的任务更长的时间。

重要注意：您可以在任何Linux机器上处理缓冲区炸弹，但为了提交您的解决方案，您需要在以下机器之一上运行：

IN STRUCTOR：插入在buflab/src/config.h中建立的合法域名列表。

第0级：蜡烛(10pts)

函数getbuf通过具有以下C代码的函数测试在BUFBOMB中调用：

```
一个空洞测试()
{
    英特瓦尔；
    /* 把金丝雀放在堆栈上，以检测可能的腐败*/
    挥发性的局部=独特的()；

    瓦尔=盖布夫()；

    /* 检查是否损坏堆栈*/
    如果（本地！=唯一的()）{
                打印f(“破坏！ ：堆栈已损坏”)；
    }
    否则，如果(VAL==cookie){
                打印(“砰！ ：getbuf返回0x%x\n“，val)；
                验证（3）；
    其他{
```

```
17            printf(“Dud：getbuf返回0x%x\n”，val);
18        }
19 }
```

当getbuf执行其返回语句(getbuf的第5行)时，程序通常在函数测试中（在此函数的第7行)恢复执行）。 我们想改变这种行为。 在文件bufbomb中，有一个函数烟雾具有以下C代码：

```
空洞的烟雾()
{
    打印f(“烟！ 您调用了烟雾()\n”)；验证（0）；
    退出（0）；
}
```

您的任务是让BUFBOMB在getbuf执行其返回语句时执行烟雾代码，而不是返回测试。 请注意，您的剥削字符串也可能损坏堆栈中与此阶段无关的部分，但这不会造成问题，因为烟雾会导致程序直接退出。

一些建议：

- 您需要为这个级别设计开发字符串的所有信息都可以通过检查BUFBOMBUseobjdump-d的拆卸版本来确定，以获得这个组装的版本。.
- 注意字节排序。
- 您可能希望使用GDB来通过getbuf的最后几个指令来步骤程序，以确保它正在做正确的事情。
- 在getbuf的堆栈框架内放置buf取决于使用哪个版本的gcc来编译bufbomb，因此您必须阅读一些程序集才能确定其真实位置。

一级：火花机(10pts)

在文件bufbomb中，还有一个函数fizz具有以下C代码：

```
无效Fizz(intVal)
{
    如果(VAL==cookie){
                打印(“Fizz！ 您调用了fizz(0x%x)\n”，val)；validate（1）；

    其他
                printf(“Misfire：您调用了fizz(0x%x)\n”，val);
    退出
     （0）；
}
```

类似于0级，您的任务是让BUFBOMB执行fizz的代码，而不是返回到测试。 然而，在这种情况下，你必须使它看起来像你已经传递了你的cookie作为它的论点。 你怎么能这么做？

一些建议：

- 请注意，程序不会真正调用fizz——它只会执行代码。 这对您要放置cookie的堆栈的位置有重要影响。

二级：鞭炮（15磅）

更复杂的缓冲区攻击形式包括提供编码实际机器指令的字符串。 然后，利用字符串将返回指针覆盖到堆栈上这些指令的起始地址。 当调用函数(在本例中为getbuf)执行其ret指令时，程序将开始执行堆栈上的指令而不是返回。 有了这种形式的攻击，你可以让程序做几乎任何事情。 *您在堆栈上放置的代码称为开发代码。* 但是，这种攻击方式很棘手，因为必须将机器代码放到堆栈上，并将返回指针设置为该代码的开始。

在文件bufbomb中有一个函数bang具有以下C代码：

```
INT global_value=0; void

bang(int val)
{
     如果(global_value==cookie){
          打印(“砰！ *将global_value设置为0x%x\n“, global_value)；验证（2）；
     其他
          打印F(“Misfire：global_value=0x%x\n”，global_value)；退出（0）；
}
```

类似于级别0和级别1，您的任务是让BUFBOMB执行bang的代码，而不是返回测试。 然而，在此之前，必须将全局变量global_value设置为userid的cookie。 您的开发代码应该设置global_value，在堆栈上推送bang的地址，然后执行ret指令，以导致跳转到bang的代码。

一些建议：

- 您可以使用GDB获取构建开发字符串所需的信息。 在getbuf中设置一个断点并运行到这个断点。 确定global_value地址和缓冲区位置等参数。
- 手工确定指令序列的字节编码是繁琐的，容易出错。 您可以通过编写包含说明和说明的汇编代码文件来让工具完成所有工作

要放在堆栈上的数据。 用gcc-m32-c组装这个文件，用objdump-d拆卸它。您应该能够得到您将在提示符下键入的确切字节序列。 (在这篇文章的结尾，我们将简要介绍如何做到这一点。)

- 请记住，您的开发字符串取决于您的机器、编译器，甚至您的userid的cookie。 在您的指导老师指定的一台机器上完成所有的工作，并确保在到BUFBOMB的命令行中包含适当的用户ID。
- 在编写程序集代码时注意使用地址模式。 请注意，movl$0x4，%eax将值0x00000004移动到寄存器%eax；而movl0x4，%eax将内存位置0x00000004的值移动到%eax。 由于该内存位置通常未定义，第二条指令将导致segfault!
- 不要试图使用jmp或调用指令跳转到bang的代码。 这些指令使用PC相对寻址，这是非常棘手的正确设置。 相反，在堆栈上推送一个地址并使用ret指令。

三级：炸药（20磅）

我们之前的攻击都导致程序跳转到其他函数的代码，然后导致程序退出。 因此，可以接受使用破坏堆栈的字符串，覆盖保存的值。

最复杂的缓冲区溢出攻击形式导致程序执行一些开发代码，这些代码更改程序的寄存器/内存状态，但使程序返回到原始调用函数（在这种情况下进行测试）。 调用函数忽略了攻击。 但是，这种攻击方式很棘手，因为您必须：1)将机器代码获取到堆栈，2)将返回指针设置为此代码的开始，3)撤消对堆栈状态的任何损坏。

这个级别的工作是提供一个explicit字符串，它将导致getbuf返回cookie进行测试，而不是值1.您可以在测试代码中看到，这将导致程序“砰！您的开发代码应该将cookie设置为返回值，恢复任何损坏的状态，在堆栈上推送正确的返回位置，并执行ret指令以真正返回到测试。

一些建议：

- 您可以使用GDB获取构建开发字符串所需的信息。 在getbuf中设置一个断点并运行到这个断点。 确定保存的返回地址等参数。
- 手工确定指令序列的字节编码是繁琐的，容易出错。 您可以通过编写包含要放在堆栈上的指令和数据的汇编代码文件来让工具完成所有工作。 用GCC组装这个文件并用OBJDUMP拆卸它，您应该能够得到您将在提示符下键入的确切字节序列。 (在这篇文章的结尾，我们将简要介绍如何做到这一点。).
- 请记住，您的开发字符串取决于您的机器、编译器，甚至您的userid的cookie。 在您的指导老师指定的机器上完成所有的工作，并确保您在到BUFBOMB的命令行中包含适当的userid。

一旦你完成了这个层次，暂停思考你已经完成了什么。 您导致程序执行自己设计的机器代码。 您这样做的方式足够隐蔽，程序没有意识到任何事情是错误的。

四级：硝酸甘油(10pts)

请注意：您需要使用“-n”命令行标志才能运行此阶段。

从一次运行到另一次运行，特别是不同的用户，给定过程使用的确切堆栈位置将有所不同。 这种变化的一个原因是，当程序开始执行时，所有环境变量的值都放在堆栈的底部附近。 环境变量存储为字符串，需要根据其值不同数量的存储。 因此，为给定用户分配的堆栈空间取决于他或她的环境变量的设置。 在GDB下运行程序时，堆栈位置也不同，因为GDB为自己的某些状态使用堆栈空间。

在调用getbuf的代码中，我们包含了稳定堆栈的特性，因此getbuf的堆栈框架的位置将在运行之间保持一致。 这使得您可以编写一个知道buf的确切起始地址的开发字符串。 如果您试图在正常程序上使用这样的漏洞，您会发现它有时会工作，但在其他时间会导致分割错误。 因此被命名为“炸药”——一种由阿尔弗雷德·诺贝尔(Alfred Nobel)开发的炸药，含有稳定元素，使其不太容易发生意外爆炸。

对于这个级别，我们走了相反的方向，使得堆栈位置甚至比通常的位置更不稳定。 因此被命名为“硝酸甘油”——一种众所周知不稳定的炸药。

当您使用命令行标志“-n”运行BUFBOMB时，它将以“Nitro”模式运行。 程序调用的不是函数getbuf，而是稍微不同的函数getbufn：

```
/* 缓 冲 大 小 为 getbufn*/# 定 义
KABOOM_BUFFER_SIZE512
```

这个函数类似于getbuf，只是它有512个字符的缓冲区。 您将需要这个额外的空间来创建一个可靠的开发。 调用getbufn的代码首先在堆栈上分配一个随机数量的存储，这样如果在连续两次执行getbufn时对%ebp的值进行采样，您会发现它们的差异最大±240。

此外，在Nitro模式下运行时，BUFBOMB要求您提供字符串5次，它将执行getbufn5次，每个都有不同的堆栈偏移。 您的开发字符串必须使它每次返回您的cookie。

您的任务与Dynamite级别的任务相同。 再一次，这个级别的工作是提供一个漏洞字符串，它将导致getbufn返回cookie进行测试，而不是值1.您可以在测试代码中看到，这将导致程序进入“KABOOM！您的开发代码应该将cookie设置为返回值，恢复任何损坏的状态，在堆栈上推送正确的返回位置，并执行ret指令以真正返回到testn。

一些建议：

- 您可以使用程序HEX2RAW发送您的开发字符串的多个副本。 如果文件explience.txt中有一个副本，那么可以使用以下命令：

  Unix>cat explain.txt|./hex2raw-n|./bufbomb-n-u bovik

  对于getbufn的所有5次执行，必须使用相同的字符串。 否则，它将失败我们的分级服务器使用的测试代码。

- 诀窍是利用nop指令。 它是用一个字节(代码0x90)编码的)。 阅读CS：APP2e教科书第262页的“NOP雪橇”可能是有用的。

后勤说明

每当您正确地解决级别并使用-s选项时，就会对分级服务器进行切换。 在收到您的解决方案后，服务器将验证您的字符串并更新缓冲区实验室评分板网页，您可以通过指向Web浏览器查看该网页

http://$Buflab：：SERVER_NAME：18213/scoreboard

您应该确保在提交后检查此页面，以确保您的字符串已被验证。 (如果您真正解决了级别，那么字符串应该是有效的。)

请注意，每个级别都是单独分级的。 您不需要按指定的顺序进行这些操作，但您只会获得服务器接收有效消息的级别的信用。 你可以检查缓冲实验室的计分板，看看你有多远。

评分服务器使用每个阶段的最新结果创建评分板。 祝你好运，玩得开心！

生成字节代码

使用GCC作为汇编程序和OBJDUMP作为反汇编程序，可以方便地生成指令序列的字节代码。 例如，假设我们编写一个文件示例。 包含以下装配代码的S：

```
#手工生成的程序集代码示例

        推送$0xabcdef                   #将值推到堆栈上
        加      $17%，%EAX              #添加17%至%eax
        对齐4                           #以下将在4的倍数上对齐
        。long    0xfedcba98            #4字节常数
```

代码可以包含指令和数据的混合。 任何“#”字符右边的东西都是评论。

我们现在可以组装和拆卸这个文件：

Unix>GCC-m32-c示例。 s
unix>objdump-d example.o>example.d

生成的文件example.d包含以下行

```
0：68efcdab00                推       $0xabcdef
5：83c011                    加       $0x11，%EAX
8: 98                        cwtl
9：巴                        字节0xba
a：DCFE                      fdivr    %st，%st（6）
```

每一行显示一个指令。 左边的数字表示起始地址（从0开始），而‘：’字符后面的十六进制数字表示指令的字节码。 因此，我们可以看到，$0xABCDEF的指令推送有六格式字节码68efcdab00。

从地址8开始，反汇编者会感到困惑。 它试图将文件example.o中的字节解释为指令，但这些字节实际上对应于数据。 但是，请注意，如果我们从地址8开始读取4个字节，我们得到：98baDCfe。这是一个字节反转版本的数据字0xFEDCBA98。 这个字节反转表示将字节作为字符串提供的正确方法，因为一个小Endian机器首先列出了最不重要的字节。

最后，我们可以将代码的字节序列读出为：

```
68ef cd ab0083c01198ba dc fe
```

. 然后，这个字符串可以通过HEX2RAW传递，以生成一个适当的输入字符串，我们可以给BUFBOMB，另外，我们可以编辑example.d，看起来像这样：

```
68ef cd ab00/* 推       $0xabcdef*/83c011/*
加                      $0x11，%EAX*/
98
巴DCFe
```

这也是一个有效的输入，我们可以通过HEX2RAW发送到BUFBOMB。