15-213，20xx年秋季
攻击实验室。了解缓冲区溢出的漏洞分配。9月29日，
星期二
截止日期：美国东部时间10月8日，星期四，11:59PM
最后可能的交卷时间。美国东部时间10月11日，星期日
，11:59PM

1 简介

这项任务涉及对两个具有不同安全漏洞的程序产生总共五次攻击。你将从这个实验室中获得的成果包括。

- 你将学习当程序不能很好地保护自己免受缓冲区溢出时，攻击者可以利用安全漏洞的不同方式。
- 通过这些，你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，使程序不那么脆弱。
- 你将对x86-64机器代码的堆栈和参数传递机制有更深入的了解。
- 你将对x86-64指令的编码方式有更深入的了解。
- 你将获得更多使用调试工具的经验，如GDB和OBJDUMP。

注意：在这个实验室中，你将获得利用操作系统和网络服务器的安全弱点的方法的第一手经验。我们的目的是帮助你学习程序运行时的操作，了解这些安全弱点的性质，以便你在编写系统代码时能够避免这些弱点。我们不赞成使用任何其他形式的攻击来获得对任何系统资源的未授权访问。

你需要学习CS:APP3e书中的3.10.3和3.10.4节，作为本实验的参考材料。

2 物流

像往常一样，这是一个个人项目。你将为目标程序产生攻击，这些攻击是为你定制的。

2.1 获取文件

你可以通过将你的网络浏览器指向以下网址来获得你的文件。

```
http://$Attacklab::SERVER_NAME:15513/
```

教员：`$Attacklab::SERVER_NAME`是运行`attacklab`服务器的机器。你在`attacklab/Attacklab.pm`和`attacklab/src/build/driverhdrs.h`中定义它。

服务器将建立你的文件，并将它们放在一个名为`targetk.tar`的压缩文件中返回给你的浏览器，其中*k*是你的目标程序的唯一编号。

注意：建立和下载你的目标需要几秒钟，所以请耐心等待。

将`targetk.tar`文件保存在一个你计划进行工作的（受保护的）Linux目录中。然后发出命令：`tar -xvf targetk.tar`，这将提取一个包含下述文件的目录`targetk`。

你应该只下载一组文件。如果由于某种原因你下载了多个目标，请选择一个目标进行工作，并删除其余的。

警告。如果你在电脑上扩展`targetk.tar`，通过使用Winzip等工具，或让你的浏览器进行提取，你将有可能重置可执行文件的权限位。

`targetk`中的文件包括。

`README.txt`。一个描述目录内容的文件

`ctarget`:一个容易受到*代码注入*攻击的可执行程序

`rtarget`。一个易受*面向返回编程*攻击的可执行程序

`cookie.txt`。一个8位数的十六进制代码，你将在你的攻击中作为一个独特的标识符。

`farm.c`:你的目标 "小工具农场 "的源代码，你将用它来生成面向返回的编程攻击。

`hex2raw`:一个用于生成攻击字符串的工具。

在下面的说明中，我们将假设你已经将文件复制到一个受保护的本地目录，并且你正在该本地目录中执行程序。

2.2 重要要点

以下是关于本实验室有效解决方案的一些重要规则的总结。当你第一次阅读本文件时，这些要点不会有什么意义。它们在此提出，作为你开始后的规则的核心参考。

- 你必须在一台与产生你的目标的机器相似的机器上做作业。
- 你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，你纳入攻击字符串中用于ret指令的任何地址都应该是到以下目的地之一。
  - 触摸1、触摸2或触摸3功能的地址。
  - 你注入的代码的地址
  - 你的一个小工具的地址，来自小工具农场。
- 你只能从文件rtarget中构建地址在start_farm和end_farm这两个功能之间的小工具。

3 目标方案

CTARGET和RTARGET都从标准输入读取字符串。它们通过函数getbuf来实现
定义如下。

```
1 无符号的getbuf()
2 {
3     char buf[BUFFER_SIZE]。
4     Gets(buf)。
5     返回 1;
6 }
```

该函数Gets类似于标准库函数get--
它从标准输入中读取一个字符串（以'\n'或文件结尾结束），并将其（连同一个空结尾）存储在指定的目的地。在这段代码中，你可以看到目的地是一个数组buf，被声明为有BUFFER_SIZE字节。在你的目标生成时，BUFFER_SIZE是一个编译时的常数，具体到你的程序版本。

函数Gets()和gets()没有办法确定它们的目标缓冲区是否足够大来存储它们读取的字符串。它们只是简单地复制字节序列，可能会超出目的地分配的存储空间的界限。

如果用户输入并由getbuf读取的字符串足够短，很明显getbuf将返回1，正如下面的执行例子所示。

```
unix> ./ctarget
```

饼干。0x1a7dd803

类型字符串。*保持简短!*

没有被利用。Getbuf返回0x1 正常返回

通常情况下，如果你输入一个长字符串，就会发生错误。

unix> *./ctarget*

饼干。0x1a7dd803

类型字符串。*这不是一个非常有趣的字符串，但它有一个属性 ...*

哎哟！。你造成了一个分段故障!

下次运气会更好

(注意显示的cookie的值将与你的不同。)程序RTARGET将有同样的行为。正如错误信息所显示的那样，缓冲区的超限通常会导致程序状态被破坏，从而导致内存访问错误。你的任务是对你提供给CTARGET和RTARGET的字符串进行更巧妙的处理，使它们做出更有趣的事情。这些被称为"*利用字*符串"。

CTARGET和RTARGET都接受几个不同的命令行参数。

-h:打印可能的命令行参数列表

-q：不向评分服务器发送结果

-i FILE。从一个文件而不是标准输入中提供输入。

你开发的字符串通常会包含与打印字符的ASCII值不一致的字节值。HEX2RAW程序将使您能够生成这些*原始字*符串。关于如何使用HEX2RAW的更多信息见附录A。

重要的一点是。

- 你的漏洞字符串在任何中间位置都不能包含字节值0x0a，因为这是换行的ASCII码（'\n'）。当Gets遇到这个字节时，它将认为你打算终止这个字符串。
- HEX2RAW希望两位数的十六进制值由一个或多个空格分隔。所以如果你想创建一个十六进制值为0的字节，你需要把它写成00。要创建0xdeadbeef这个词，你应该把 "ef be ad de "传给HEX2RAW（注意小-endian字节排序所需的反转）。

当你正确解决了其中一个关卡，你的目标程序将自动向评分服务器发送通知。比如说

unix> *./hex2raw < ctarget.l2.txt | ./ctarget*

饼干。0x1a7dd803

Type string:Touch2!你调用了touch2(0x1a7dd803)有效的解决方

案，2级的目标ctarget。

通过了。发送漏洞字符串到服务器进行验证。不错的工作!

阶段	节目	级别	方法	职能	积分
1 2 3	CTARGET ctarget ctarget	1 2 3	识别和识别(CI) CI CI	触摸1 触摸2 触摸3	10 25 25
4 5	追踪目标(RTARGET) 追踪目标(RTARGET)	2 3	ROP ROP	触摸2 触摸3	35 5

CI: 代码注入
ROP。 面向返回的编程

图1：攻击实验室阶段的总结

服务器将测试你的漏洞字符串，以确保它确实有效，并将更新Attacklab的评分板页面，表明你的用户名（为了匿名，由你的目标号码列出）已完成这一阶段。

你可以通过将你的网络浏览器指向以下位置来查看记分牌

```
http://$Attacklab::SERVER_NAME:15513/scoreboard
```

与炸弹实验室不同的是，在这个实验室里犯错是没有惩罚的。请随意向CTARGET开火。
和RTARGET，用你喜欢的任何字符串。

重要提示：你可以在任何Linux机器上研究你的解决方案，但为了提交你的解决方案，你需要在以下机器上运行。

导师：插入你在`buflab/src/config.c`中建立的合法域名列表。

图1概述了该实验室的五个阶段。可以看出，前三个阶段涉及对CTARGET的代码注入（CI）攻击，而后两个阶段涉及对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，你的漏洞字符串将攻击CTARGET。这个程序的设置方式是，堆栈位置在每次运行时都是一致的，因此堆栈上的数据可以被视为可执行代码。这些特点使该程序容易受到攻击，即利用字符串包含可执行代码的字节编码。

4.1 第1级

对于第1阶段，你不会注入新的代码。相反，你的漏洞字符串将重新引导程序执行一个现有的程序。

函数`getbuf`在CTARGET内被一个具有以下C代码的函数测试所调用。

```
1 无效的测试 () 。
2 {
3     int val;
4     val = getbuf()。
5     printf("No exploit. Getbuf returned 0x%x\n", val)。
6 }
```

当getbuf执行其返回语句时（getbuf的第5行），程序通常在函数test内恢复执行（在该函数的第5行）。我们想改变这种行为。在文件ctarget 中，有一个函数touch1的代码，其C语言表示如下。

```
1 空白 触摸1()
2 {
3     vlevel = 1;/* 验证协议的一部分 */
4     printf("Touch1!You called touch1()\n")。
5     验证(1)。
6     退出(0)。
7 }
```

你的任务是让CTARGET在getbuf执行其返回语句时执行touch1的代码，而不是返回到 测 试。请注意，你的漏洞字符串也可能破坏堆栈中与这个阶段没有直接关系的部分，但这不会造成问题，因为touch1会导致程序直接退出。

一些建议。

- 所有你需要的信息都可以通过检查CTARGET的反汇编版本来确定，以设计出你的利用字符串。使用objdump -d来获得这个反汇编的版本。
- 我们的想法是为touch1的起始地址定位一个字节的表示，这样，ret 指令在getbuf的代码末尾将控制权转移到touch1。
- 要注意字节的排序。
- 你可能想用GDB来引导程序完成getbuf的最后几条指令，以确保它正在做正确的事情。
- buf在getbuf的堆栈框架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。你需要检查反汇编的代码以确定其位置。

4.2 第2级

第二阶段包括注入少量代码作为你的漏洞字符串的一部分。

在ctarget文件中，有一个函数touch2的代码，其C语言表示如下。

1 空白的touch2(unsigned val)。

```
2 {
3     vlevel = 2;/* 验证协议的一部分 */
4     如果(val == cookie) {
5          printf("Touch2!     你叫touch2(0x%.8x)/n", val)。
6         验证(2)。
7     } else {
8            printf("失火。      你叫touch2(0x%.8x)/n", val)。
9         不合格(2)。
10    }
11    退出(0)。
12 }
```

你的任务是让CTARGET执行touch2的代码，而不是返回到test。然而，在这种情况下，你必须让touch2看起来就像你把你的cookie作为它的参数一样。

一些建议。

- 你将想把你注入的代码的地址的字节表示法定位在这样一种方式上，即getbuf代码末尾的ret指令将控制权转移给它。
- 回顾一下，函数的第一个参数是在寄存器%rdi中传递的。
- 你注入的代码应该将寄存器设置为你的cookie，然后使用ret指令将控制权转移到touch2的第一条指令。
- 不要试图在你的漏洞代码中使用jmp或调用指令。这些指令的目标地址的编码很难制定。在所有的控制转移中使用ret指令，即使你没有从调用中返回。
- 参见附录B中关于如何使用工具生成指令序列的字节级表示的讨论。

4.3 3级

第3阶段也涉及代码注入攻击，但传递一个字符串作为参数。

在ctarget文件中，有函数hexmatch和touch3的代码，其C语言表达方式如下。

```
1 /* 将字符串与无符号值的十六进制代表进行比较 */
2 int hexmatch(unsigned val, char *sval)
3 {
4     char cbuf[110];
5     /* 使检查字符串的位置不可预测 */
6     char *s = cbuf + random() % 100;
7     sprintf(s, "%.8x", val)。
8     return strncmp(sval, s, 9) == 0;
9 }
```

```
10
11 空白触摸3(char *sval)
12 {
13     vlevel = 3;/* 验证协议的一部分 */
14     如果(hexmatch(cookie, sval)) {
15          printf("Touch3! 你叫touch3(%s\")\n", sval)。
16         验证(3)。
17     } else {
18            printf("失火。 你叫touch3(%s\")\n", sval)。
19         失败(3)。
20     }
21     退出(0)。
22 }
```

你的任务是让CTARGET为touch3执行代码，而不是返回到测试
。你必须让它在touch3看来，就像你把你的cookie的一个字符串表示作为它的参数一样。

一些建议。

- 你将需要在你的漏洞字符串中包括一个表示你的cookie的字符串。该字符串应该由八个十六进制数字组成（从最重要的到最不重要的排序），没有前导 "0x"。
- 回顾一下，在C语言中，一个字符串被表示为一串字节，后面是一个值为0的字节。在任何一台Linux机器上输入 "man ascii"，可以看到你需要的字符的字节表示。
- 你注入的代码应该将寄存器%rdi设置为这个字符串的地址。
- 当函数hexmatch和strncmp被调用时，它们将数据推入堆栈，覆盖了存放getbuf使用的缓冲区的部分内存。因此，你需要小心地把你的cookie的字符串表示放在哪里。

5 第二部分：面向回报的编程

对程序RTARGET进行代码注入攻击要比CTARGET困难得多，因为它使用两种技术来挫败这种攻击。

- 它使用随机化，使堆栈位置从一个运行到另一个不同。这使得你不可能确定你注入的代码将位于何处。
- 它将存放堆栈的内存部分标记为不可执行，所以即使你能将程序计数器设置为你注入的代码的开始，程序也会因分段故障而失败。

幸运的是，聪明的人们已经设计出了一些策略，通过执行现有的代码，而不是注入新的代码来完成程序中有用的东西。这方面最普遍的形式被称为*面向返回的编程*（ROP）[1, 2]。ROP的策略是在现有的程序中找出由一条或多条指令组成的字节序列，然后是指令ret。这

样的片段被称为

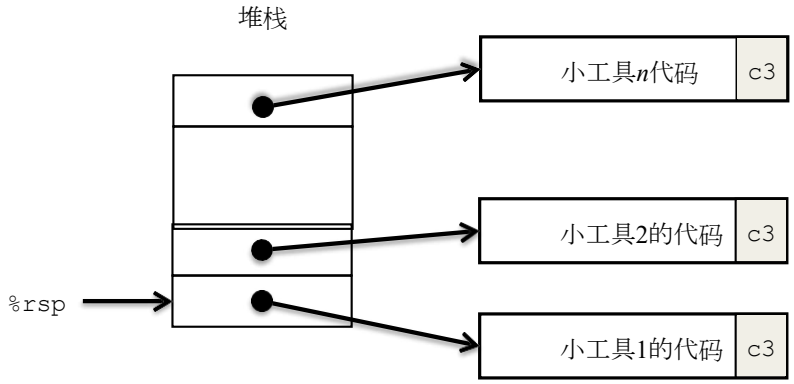

图2：设置小工具的执行序列。字节值0xc3对ret指令进行编码。

小工具。图2说明了堆栈如何被设置为执行一连串的***n*** *个*小工具。在这个图中，堆栈包含一连串的小工具地址。每个小工具由一系列的指令字节组成，最后一个是0xc3，编码为ret指令。当程序从这个配置开始执行ret指令时，它将启动一连串的小工具执行，每个小工具末尾的ret指令会使程序跳到下一个小工具的开头。

一个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，特别是在函数末端的语句。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，一个编译后的函数极不可能将popq %rdi作为ret前的最后一条指令。幸运的是，对于面向字节的指令集，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一个小工具。

例如，一个版本的rtarget包含为以下C函数生成的代码。

```
空白的setval_210(unsigned *p)。
{
    *p = 3347663060U。
}
```

这个函数对攻击系统有用的机会似乎很渺茫。但是，这个函数的反汇编机器代码显示了一个有趣的字节序列。

```
0000000000400f15 <setval_210>:
  400f15:       c7 07 d4 48 89 c7     搬家公司    $0xc78948d4,(%rdi)
  400f1b:       c3                       retq
```

字节序列48 89 c7编码了指令movq %rax, %rdi。(参见图3A中有用的movq指令的编码。)这个序列后面是字节值c3，它编码的是ret指令。该函数从地址0x400f15开始，该序列从该函数的第四个字节开始。因此，这段代码包含一个小工具，其起始地址为0x400f18，它将把寄存器%rax中的64位值复制到寄存器%rdi中。

你的RTARGET的代码包含了一些与上面显示的setval_210函数类似的函数，在一个我们称为*小工具场*的区域。你的工作将是在小工具场中识别有用的小工具，并利用这些小工具进行类似于你在第二和第三阶段所做的攻击。

重要提示：小工具农场由函数start_farm和end_farm划定，在你的rtarget。不要试图从程序代码的其他部分构建小工具。

5.1 第2级

对于第四阶段，你将重复第二阶段的攻击，但在程序RTARGET上使用你的小工具农场的小工具。你可以使用由以下指令类型组成的小工具构建你的解决方案，并且只使用前八个x86-64寄存器（%rax-%rdi）。

movq 。这些代码如图3A所示。

popq 。这些代码在图3B中显示。

ret ：该指令由单字节0xc3编码。

nop :这条指令（读作 "no op"，是 "无操作"的简称）由单字节0x90编码。它的唯一作用是使程序计数器增量为1。

一些建议。

- 你需要的所有小工具都可以在由函数start_farm和mid_farm划分的rtarget代码区域中找到。
- 你只需用两个小工具就可以进行这种攻击。
- 当一个小工具使用popq指令时，它将从堆栈中弹出数据。因此，你的漏洞字符串将包含一个小工具地址和数据的组合。

5.2 三级

在你进行第五阶段的学习之前，暂停一下，想想你到目前为止已经完成了什么。在第二阶段和第三阶段，你使一个程序执行你自己设计的机器代码。如果CTARGET是一个网络服务器，你就可以把你自己的代码注入到一个遥远的机器中。在第四阶段，你绕过了现代系统用来阻止缓冲区溢出攻击的两个主要装置。虽然你没有注入自己的代码，但你能够注入一种通过拼接现有代码序列来运作的程序。你也得到了95/100分的实验成绩。这是一个不错的分数。如果你有其他紧迫的义务，可以考虑现在就停止。

第五阶段要求你对RTARGET进行ROP攻击，用一个指向你的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难得多，只是我们把它变成了这样。此外，第5阶段只占5分，这并不是对其所需努力的真正衡量。对于那些想超越课程正常预期的人来说，可

以把它看作是一个额外的信用问题。

A. movq指令的编码

movq ***S***, ***D***

来源 ***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	48 89 cc	48 89 ĀĀĀ	48 89 终止	48 89 cf
%rdx	48 89 d0	48 89 d1	48 89 d2	48 89 d3	48 89 d4	48 89 d5	48 89 d6	48 89 d7
%rbx	48 89 d8	48 89 d9	48 89 da	48 89 db	48 89 邓小平	48 89 dd	48 89 德	48 89 df
%rsp	48 89 e0	48 89 e1	48 89 e2	48 89 e3	48 89 e4	48 89 e5	48 89 e6	48 89 e7
%rbp	48 89 e8	48 89 e9	48 89 ĀĀĀ	48 89 eb	48 89 生态	48 89 编制	48 89 ee	48 89 ef
%rsi	48 89 f0	48 89 f1	48 89 f2	48 89 f3	48 89 f4	48 89 f5	48 89 f6	48 89 f7
%rdi	48 89 f8	48 89 f9	48 89 fa	48 89 fb	48 89 fc	48 89 fd	48 89 羽毛	48 89 ff

B. popq指令的编码

运作	寄存器***R***							
	%rax	%rcx	%rdx	%rbx	%rsp	%rbp	% rsi	%rdi
popq ***R***	58	59	5a	5b	5c	5d	5e	5f

C. movl指令的编码

movl ***S***, ***D***

来源 ***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毫升	89cd	第89届	89 cf
%edx	89 d0	89 d1	89 d2	89 d3	89 d4	89 d5	会议	89 d7
%ebx	89 d8	89 d9	89 da	89分贝	89 dc	89 dd	89 d6	89 df
%esp	89 e0	89 e1	89 e2	89 e3	89 e4	89 e5	89德	89 e7
%ebp	89 e8	89 e9	89 ea	89 eb	89 生	89版	89 e6	89 ef
%esi	89 f0	89 f1	89 f2	89 f3	态	89 f5	89 ee	89 f7
%edi	89 f8	89 f9	89 fa	89 fb	89 f4	89 fd	89 f6	89 ff
					89 fc		89羽	

D. 2字节功能nop指令的编码

运作		寄存器***R***			
		%al	%cl	%dl	%bl
和b	***R***, ***R***	20 c0	20 c9	20 d2	20 db
兽骨	***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。所有数值都以十六进制显示。

为了解决第5阶段，你可以在rtarget中由函数start_farm和end_farm划分的代码区域使用小工具。除了第4阶段使用的小工具外，这个扩展的农场还包括不同movl指令的编码，如图3C所示。农场这部分的字节序列还包含作为*功能nops的*2字节指令，即它们不改变任何寄存器或内存值。这些指令包括图3D所示的指令，如andb
%al,%al，它们对一些寄存器的低阶字节进行操作，但不改变其值。

一些建议。

- 你要回顾一下movl指令对寄存器上部4个字节的影响，这在文中第183页有描述。
- 官方的解决方案需要八个小工具（并非所有的小工具都是独一无二的）。

祝您好运，玩得开心!

A 使用HEX2RAW

HEX2RAW接受一个*十六进制格式的*字符串作为输入。在这种格式中，每个字节的值由两个十六进制数字表示。例如，字符串 "012345 "可以用十六进制格式输入为 "30 31 32 33 34 35 00"。(记得十进制数字*x*的ASCII码是0x3x，字符串的结尾用一个空字节表示）。

您传递给HEX2RAW的十六进制字符应该用空格（空白或换行）分隔。我们建议在您工作时用换行符分隔您的漏洞利用字符串的不同部分。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。比如说。

```
48 c7 c1 f0 11 40 00 /* mov$0x40011f0     ,%rcx */
```

请确保在开始和结束的注释字符串（"/*"，"*/"）周围都留有空格，这样注释就会被正确地忽略掉。

如果你在文件exploit.txt中生成一个十六进制格式的漏洞字符串，你可以将原始字符串应用于CTARGET或RTARGET有几种不同的方式。

1. 你可以设置一系列的管道来通过HEX2RAW传递字符串。

   ```
   unix> cat exploit.txt | ./hex2raw | ./ctarget
   ```

2. 你可以将原始字符串存储在一个文件中并使用I/O重定向。

   ```
   unix> ./hex2raw < exploit.txt > exploit-raw.txt
   unix> ./ctarget < exploit-raw.txt
   ```

   这种方法也可以在从GDB内部运行时使用。

```
unix> gdb ctarget
(gdb) 运行 < exploit-raw.txt
```

3. 你可以将原始字符串存储在一个文件中，并提供文件名作为命令行参数。

```
unix> ./hex2raw < exploit.txt > exploit-raw.txt
unix> ./ctarget -i exploit-raw.txt
```

这种方法也可以在从GDB内部运行时使用。

B 生成字节代码

将GCC作为汇编程序，将OBJDUMP作为反汇编程序，可以方便地生成指令序列的字节码。例如，假设你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的例子
        pushq   $0xabcdef# 将数值推到堆栈上 addq  $17,%rax#
                把17加到%rax上
        movl    %eax,%edx# 复制低32位至%edx
```

代码可以包含指令和数据的混合物。在 "#"字符右边的任何东西都是注释。

你现在可以组装和拆卸这个文件了。

```
unix> gcc -c example.s
unix> objdump -d example.o > example.d
```

生成的文件example.d包含以下内容。

```
example.o:文件格式elf64-x86-64


拆解部分.text。

0000000000000000 <.text>:
   0: 68 ef cd ab 00pushq $0xabcdef
   5: 48 83 c0 11           添加    $0x11,%rax
   9: 89 c2                 搬家    %eax,%edx
```

底部的几行显示的是由汇编语言指令生成的机器代码。每一行的左边有一个十六进制的数字，表示指令的起始地址（从0开始），而

':'字符后的十六进制数字表示指令的字节码。因此，我们可以看到指令`push $0xABCDEF`的十六进制代码为`68 ef cd ab 00`。

从这个文件中，你可以得到代码的字节序列。

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

然后这个字符串可以通过HEX2RAW来生成目标程序的输入字符串。修改后，你可以编辑example.d，省略不相干的值，并包含C语言风格的注释，以提高可读性，结果是。

```
68 ef cd ab 00/* pushq $0xabcdef */
48 83 c0 11/* add$0x11      ,%rax */
89 C2/* mov%eax   ,%edx */
```

这也是一个有效的输入，你可以在发送给某个目标程序之前通过HEX2RAW。

参考文献

[1] R.Roemer, E. Buchanan, H. Shacham, and S. Savage.面向返回的编程。系统、语言和应用。*ACM Transactions on Information System Security*, 15(1):2:1-2:34, March 2012.

[2] E.J. Schwartz, T. Avgerinos, and D. Brumley.Q: 漏洞加固变得简单。在*USENIX安全研讨会上*，2011年。