15-213，20XX年秋季

攻击实验室：了解缓冲区溢出错误分配：星期二，九月29

截止时间:10月8日星期四，美国东部时间晚上11:59

最后可能的上交时间：星期日，10月11日，美国东部时间晚上11点59分

1 介绍

此任务涉及对具有不同安全漏洞的两个程序生成总共五个攻击。您将从本实验中获得的成果包括:

- 您将了解到，当程序不能很好地防止缓冲区溢出时，攻击者可以通过不同的方式利用安全漏洞。
- 通过本文，您将更好地了解如何编写更安全的程序，以及编译器和操作系统提供的一些功能，以使程序不易受到攻击。
- 您将对x86-64机器码的堆栈和参数传递机制有更深入的了解。
- 您将更深入地了解x86-64指令的编码方式。
- 您将获得更多使用调试工具（如GDB和objdump）的经验。

注意：在本实验中，您将获得利用操作系统和网络服务器中的安全漏洞的方法的第一手经验。我们的目的是帮助您了解程序的运行时操作，并了解这些安全漏洞的本质，以便您在编写系统代码时可以避免这些漏洞。我们不会容忍使用任何其他形式的攻击来获取对任何系统资源的未经授权的访问。

您需要学习《CS：app3e》一书的第3.10.3节和第3.1 0.4节，作为本实验的参考资料。

2 物流

像往常一样，这是一个单独的项目。您将为自定义生成的目标程序生成攻击。

2.1 正在获取文件

您可以通过将Web浏览器指向以下位置来获取文件:

```
HTTP：//$AttackLab：:服务器_名称:15513/
```

讲师：$AttackLab：:Server_名称是运行AttackLab服务器的计算机。您可以在 AttackLab/AttackLab.pm和AttackLab/SRC/build/DriverHDRS.H中定义它

服务器将构建您的文件，并在名为targetk.tar的tar文件中将它们返回到您的浏览器，其中K是您的目标程序的唯一编号。

注意：构建和下载目标需要几秒钟的时间，所以请耐心等待。

将targetk.tar文件保存在您计划在其中执行工作的（受保护的）Linux目录中。然后给出命令：tar-xvf targetk.tar。这将提取包含下面描述的文件的目录targetk。

您应该只下载一组文件。如果出于某种原因，您下载了多个目标，请选择一个目标并删除其余目标。

警告：如果您使用WinZip等实用程序在PC上展开Targetk.tar，或让浏览器执行提取操作，则可能会重置可执行文件的权限位。

TargetK中的文件包括:

README.TXT：描述目录内容的文件

ctarget：易受代码注入攻击的可执行程序

rTarget：易受面向返回的编程攻击的可执行程序

cookie.txt:8位十六进制代码，用作攻击中的唯一标识符。

Farm.C：目标的“ Gadget Farm ”的源代码，您将使用它来生成面向返回的编程攻击。

hex2raw：用于生成攻击字符串的实用程序。

在以下说明中，我们假定您已将文件复制到受保护的本地目录中，并且正在执行该本地目录中的程序。

2.2 重要观点

以下是关于本实验有效解决方案的一些重要规则的摘要。当您第一次阅读此文档时，这些要点将没有多大意义。一旦你开始，它们在这里作为规则的中心参考。

- 您必须在与生成目标的计算机类似的计算机上执行分配。
- 您的解决方案不得使用攻击来规避程序中的验证代码。具体来说，任何合并到攻击字符串中供RET指令使用的地址都应该是以下目的地之一:
  - 功能Touch1、Touch2或Touch3的地址。
  - 注入代码的地址
  - 小工具农场中您的一个小工具的地址。
- 您只能从文件rtarget构造Gadget，其地址范围介于函数Start_Farm和End_Farm之间。

3 目标程序

ctarget和rtarget都从标准输入中读取字符串。它们使用函数getbuf来执行此操作。
定义如下:

```
1 未签名的GETBUF（）。
2 {
3     CHAR BUF[缓冲区_大小];
4     获取（buf）；
5     返回1；
6 }
```

函数GETS类似于标准库函数GETS—它从标准输入读取字符串（以'\n '或文件结尾终止），并将其（与NULL终止符一起）存储在指定的目标。在这段代码中，您可以看到目标是一个数组buf，声明为HAVING BUFFER_SIZE字节。在生成目标时，缓冲区_大小是特定于程序版本的编译时常量。

函数获取（）和获取（）无法确定它们的目标缓冲区是否足够大，可以存储它们读取的字符串。它们只是复制字节序列，可能会超出在目的地分配的存储边界。

如果用户键入并由getbuf读取的字符串足够短，则很明显getbuf将返回1，如以下执行示例所示:

```
UNIX amp; gt; ./C目标
```

```
Cookie: 0x1a7dd803
类型字符串：保持简短！
没有漏洞。GetBuf返回0x1正常返回
```

通常，如果键入长字符串，则会发生错误:

```
UNIX amp; gt; ./C目标
Cookie: 0x1a7dd803
类型字符串：这不是一个非常有趣的字符串，但它具有..属性。
哎哟！：您导致了分段错误！
祝你下次好运
```

（请注意，显示的cookie的值将与您的不同。）程序rtarget将具有相同的行为。如错误消息所示，溢出缓冲区通常会导致程序状态被破坏，从而导致内存访问错误。你的任务是更聪明地处理你喂的绳子。
CTARGET和RTARGET，这样它们可以做更多有趣的事情。这些字符串称为漏洞利用字符串。

ctarget和rtarget都有几个不同的命令行参数:

-H：打印可能的命令行参数列表

-Q：不要将结果发送到评分服务器

-I文件：从文件提供输入，而不是从标准输入提供输入

您的漏洞字符串通常包含与用于打印字符的ASCII值不对应的字节值。程序hex2raw将使您能够生成这些原始字符串。有关如何使用HEX2RAW的详细信息，请参阅附录A。

重要事项:

- 您的漏洞字符串不得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Gets遇到这个字节时，它将假定您打算终止该字符串。
- HEX2RAW需要由一个或多个空格分隔的两位十六进制值。因此，如果要创建十六进制值为0的字节，则需要将其写为00。要创建单词0xDeadBeef，应将“ ef be ad de ”传递给hex2raw（注意little-endian字节排序所需的反转）。

当您正确解决其中一个级别时，您的目标程序将自动向评分服务器发送通知。例如:

```
UNIX amp; gt; ./hex2raw&lt; 2.ctarget.Ltxt|./ctarget
Cookie: 0x1a7dd803
键入字符串：Touch2！：您调用了Touch2（0x1A7DD803）级别2的有
效解决方案，目标为cTarget
通过：将漏洞字符串发送到服务器以进行验证。干得好！
```

阶段	项目	水平	方法	功能	分
1 2 3	C目标 目标目标	1 2 3	CI 慈慈	触摸1 触摸2触摸3	10 25 25
4 5	R目标 R目标	2 3	罗普 罗普	触摸2 触摸3	35 5

CI: 代码注入
钻速: 面向返回的程序设计

图1:攻击实验阶段总结

服务器将测试您的漏洞字符串，以确保它确实有效，并将更新AttackLab记分板页面，表明您的用户ID（按匿名目标编号列出）已完成此阶段。

您可以通过将Web浏览器指向来查看记分板

```
HTTP：//$AttackLab：:服务器_名称:15513/Scoreboard
```

与炸弹实验室不同的是，在这个实验室里犯错不会受到惩罚。请随意向cTarget开火。
和rtarget以及您喜欢的任何字符串。

重要说明：您可以在任何Linux计算机上处理您的解决方案，但为了提交您的解决方案，您需要在以下计算机之一上运行:

讲师：插入您在buflab/SRC/config.C中建立的合法域名列表。

图1总结了实验的五个阶段。可以看出，前三个涉及对CTarget的代码注入（CI）攻击，而后两个涉及对RTarget的返回导向编程（ROP）攻击。

4 第一部分：代码注入攻击

在前三个阶段，您的漏洞字符串将攻击cTarget。这个程序是以这样一种方式设置的，即堆栈位置从一次运行到下一次运行是一致的，因此堆栈上的数据可以被视为可执行代码。这些功能使程序容易受到攻击，其中利用字符串包含可执行代码的字节编码。

4.1 1级

对于第1阶段，您将不会注入新代码。相反，您的漏洞字符串将重定向程序以执行现有过程。

函数getbuf在CTarget中由具有以下C代码的函数测试调用:

```
1 空隙试验（）。
2 {
3     int val;
4     val = getbuf();
5     printf（&quot；无漏洞。GetBuf返回0x%X\n&quot；，Val）；
6 }
```

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恢复执行（在此函数的第5行）。我们想改变这种行为。在文件CTARGET内，存在具有以下C表示的函数touch1的代码:

```
1 无效Touch1（）。
2 {
3     vlevel = 1;        /*验证方案部分*/
4     printf（&quot；touch1！：（）调用了touch1\n&quot；）；
5     验证（1）；
6     退出（0）；
7 }
```

您的任务是让ctarget在getbuf执行其return语句时执行touch1的代码，而不是返回到test。请注意，您的漏洞字符串也可能会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Touch1会导致程序直接退出。一些建议:

- 通过检查cTarget的反汇编版本，可以确定设计此级别的漏洞利用字符串所需的所有信息。使用objdump-d获取此反汇编版本。
    - 其思想是定位touch1的起始地址的字节表示，以便getbuf的代码末尾的ret指令将控制转移到touch1。
- 请注意字节排序。
- 您可能希望使用GDB通过getbuf的最后几条指令来逐步执行程序，以确保它正在执行正确的操作。
- buf在getbuf的堆栈帧中的位置取决于编译时常量buffer_size的值，以及GCC使用的分配策略。您需要检查反汇编的代码以确定其位置。

4.2 2级

第2阶段涉及将少量代码作为漏洞利用字符串的一部分注入。

在文件CTARGET内，存在具有以下C表示的函数touch2的代码:

```
1无效触摸2（无符号Val）
```

```
2 {
3     vlevel = 2;        /* 部分验证方案*/
4      如果(Val==cookie){
5           printf(&quot; 您调用了touch2(0x%.8x)\n&quot;,
                touch2!: Val);
6          验证(2);
7     }否则{
8      printf(&quot;发射失 您调用了touch2(0x%.8x)\n&quot;,
                    败: Val);
9         失败(2);
10    }
11    退出(0);
12 }
```

您的任务是让CTarget执行Touch2的代码，而不是返回测试。但是，在这种情况下，您必须使它看起来像是touch2，就好像您已经将cookie作为其参数传递一样。

一些建议:

- 您将希望以这样一种方式定位插入代码地址的字节表示:
  getbuf代码末尾的ret指令将把控制权转移给它。
- 回想一下，函数的第一个参数在寄存器%RDI中传递。
- 注入的代码应该将寄存器设置为cookie，然后使用RET指令将控制转移到Touch2中的第一条指令。
- 不要尝试在您的漏洞代码中使用JMP或CALL指令。这些指令的目的地地址的编码难以公式化。使用RET指令进行所有控制转移，即使您没有从呼叫中返回。
- 有关如何使用工具生成指令序列的字节级表示，请参阅附录B中的讨论。

4.3 第3级

阶段3还涉及代码注入攻击，但将字符串作为参数传递。

在文件CTARGET中，函数hexmatch和touch3的代码具有以下C表示:

```
1 /*比较字符串与无符号值的十六进制表示*/
2 INT HEXMATCH（无符号值，字符*SVAL）
3 {
4     充电CBUF[110];
5     /*使检查字符串的位置不可预测*/
6     char*s=cbuf+随机（）%100;
7     sprintf（s，&quot;%.8x&quot;，Val）;
8     返回strncmp（sval，s，9）==0;
9 }
```

```
10
11 无效触摸3（字符*SVAL）
12 {
13      vlevel = 3;          /*验证方案部分*/
14     如果（hexmatch（cookie，  斯瓦尔））{
15    printf（&quot；touch3！：  您调用了touch3（\&quot；%s\&quot；）
                                \n&quot；，sval）；
16           验证（3）；
17     }否则{
18     printf（&quot；发射失败：  您调用了touch3（\&quot；%s\&quot；）
                                \n&quot；，sval）；
19          失败（3）；
20     }
21     退出（0）；
22 }
```

您的任务是让CTarget执行Touch3的代码，而不是返回测试。您必须使它看起来像是Touch3，就好像您已经传递了cookie的字符串表示作为其参数。

一些建议:

- 您需要在漏洞字符串中包含cookie的字符串表示。该字符串应由八个十六进制数字（从最高位到最低位排序）组成，不带前导“ 0x ”。
- 回想一下，字符串在C语言中表示为一个字节序列，后跟一个值为0的字节。在任何Linux机器上键入“ man ASCII ”以查看所需字符的字节表示。
- 插入的代码应将寄存器%RDI设置为此字符串的地址。
- 当调用函数hexmatch和strncmp时，它们将数据压入堆栈，覆盖保存getbuf使用的缓冲区的内存部分。因此，您需要注意放置cookie的字符串表示的位置。

5 第二部分：面向回报的程序设计

对程序rtarget执行代码注入攻击比对ctarget执行代码注入攻击要困难得多，因为它使用两种技术来阻止此类攻击:

- 它使用随机化，以便堆栈位置从一次运行到另一次运行不同。这使得无法确定您的注入代码将位于何处。
- 它将保存堆栈的内存部分标记为不可执行，因此即使您可以将程序计数器设置为插入代码的开头，程序也会因分段错误而失败。

幸运的是，聪明的人已经设计出了通过执行现有代码而不是注入新代码来在程序中完成有用工作的策略。最常见的形式是面向返回的编程（ROP）[1，2]。ROP的策略是识别现有程序中的字节序列，这些字节序列由一个或多个指令组成，后面跟着指令RET。这样的段称为

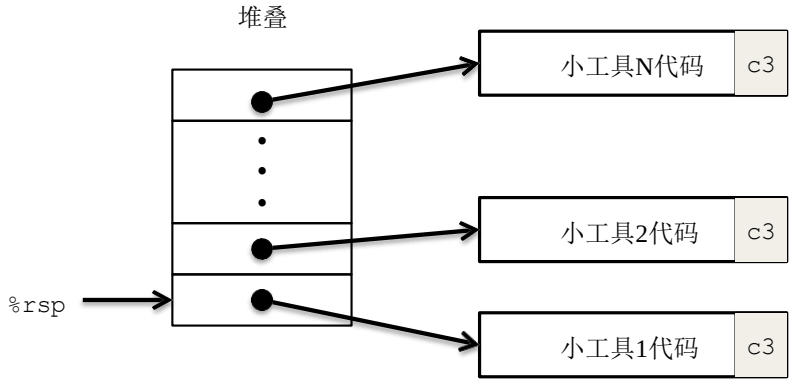

图2:设置小工具的执行顺序。字节值0xC3对RET指令进行编码。

小玩意。图2说明了如何设置堆栈以执行N个小工具的序列。在此图中，堆栈包含一系列小工具地址。每个小工具由一系列指令字节组成，最后一个是0xC3，编码RET指令。当程序执行以此配置开始的RET指令时，它将启动一系列小工具执行，每个小工具末尾的RET指令将使程序跳转到下一个小工具的开头。

小工具可以利用与编译器生成的汇编语言语句相对应的代码，尤其是函数末尾的代码。在实践中，可能有一些这种形式的有用的小工具，但不足以实现许多重要的操作。例如，编译后的函数不太可能将popq%RDI作为ret之前的最后一条指令。幸运的是，使用面向字节的指令集（如x86-64），通常可以通过从指令字节序列的其他部分提取模式来找到小工具。

例如，rtarget的一个版本包含为以下C函数生成的代码:

```
VOID SETVAL_210（（无符号*P）
{
    *p = 3347663060U;
}
```

此函数用于攻击系统的可能性似乎非常小。但是，这个函数的反汇编机器码显示了一个有趣的字节序列:

```
0000000000400F15&lt; SETVAL_210&gt; :
  400f15:       c7  07  d4  48  89 c7       莫夫尔雷特克    $0xc78948d4,(%rdi)
  400f1b:       c3
```

字节序列4889 C7对指令MOVQ%RAX，%RDI进行编码。（有关有用的MOVQ指令的编码，请参见图3A。）此序列后面是字节值C3，它对RET指令进行编码。函数从地址0x400F15开始，序列从函数的第四个字节开始。因此，此代码包含一个起始地址为0x400F18的小工具，该小工具将把寄存器%RAX中的64位值复制到寄存器%RDI。

rtarget的代码包含许多类似于上面显示的setval_210函数的函数，这些函数位于我们称为GadgetFarm的区域中。您的工作将是在小工具场中识别有用的小工具，并使用这些小工具执行类似于您在第2阶段和第3阶段所执行的攻击。

重要提示：小工具场由副本中的“开始_场”和“结束_场”功能来划分。
R目标。不要尝试从程序代码的其他部分构造小工具。

5.1 2级

对于第4阶段，您将重复第2阶段的攻击，但使用您的小工具场中的小工具在程序rtarget上执行此操作。您可以使用由以下指令类型组成的小工具构建解决方案，并且仅使用前八个x86-64寄存器（%RAX–%RDI）。

MOVQ：这些代码如图3A所示。

POPQ：这些代码如图3B所示。

RET：此指令由单字节0xC3编码。

NOP：此指令（发音为“ no op ”，是“ no operation ”的缩写）由单字节0x90编码。其唯一的作用是使程序计数器递增1。

一些建议：

- 您需要的所有小工具都可以在rtarget的代码区域中找到，该区域由函数start_farm和mid_farm划分。
- 您只需使用两个小工具即可完成此攻击。
- 当小工具使用POPQ指令时，它将从堆栈中弹出数据。因此，您的漏洞利用字符串将包含小工具地址和数据的组合。

5.2 第3级

在你开始第五阶段之前，停下来想一想到目前为止你已经完成了什么。在第2阶段和第3阶段，您使程序执行自己设计的机器码。如果ctarget是一个网络服务器，您可以将自己的代码注入到远程计算机中。在第4阶段中，您绕过了现代系统用于阻止缓冲区溢出攻击的两个主要设备。尽管您没有注入自己的代码，但您能够注入一种通过将现有代码序列拼接在一起来运行的程序。你在实验中也得到了95/100分。这是一个很好的成绩。如果你有其他紧迫的义务，考虑现在就停止。

第5阶段要求您对rtarget进行ROP攻击，以使用指向cookie的字符串表示的指针调用函数touch3。这似乎并不比使用ROP攻击来调用Touch2更困难，除非我们已经这样做了。此外，第5阶段只计5分，这并不是对其所需努力的真正衡量。对于那些想要超出课程正常预期的人来说，这更像是一个额外的学分问题。

A. MOVQ指令的编码

莫夫克S，D

源头 *S*	目的地D							
	%rax	%rcx	%rdx	%rbx	%rsp	%rbp	%rsi	%rdi
%rax	48 89 c0	48 89 c1	48 89 c2	48 89 c3	48 89 c4	48 89 c5	48 89 c6	48 89 c7
%rcx	48 89 c8	48 89 c9	48 89 ca	48 89 cb	4889立方	4889光盘	公元4889	4889参见
%rdx	48 89 d0	48 89 d1	48 89 d2	48 89 d3	厘米	48 89 d5	年	48 89 d7
%rbx	48 89 d8	48 89 d9	48 89 da	4889分贝	48 89 d4	48 89 dd	48 89 d6	48 89 df
%rsp	48 89 e0	48 89 e1	48 89 e2	48 89 e3	4889直流	48 89 e5	48 89 de	48 89 e7
%rbp	48 89 e8	48 89 e9	4889个	48 89 eb	48 89 e4	48/89版	48 89 e6	48 89 ef
%rsi	48 89 f0	48 89 f1	48 89 f2	48 89 f3	48-89 EC	48 89 f5	48 89 ee	48 89 f7
%rdi	48 89 f8	48 89 f9	48 89 fa	48 89 fb	48 89 f4	48 89 fd	48 89 f6	第48至89
					4889英尺3英寸		4889铁	页

B. POPQ指令的编码

手术	寄存器R							
	%rax	%rcx	%rdx	%rbx	%rsp	%rbp	%rsi	%rdi
波普克河	58	59	5a	5b	5c	5d	5e	5f

C. MOVL指令的编码

莫夫尔S，D

源头 *S*	目的地D							
	%eax	%ecx	%edx	%ebx	%esp	%ebp	%esi	%edi
%eax	89 c0	89 c1	89 c2	89 c3	89 c4	89 c5	89 c6	89 c7
%ecx	89 c8	89 c9	89 ca	89 cb	89立方	89张CD	公元89	89参
%edx	89 d0	89 d1	89 d2	89 d3	厘米	89 d5	年	89 d7
%ebx	89 d8	89 d9	89 da	89分贝	89 d4	89 dd	89 d6	89 df
%esp	89 e0	89 e1	89 e2	89 e3	89直流	89 e5	89 de	89 e7
%ebp	89 e8	89 e9	89个	89 eb	89 e4	第89版	89 e6	89 ef
%esi	89 f0	89 f1	89 f2	89 f3	89欧共体	89 f5	89 ee	89 f7
%edi	89 f8	89 f9	89 fa	89 fb	89 f4	89 fd	89 f6	第89页起
					89个FC		89铁	

D. 2字节功能NOP指令的编码

手术		寄存器R			
		%铝	%cl	%dl	%bl
和B	*R*, *R*	20 c0	20 c9	20 d2	20 db
宝珠	*R*, *R*	08 c0	08 c9	08 d2	08 db
cmpb	*R*, *R*	38 c0	38 c9	38 d2	38 db
测试B	*R*, *R*	84 c0	84 c9	84 d2	84 db

图3:指令的字节编码。所有值均以十六进制显示。

要解决第5阶段的问题，您可以在rTarget中由函数“启动_农场”和“结束_农场”划分的代码区域中使用小工具。除了阶段4中使用的小工具之外，该扩展场还包括不同MOVL指令的编码，如图3C所示。群的该部分中的字节序列还包含用作功能NOP的2字节指令，即，它们不改变任何寄存器或存储器值。这些指令包括图3D所示的指令，如ANDB%AL、%AL，它们对某些寄存器的低位字节进行操作，但不改变其值。

一些建议:

- 如正文第183页所述，您将需要查看MOVL指令对寄存器的高4个字节的影响。
- 官方解决方案需要八个小工具（并非所有小工具都是唯一的）。

祝你好运，玩得开心！

A 使用HEX2RAW

HEX2RAW将十六进制格式的字符串作为输入。在此格式中，每个字节值由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“ 30313233343500 ”。（请记住，十进制数字X的ASCII代码是0x3x，字符串的结尾由空字节表示。）

传递给HEX2RAW的十六进制字符应由空格（空白或换行符）分隔。我们建议您在处理漏洞时，用换行符分隔漏洞字符串的不同部分。
Hex2Raw支持C风格的块注释，因此您可以标记出您的漏洞字符串的部分。例如:

```
48 C7 C1 F0 114000/*电动势       $0x40011f0,%rcx */
```

请确保在开始和结束注释字符串（“/*”，“*/”）周围留出空间，以便正确忽略注释。

如果在Exploit.txt文件中生成十六进制格式的漏洞利用字符串，则可以将原始字符串应用于ctarget或rtarget有几种不同的方式:

1. 您可以设置一系列管道来通过hex2raw传递字符串。

   ```
   UNIX amp; gt; cat Exploit.txt|./hex2raw|./ctarget
   ```

2. 您可以将原始字符串存储在文件中并使用I/O重定向:

   ```
   UNIX amp; gt; ./hex2raw&lt; Exploit.txt&gt; Exploit-raw.txt
   UNIX amp; gt; ./C目标&lt; Exploit-raw.txt
   ```

   从GDB内部运行时也可以使用这种方法:

```
UNIX amp; gt; 广东发展银行目标
（广发行）运行&lt; Exploit-raw.txt
```

3. 您可以将原始字符串存储在文件中，并将文件名作为命令行参数提供:

```
UNIX amp; gt; ./hex2raw&lt; Exploit.txt&gt; Exploit-raw.txt
UNIX amp; gt; ./ctarget-I exploit-raw.txt
```

当从GDB内部运行时，也可以使用这种方法。

B 正在生成字节代码

使用GCC作为汇编器，OBJDUMP作为反汇编器，可以方便地生成指令序列的字节码。例如，假设您编写了一个包含以下汇编代码的文件example.s:

```
#手工生成的汇编代码示例
        普什克    $0xabcdef           #将值压入堆栈addQ    17美
        元，%RAX #将17添加到%rax
        莫夫尔    %eax,%edx           #将低32位复制到%EDX
```

代码可以包含指令和数据的混合。“#”字符右边的任何内容都是注释。

现在，您可以对此文件进行汇编和反汇编:

```
UNIX amp; gt; GCC-C example.s
UNIX amp; gt; objdump-d示例.o&gt; 示例.d
```

生成的文件example.d包含以下内容:

```
示例o:          文件格式ELF64-x86-64


节的分解。文本：

0000000000000000&lt; .文本&gt; :
   0: 68 ef cd ab 00        pushq  $0xabcdef
   5: 48 83 c0 11           加     $0x11,%rax
   9: 89 c2                 移动    %eax,%edx
```

底部的行显示了从汇编语言指令生成的机器代码。每一行的左边都有一个十六进制数字，表示指令的起始地址（从0开始），而

“：”字符后面的十六进制数字表示指令的字节代码。因此，我们可以看到指令PUSH$0xABCDEF具有十六进制格式的字节代码68 EF CD AB 00。
从这个文件中，您可以获得代码的字节序列:

```
68 ef cd ab 00 48 83 c0 11 89 c2
```

然后，可以通过HEX2RAW传递该字符串，以生成目标程序的输入字符串。或者，您可以编辑example.d以省略无关的值并包含C样式的注释以提高可读性，从而产生:

```
68 ef cd ab 00   /* pushq  $0xabcdef  */
48 83 c0 11      /*添加     $0x11,%rax */
89 c2            /*移动     %eax,%edx  */
```

这也是一个有效的输入，您可以在发送到其中一个目标程序之前通过hex2raw。

参考文献

[1] R.Roemer，E.Buchanan，H.Shacham和S.Savage。面向返回的编程：系统、语言和应用。《美国计算机学会信息系统安全学报》，15（1）:2:1–2:34，2012年3月。

[2] E.J.Schwartz、T.Avgerinos和D.Brumley。Q：利用强化变得容易。2011年USENIX安全研讨会。