15-213年，秋季20xx

实验室任务L3：缓冲区炸弹

指定：XXX，到期：XXX

最后一次上班时间：XXX

哈里·博维克(bovik@cs。cmu.他是这项任务的负责人。

介绍

此任务将帮助您详细了解IA-32调用约定和堆栈组织。它涉及到对实验室目录中的一个可执行文件应用一系列的缓冲区溢出攻击。

注意：在本实验室中，您将获得使用通常用于利用操作系统和网络服务器中的安全弱点的常用方法之一的第一手经验。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本质，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攻击或任何其他形式的攻击来获得对任何系统资源的未经授权的访问。这类活动有刑事法规。

物流

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，即使主机是一个x86-64系统。这应该足以让您相信，编译器可以使用它想要的任何调用约定，只要它是一致的。

分发说明

你可以通过指向你的网络浏览器来获得你的缓冲炸弹：

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

服务器将返回一个称为buflab讲义的tar文件。tar到你的浏览器。从复制buflab讲义开始。标记您计划在其中执行工作的（受保护的）目录。然后给出命令“tarxvf错误讲义”。焦油这将创建一个名为buflab-讲义的目录，其中包含以下三个可执行文件：

你要攻击的缓冲炸弹计划。

制作饼干：根据你的用户名生成一个“饼干”。

十六进制2raw：一个可以帮助在字符串格式之间进行转换的实用程序。

在下面的说明中，我们将假设您已经将这三个程序复制到一个受保护的本地目录中，并且您正在在该本地目录中执行它们。

用户和cookie

这个实验室的各个阶段需要每个学生的解决方案略有不同。正确的解决方案将基于您的用户id。

cookie是由8个十六进制数字组成的字符串，它（是用户id唯一的高概率）。您可以使用制造烹饪程序生成您的用户名作为参数。例如：

unix>。/makecookie波维克
0x1005b2b7

在你的五次缓冲攻击中，有四次，你的目标将是让你的饼干出现在它通常不会出现的地方。

炸弹计划

BUFBOMB程序从标准输入中读取一个字符串。它使用下面定义的函数getbuf来这样做：

```
1 /*getbuf的缓冲区大小*/
2#定义NORMAL_BUFFER_SIZE 32
3
4 int getbuf() 5
{
6 char buf[NORMAL_BUFFER_SIZE];
7得到（buf）；
8返回1；
9 }
```

函数Gets与标准库函数类似—它从标准输入（以“\n”或文件结尾）读取字符串，并将其存储在指定的目的地（连同空终止符）。在这段代码中，您可以看到目标是一个数组buf，有足够的空间容纳32个字符。

获取（并获取）从输入流中获取一个字符串，并将其存储到其目标地址（在这种情况下为buf)。但是，Gets（）无法确定buf是否足够大来存储整个输入。它只是复制整个输入字符串，可能会覆盖在目标上分配的存储的边界。

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

```
unix>./bufbom-u波维克
我喜欢15-213。
Dud： getbuf返回0x1
```

如果我们输入一个较长的字符串，通常会出现错误：

```
unix>./bufbom-u波维克
输入字符串：当你是一个助教时，你更容易喜欢这个类。
哎哟！：你造成了一个分割故障！
```

如错误消息所示，覆盖缓冲区通常会导致程序状态损坏，从而导致内存访问错误。你的任务是更聪明地使用你提供的琴弦，这样
它能做一些更多有趣的事情。*这些都被称为利用字符串。*

BUFBOMB采用了几个不同的命令行参数：

-u用户名：为指定的用户名操作炸弹。你应该总是提供这几个论点
理由

- o需要将您成功的攻击提交给分级服务器。
- o BUFBOMB根据你的用户名决定你将使用的饼干，以及程序制造饼干。
- o我们已经在BUFBOMB中构建了一些特性，这样你需要使用的一些关键堆栈地址就取决于你的用户id的饼干。

-h：打印可能的命令行参数的列表。

-n：在“硝基”模式下操作，如下面第4级中使用。

-s：提交您的解决方案利用字符串到分级服务器。

在这一点上，您应该考虑一下x86堆栈结构，并找出您将针对的堆栈条目。您可能还想思考最后一个示例创建分割错误的确切原因，尽管这不太清楚。

您的利用字符串通常将包含与打印字符的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。*它以一个六边制格式的字符串作为输入*。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”可以以十六进制格式输入为“30 31 32 33 34 35”。（回想一下，十进制数字x的ASCII代码是0x3x。）

传递给HEX2RAW的十六进制字符应该用空格（空格或换行）分隔。我建议您在工作时用换行符分隔利用字符串的不同部分。HEX2RAW还支持c风格的块注释，因此您可以标记利用字符串的部分。例如：

```
bf 66 7b 32 78 /* mov $0x78327b66,%edi*/
```

一定要在开始和结束的注释字符串周围都留出空间(‘/*’， ‘*/‘)，因此它们将被正确地忽略。

如果您在文件中生成一个十六进制格式的利用字符串。你可以用几种不同的方式将原始字符串应用到bufbonb上：

1. 您可以设置一系列的管道来使字符串通过HEX2RAW。

   unix>猫开发。txt |。/十六进制2原始|。你

2. 您可以将原始字符串存储在一个文件中，并使用I/O重定向将其提供给BUFBOMB：

   unix>./hex2原始<利用。txt >开发原始。txt
   unix>./你<利用-原始。txt

   这种方法也可以用于从GDB内运行错误炸弹：

   unix> gdb bufbomb
   （gdb）运行-u bovik <利用-原始。txt

要点：

- o您的利用字符串不能在任何中间位置包含字节值0x0A，因为这是换行符（‘\n’）的ASCII代码。当获取遇到此字节时，它将假定您打算终止该字符串。
- o HEX2RAW期望用空格分隔的两位十六进制值。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，你应该通过EF BE AD DE HEX2RAW。

当你正确地解决了其中一个级别时，比如说第0级：

```
../十六进制2原始<烟-bovik。txt |        ../bufbom-ubovik
Userid：博维克
饼干： 0x1005b2b7
类型字符串：烟雾！：你打电话来了        吸烟
有效的
```

干得好！

然后，您可以使用-s选项将解决方案提交到分级服务器：

```
/十六进制2原始<烟-bovik。txt|./炸弹炸弹
Userid：博维克
饼干： 0x1005b2b7
类型字符串：烟雾！：你叫烟（）
有效的
已将利用漏洞的字符串发送到要验证的服务器。
干得好！
```

服务器将测试您的利用字符串，以确保它真的工作，它将更新缓冲实验室记分牌页面，表明您的用户id（由您的cookie列出的匿名）已经完成这个级别。

你可以通过指向你的网络浏览器

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

与炸弹实验室不同，在这个实验室里犯错不会受到惩罚。请随意用任何你喜欢的绳子发射炸弹。当然，你也不应该强行强迫这个实验室，因为这需要比你完成任务的时间更长。

重要注意：您可以在任何Linux机器上使用缓冲炸弹，但为了提交您的解决方案，您将需要在以下机器之一上运行：

讲师：插入您所使用的合法域名的列表
在buflab/src/配置中建立。h.

0级：烛台(10分）

该函数getbuf在BUFBOMB中通过一个函数测试被调用，其C代码如下：

```
1空测试（）
2 {
3 int val;
4         /*将金丝雀放在堆栈上，以检测可能的损坏*/
5个易稳定的int本地=唯一的（）；
6
7 val = getbuf(); 8
9       /*检查堆栈是否损坏*/
10如果（本地！=唯一的（））{
11 printf("Sabotaged!：堆栈已损坏”
如果（val == cookie）{
14 printf("Boom!： getbuf返回0x%x\n“，val);
15验证(3)；16 }其他{
```

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

当getbuf执行它的返回语句（getbuf的第5行）时，程序通常会在函数测试中恢复执行（在这个函数的第7行）。我们想改变这种行为。在文件bufbomb中，有一个函数包含以下C代码：

```
空烟（）
{
     printf("Smoke!：你叫烟（）\n”）；
     验证(0)；
     退出(0)；
}
```

您的任务是在betbuf执行其返回语句时，让bufbomm执行烟雾的代码，而不是返回测试。请注意，您的利用漏洞字符串也可能损坏与此阶段不直接相关的堆栈部分，但这不会导致问题，因为烟雾会导致程序直接退出。

一些建议：

- o为这个级别设计漏洞字符串所需的所有信息都可以通过检查BUFBOMB的分解版本来确定。使用 objdump -d来获得这个伪装过的版本。
- o请注意字节排序。
- o你可能想使用GDB来完成程序的最后几条指令，以确保它在做正确的事情。
- o在getbuf的堆栈帧中放置buf的位置取决于使用哪个版本的GCC来编译bufbomb，所以你必须读取一些汇编来找出它的真实位置。

1级：火花器(10分）

在文件bufbomb中还有一个函数fizz具有以下C代码：

```
空fizz（int val）
{
     如果（val == cookie）{
                    printf("Fizz!：你叫fizz（0x%x）\n”，val)；
                    验证(1)；

     }其他
                    你叫fizz（0x%x）\n”，val)；
     退出(0)；
}
```

与第0级类似，您的任务是让bufbomm执行fizz代码，而不是返回测试。然而，在这种情况下，你必须让它看起来嘶嘶作响，就好像你已经把你的饼干作为它的论点。你怎么做？

一些建议：

- o注意，程序不会真正调用fizz——它会简单地执行它的代码。这对于你想要把饼干放在堆栈上的位置有重要的意义。

第2级：爆竹(15分）

一种更复杂的缓冲区攻击形式涉及提供一个编码实际机器指令的字符串。然后，利用漏洞的字符串会在堆栈上用这些指令的起始地址覆盖返回指针。当调用函数（在本例中是getbuf）执行它的ret指令时，程序将开始在堆栈上执行指令，而不是返回。使用这种形式的攻击，你可以让程序做几乎任何事情。*您放置在堆栈上的代码称为利用代码*。但是，这种攻击方式很棘手，因为您必须将机器代码获取到堆栈中，并将返回指针设置为此代码的开头。

在文件bufbomb中有一个函数，其中有以下C代码：

```
int全局值=0;

空爆（int val）
{
     如果（global_value == cookie）{
          printf("Bang!：您将全局值设置为0x%x\n“，global_value);
          验证(2);
     }其他
          Printf（“失火：全局值=0x%x\n”，全局值）；
     退出(0)；
}
```

类似于级别0和级别1，您的任务是让bufbonb来执行爆炸的代码，而不是返回到测试中。但是，在此之前，您必须设置全局变量global_valueto，即您的用户名的cookie。您的利用代码应该设置全局值，在堆栈上推送bang的地址，然后执行一个ret指令，以导致跳转到要执行bang的代码中。

一些建议：

- o你可以使用GDB来获取构建利用漏洞字符串所需的信息。在getbuf中设置一个断点，并运行到此断点。确定一些参数，如全局值的地址和缓冲区的位置。
- o手工确定指令序列的字节编码比较繁琐且容易出错。您可以让工具通过编写一个包含指令和

您想放在堆栈上的数据。用gcc -m32 -c组装这个文件，然后用objdump -d分解它。您应该能够获得您将在提示符处键入的确切字节序列。（在本文的结尾部分，会有一个关于如何做到这一点的简短例子。）

- o请记住，你的漏洞字符串取决于你的机器，你的编译器，甚至你的用户名的cookie。在你的老师指定的机器上做你所有的工作，并确保你在命令行中包含适当的用户id。
- o在编写汇编代码时，请注意您对地址模式的使用情况。注意，movl $0x4，%eax将值0x00000004移到寄存器%eax；而movl 0x4，%eax将内存位置0x00000004处的值移到%eax。由于该内存位置通常是未定义的，因此第二条指令将导致分段故障！
- o不要尝试使用jmp或调用指令来跳转到执行bang的代码中。这些指令使用pc相对寻址，这是非常棘手的设置正确。相反，在堆栈上推送一个地址并使用ret指令。

第3级：炸药(20分）

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

最复杂的缓冲区溢出攻击形式会导致程序执行一些漏洞代码，改变程序的寄存器/内存状态，但使程序返回原始调用函数（在本例中测试）。调用函数无视这次攻击。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上，2)将返回指针设置为这段代码的开头，3)撤销对堆栈状态造成的任何损坏。

这个级别的工作是提供一个漏洞字符串，该字符串将导致getbuf将您的cookie返回到测试中，而不是值1。您可以在测试代码中看到，这将导致程序进入“启动！”."您的利用代码应该将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，并执行一个ret指令以真正返回到测试中。

一些建议：

- o你可以使用GDB来获取构建利用漏洞字符串所需的信息。在getbuf中设置一个断点，并运行到此断点。确定诸如已保存的返回地址等参数。
- o手工确定指令序列的字节编码比较繁琐且容易出错。您可以让工具通过编写一个包含要放在堆栈上的指令和数据的汇编代码文件来完成所有的工作。用GCC组装这个文件，然后用OBJDUMP分解它。您应该能够获得您将在提示符处键入的确切字节序列。（在本文的结尾部分，会有一个关于如何做到这一点的简短例子。）
- o请记住，你的漏洞字符串取决于你的机器，你的编译器，甚至你的用户名的cookie。在你的老师指定的机器上完成你所有的工作，并确保你在命令行中包含了适当的用户id。

一旦你完成了这个关卡，停下来思考一下你已经完成了什么。您使一个程序执行您自己设计的机器代码。您已经以一种足够隐秘的方式这样做了，以至于程序没有意识到有任何问题。

第4级：硝酸甘油(10分）

请注意：您需要使用“-n”命令行标志来运行这个阶段。

从一次运行到另一次运行，特别是由不同的用户运行，给定过程所使用的确切堆栈位置将会有所不同。造成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被放置在堆栈的底部附近。环境变量以字符串的形式存储，根据它们的值需要不同的存储量。因此，为给定用户分配的堆栈空间取决于其环境变量的设置。当在GDB下运行程序时，堆栈位置也不同，因为GDB为自己的一些状态使用堆栈空间。

在调用getbuf的代码中，我们合并了稳定堆栈的特性，这样getbuf的堆栈帧的位置在运行之间将是一致的。这使得您可以编写一个知道buf的确切起始地址的利用字符串。如果你试图在一个普通的程序上使用这样的漏洞，你会发现它有时可以工作，但在其他时候它会导致分割故障。因此得名“炸药”——由阿尔弗雷德·诺贝尔发明的一种炸药，它含有稳定元素，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，使得堆栈位置比通常更不稳定。因此得名“硝酸甘油”——一种出了名的不稳定的炸药。

当你用命令行标志“-n”运行BUFBOMB时，它将在“硝基”模式下运行。该程序不是调用函数getbuf，而是调用一个稍微不同的函数getbufn：

```
/*getbufn的缓冲区大小*/
#定义KABOOM_BUFFER_SIZE 512
```

这个函数类似于getbuf，只是它有一个包含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利用。调用getbufn的代码首先在堆栈上分配随机数量的存储量，这样，如果您在连续两次执行getbufn期间采样%ebp的值，您会发现它们的差异高达±240。

此外，当在硝基模式下运行时，BUFBOMB要求您提供您的字符串5次，它将执行getbufn 5次，每个都有不同的堆栈偏移量。您的利用字符串必须使它每次返回您的cookie。

您的任务与炸药级别的任务相同。同样，这个级别的工作是提供一个漏洞字符串，它将使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！”."您的利用代码应该将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，并执行一个ret指令以真正返回到testn。

一些建议：

o您可以使用程序HEX2RAW来发送您的利用字符串的多个副本。如果您的文件中有一个副本的漏洞利用。然后，您可以使用以下命令：

unix>猫开发。txt |。/十六进制2原始|。

您必须对getbufn的所有5个执行使用相同的字符串。否则，它将无法通过我们的分级服务器所使用的测试代码。

o的诀窍是利用nop指令。它用一个字节编码（代码0x90)。在CS： APP2e教科书的第262页上阅读关于“nop雪橇”可能会很有用。

后勤笔记

当您正确地解决一个级别并使用-s选项时，分级服务器就会发生切换。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记分牌网页，您可以通过指向您的Web浏览器来查看

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。（如果您真的解决了这个级别，那么您的字符串应该是有效的。）

请注意，每个级别都是单独分级的。您不需要按照指定的顺序执行它们，但您将只获得服务器接收到有效消息的级别的积分。你可以查看缓冲实验室的记分牌，看看你已经走了多远。

评分服务器通过使用每个阶段的最新结果来创建记分牌。

祝你好运，玩得开心吧！

生成字节码

将GCC作为汇编程序，将OBJDUMP作为反汇编程序，可以方便地为指令序列生成字节码。例如，假设我们编写了一个文件示例。S，其中包含以下装配代码：

```
#手工生成的汇编代码的示例

        推0美元xabcdef          #将值推到堆栈上
        增加17美元，%eax        #添加17到%eax
        .对齐4                  #下面的内容将以4的倍数对齐
        .长0xfedcba98           #A4字节常数
```

该代码可以包含指令和数据的混合物。任何在“#”字符右边的任何东西都是一个注释。

我们现在可以组装和拆卸这个文件：

unix> gcc -m32 -c示例。S
unix>数据库的例子。o>示例.d

生成的文件示例。D包含以下几行

```
0:68 ef cd ab 00             推       $0xabcdef
5: 83 c0 11                  添加     $0x11,%eax
8: 98                        cwtl
9: ba                        .字节   0xba
a: dc fe                     fdivr    %st,%st (6)
```

每一行都显示一条指令。左边的数字表示起始地址（以0开始），而“：”字符后面的十六进制数字表示该指令的字节码。因此，我们可以看到指令推送$0xABCDEFhas十六进制格式的字节码68 ef cd ab 00。

从地址8开始，拆解汇编器会被混淆。它试图解释文件示例中的字节。oas指令，但这些字节实际上对应于数据。但是，请注意，如果我们读取从地址8开始的4个字节，我们会得到： 98 ba dc fe。这是数据字0xFEDCBA98的字节反转版本。这个字节反转表示提供字节作为字符串的正确方式，因为一个小的环境机器首先列出最不重要的字节。

最后，我们可以读取我们的代码的字节序列为：

```
68 ef cd ab 00 83 c0 11 98 ba dc fe
```

然后，这个字符串可以通过HEX2RAW生成一个适当的输入字符串，我们可以给bufbonb。或者，我们也可以编辑示例。d要这样看：

```
68 ef cd ab 00 /*推$0xabcdef*/
83 c0 11 /*添加$0x11，%eax*/ 98
ba dc fe
```

这也是一个有效的输入，我们可以通过HEX2RAW之前发送到buf炸弹。