15-213,20xx 秋季
攻击实验室:了解缓冲区溢出错误分配时间:9 月 29 日星期二
截止日期:美国东部时间 10
月 8 日星期四晚上 11:59最后可能的上
交时间:美国东部时间 10 月 11 日星期日晚上 11:59

1 简介

该任务涉及对具有不同安全漏洞的两个程序进行总共五次攻击。您将从本实验中获得的成果包括:

- 您将了解攻击者可以在程序未完成时利用安全漏洞的不同方式
  足够好地保护自己免受缓冲区溢出。
- 借此,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使程序不易受到攻击的功能。

- 您将对x86-64 的堆栈和参数传递机制有更深入的了解
  机器代码。
- 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GDB和OBJDUMP 等调试工具的经验。

注意:在本实验中,您将亲身体验用于利用操作系统和网络服务器中的安全漏洞的方法。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它们。我们不容忍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。

您需要学习 CS:APP3e 书的第 3.10.3 节和第 3.10.4 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。您将为为您定制的目标程序生成攻击。

2.1 获取文件

您可以通过将 Web 浏览器指向以下位置来获取您的文件:

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

INSTRUCTOR:$Attacklab::SERVER_NAME 是运行
攻击实验室服务器。您在 attacklab/Attacklab.pm 和 attacklab/src/build/driverhdrs.h 中定义它

服务器将构建您的文件并以名为 targetk.tar 的 tar 文件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,所以请耐心等待。

将 targetk.tar 文件保存在您计划执行工作的(受保护的)Linux 目录中。然后给出命令:tar -xvf targetk.tar。这将提取包含下述文件的目录 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,请选择一个目标进行处理并删除其余目标。

警告:如果您在 PC 上扩展 targetk.tar,使用 Winzip 等实用程序或让您的浏览器进行提取,您将面临重置可执行文件权限位的风险。

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:易受面向返回编程攻击的可执行程序

cookie.txt:一个 8 位十六进制代码,您将在攻击中用作唯一标识符。

farm.c:目标“小工具农场”的源代码,您将使用它来生成面向返回的编程攻击。

hex2raw:生成攻击字符串的实用程序。

在以下说明中,我们假设您已将文件复制到受保护的本地目录,并且您正在该本地目录中执行程序。

2.2 要点

以下是有关此实验室有效解决方案的一些重要规则的摘要。当您第一次阅读本文档时,这些要点没有多大意义。一旦您开始使用它们,它们将在此处作为规则的主要参考。

- 您必须在与生成目标的机器类似的机器上进行分配。
- 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,您合并到攻击字符串中供 ret 指令使用的任何地址都应该指向以下目的地之一:
  - –函数 touch1、touch2 或 touch3 的地址。
  - –你注入代码的地址
  - –来自小工具农场的其中一个小工具的地址。
- 您只能从文件 rtarget 构建小工具,其地址范围介于 func 之间
  tions start_farm 和 end_farm。

3 目标项目

CTARGET和RTARGET都从标准输入读取字符串。他们使用下面定义的函数 getbuf 来做到这一点:

```
1未签名的 getbuf() 2 {
3          char buf [BUFFER_SIZE];
4          获取（buf）;返回
5          1;
6 }
```

函数 Gets 类似于标准库函数 gets 它从标准输入（以 \n 或文件结尾结束)读取字符串并将其存储（连同空终止符)在指定的目的地。
在此代码中,您可以看到目标是一个数组 buf,声明为具有 BUFFER_SIZE 字节。在生成目标时,BUFFER_SIZE 是特定于您的程序版本的编译时常量。

函数 Gets() 和 gets() 无法确定它们的目标缓冲区是否足够大以存储它们读取的字符串。它们只是简单地复制字节序列,可能会超出在目的地分配的存储范围。

如果getbuf读取的用户输入的字符串足够短,显然getbuf会返回1,如下执行示例所示:

```
unix> ./ctarget
```

饼干:0x1a7dd803
输入字符串:保持简短!
没有剥削。 getbuf 返回 0x1 正常返回

如果键入长字符串,通常会发生错误:

unix> ./ctarget Cookie:
0x1a7dd803 Type string: 这不是一
个很有趣的字符串,但它有属性 ...
哎哟!:你造成了分段错误!
祝下次好运

(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,缓冲区溢出通常会导致程序状态被破坏,从而导致内存访问错误。您的任务是更巧妙地处理您提供给CTARGET和RTARGET 的字符串,以便它们做更多有趣的事情。这些被称为漏洞利用字符串。

CTARGET和RTARGET都采用几个不同的命令行参数:

-h:打印可能的命令行参数列表

-q:不将结果发送到评分服务器

-i FILE:从文件而不是标准输入提供输入

您的漏洞利用字符串通常包含与打印字符的 ASCII 值不对应的字节值。 HEX2RAW程序将使您能够生成这些原始字符串。有关如何使用HEX2RAW 的更多信息,请参阅附录 A。

要点:

- 您的利用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,它会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,如果你想创建一个十六进制值为 0 的字节,你需要将其写为 00。要创建单词 0xdeadbeef,你应该将“ef be ad de”传递给 HEX2RAW(注意小端字节序所需的反转) .

当您正确解决了其中一个级别后,您的目标程序将自动向评分服务器发送通知。例如:

unix> ./hex2raw < ctarget.l2.txt | ./ctarget Cookie: 0x1a7dd803

输入字符串:Touch2!:您调用了 touch2(0x1a7dd803)
目标 ctarget 通过的第 2 级的有效解决方案:已将漏洞利用字符串发送到要验证的服务器。

不错的工作!

阶段程序级方法功能点					
1目标1 2目标2 3目标3			CI touch1 10 CI touch2 25 CI touch3 25		
4 RTARGET 2 ROP 触摸 2 35 5个	目标3		ROP touch3		5个

那里： 代码注入

ROP:面向返回的编程

图 1:攻击实验室阶段总结

服务器将测试您的漏洞利用字符串以确保它确实有效,并且它会更新 Attacklab 记分板页面,表明您的用户 ID（按您的目标号码列出以匿名）已完成此阶段。

您可以通过将 Web 浏览器指向

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

与炸弹实验室不同,在这个实验室中犯错不会受到惩罚。随意使用您喜欢的任何字符串在CTARGET和RTARGET上开火。

重要说明:您可以在任何 Linux 机器上处理您的解决方案,但为了提交您的解决方案,您需要在以下机器之一上运行：

INSTRUCTOR:插入您的合法域名列表
在 buflab/src/config.c 中建立。

图 1 总结了实验室的五个阶段。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

对于前三个阶段,您的漏洞利用字符串将攻击CTARGET。该程序的设置方式使堆栈位置从一次运行到下一次运行都是一致的,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 一级

对于第 1 阶段,您不会注入新代码。相反,您的漏洞利用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无效测试（） 2 {
3            整数值； val
4            = getbuf(); printf(    没有
5            利用。Getbuf 返回 0x%x\n    , val);
6 }
```

当 getbuf 执行它的 return 语句时（getbuf 的第 5 行），程序通常会在函数 test 中恢复执行（在该函数的第 5 行）。我们想改变这种行为。在文件 ctarget 中，有一个具有以下 C 表示的函数 touch1 的代码：

```
1 void touch1() 2 {
3            vlevel = 1;                    /* 验证协议的一部分 */
4            printf(    Touch1!: 你调用了 touch1()\n    );验证（1）；退出（0）；
5
6
7 }
```

你的任务是让CTARGET在 getbuf 执行它的 return 语句时执行 touch1 的代码，而不是返回到 test。请注意，您的漏洞利用字符串也可能会破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。

一些忠告：

- 可以通过检查CTARGET的反汇编版本来确定为此级别设计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。
- 想法是定位 touch1 起始地址的字节表示，以便 ret
  getbuf 代码末尾的指令会将控制转移到 touch1。
- 注意字节顺序。
- 您可能希望使用GDB逐步执行getbuf 的最后几条指令，以确保它在做正确的事情。
- buf 在getbuf 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。您将需要检查反汇编代码以确定其位置。

4.2 二级

第 2 阶段涉及注入少量代码作为漏洞利用字符串的一部分。

在文件 ctarget 中，有一个函数 touch2 的代码，具有以下 C 表示形式：

```
1 void touch2(unsigned val)
```

```
2 {
3               vlevel = 2;如果（val                    /* 验证协议的一部分 */
4               == cookie){
5                       printf(    Touch2!: 你调用了 touch2(0x%.8x)\n    , val);验证（2）；  } else { printf(    失火:你调用了
6                       touch2(0x%.8x)\n    ,
7               val);失败（2）；
8
9
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。然而,在这种情况下,您必须使 touch2 看起来好像您已将 cookie 作为其参数传递。

一些忠告：

- 您将希望以这样的方式定位注入代码地址的字节表示形式
  getbuf 代码末尾的 ret 指令会将控制权转移给它。
- 回想一下函数的第一个参数是在寄存器%rdi 中传递的。
- 你注入的代码应该设置寄存器到你的 cookie,然后使用 ret 指令来传输
  控制 touch2 中的第一条指令。
- 不要尝试在您的利用代码中使用jmp 或call 指令。这些指令的目标地址编码很难制定。使用 ret 指令进行所有转移

  控制,即使您没有从通话中返回。
- 请参阅附录 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 三级

第 3 阶段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,有函数 hexmatch 和 touch3 的代码,具有以下 C 表示形式：

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

```
10
11 void touch3(char *sval) 12 {

13          vlevel = 3;如果（十                    /* 验证协议的一部分 */
14          六进制匹配（cookie,sval）){
15                  printf(   Touch3!: 你调用了 touch3(\   %s\   )\n   , sval);验证（3）；} else { printf(   失火:你调用
16                  了 touch3(\   %s\   )
17          \n   , sval);失
18                  败（3）；
19
20          }
21          退出（0）；
22 }
```

你的任务是让CTARGET执行 touch3 的代码而不是返回测试。你必须让 touch3 看起来好像你已经传递了你的 cookie 的字符串表示作为它的参数。

一些忠告：

- 您需要在利用字符串中包含您的cookie 的字符串表示。该字符串应包含八个十六进制数字（从最高有效位到最低有效位排序），不带前导“0x”。
- 回想一下，字符串在 C 语言中表示为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。
- 您注入的代码应将寄存器%rdi 设置为该字符串的地址。
- 当函数 hexmatch 和 strncmp 被调用时，它们将数据压入堆栈，覆盖 getbuf 使用的缓冲区所在的内存部分。因此，您需要小心放置 cookie 的字符串表示形式。

5 第二部分：面向返回的编程

对程序RTARGET执行代码注入攻击比对CTARGET 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，因此每次运行的堆叠位置都不同。这使它很有影响力
  可以确定注入代码的位置。
- 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注入代码的开头，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事情的策略。最一般的形式称为面向返回编程 (ROP) [1, 2]。 ROP 的策略是识别现有程序中的字节序列，该字节序列由一条或多条指令组成，后跟指令 ret。这样的片段被称为

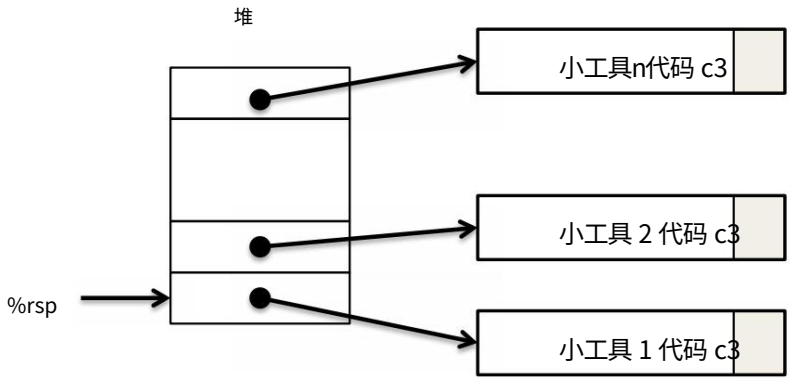

图 2:设置要执行的小工具序列。字节值 0xc3 对 ret 指令进行编码。

小工具。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。每个小工具都由一系列指令字节组成,最后一个是 0xc3,对 ret 指令进行编码。当程序以该配置开始执行 ret 指令时,它将启动小工具执行链,每个小工具末尾的 ret 指令使程序跳转到下一个小工具的开头。

小工具可以使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,可能会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,编译后的函数极不可能将 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,例如 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,一个版本的 rtarget 包含为以下 C 函数生成的代码:

```
void setval_210(unsigned *p) {

        *p = 3347663060U;
}
```

这个函数对攻击系统有用的可能性似乎很小。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                movl $0xc78948d4,(%rdi) retq
    400f1b:                 c3
```

字节序列 48 89 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。)此序列后跟字节值 c3,它对 ret 指令进行编码。该函数从地址 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,它将复制寄存器 %rax 中的 64 位值到寄存器 %rdi。

您的RTARGET代码包含许多类似于上面显示的 setval_210 函数的函数,在我们称为小工具农场的区域中。您的工作是在小工具农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 阶段所做的攻击。

重要提示:小工具农场由 rtarget 副本中的函数 start_farm 和 end_farm 划分。不要尝试从程序代码的其他部分构造小工具。

5.1 二级

对于第 4 阶段,您将重复第 2 阶段的攻击,但使用您的小工具农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您的解决方案,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 :该指令由单字节 0xc3 编码。

nop:这条指令(发音为“no op”,是“no operation”的缩写)由单字节 0x90 编码。它的唯一作用是使程序计数器加 1。

一些忠告:

- 您需要的所有小工具都可以在由
  函数 start_farm 和 mid_farm。
- 您可以只用两个小工具进行这种攻击。
- 当小工具使用popq 指令时,它将从堆栈中弹出数据。结果,你的利用
  字符串将包含小工具地址和数据的组合。

5.2 三级

在你开始第 5 阶段之前,停下来想想你到目前为止已经完成了什么。在第 2 和第 3 阶段,您使程序执行您自己设计的机器代码。如果CTARGET是网络服务器,您可以将自己的代码注入远程机器。在第 4 阶段,您绕过了现代系统用来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,但您可以注入一种程序,该程序通过将现有代码序列拼接在一起来运行。

您还获得了实验室的 95/100 分。这是一个很好的成绩。如果您有其他紧迫的义务,请考虑立即停止。

阶段 5 要求您对RTARGET进行 ROP 攻击,以调用带有指向 cookie 字符串表示形式的指针的函数 touch3。这似乎并不比使用 ROP 攻击调用 touch2 困难多少,只是我们已经做到了。此外,第 5 阶段仅计为 5 分,这并不是对其所需努力的真实衡量。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

移动 S, D

来源	目的地 D %rbx							
小号	%rax	%rcx	%rdx	%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 cd 48 89 ce 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 dc 48 89 dd 48 8 9 月 48 日								
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 ea 48 89 eb 48 89 ec 48 89编辑 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 89fd 48 89 日 48 89 日								

B. popq指令的编码

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

C. movl指令的编码

移动 S, D

来源	目的地 D %eax							
小号	%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 cc 89 cd 89 ce 89 cf								
%edx 89 d0 89 d1 89 d2 89 d3 89 d4 89 d5 89 d6 89 d7								
%ebx 89 d8 89 d9 89 da 89 db 89 dc 89 dd 89 de 89 df								
%esp 89 e0 89 e1 89 e2 89 e3 89 e4 89 e5 89 e6 89 e7 %ebp 89 e8 89 e9 89 ea 89 eb 89 ec 89 ed 89 ee 89 ef								
%esi 89 f0 89 f1 89 f2 89 f3 89 f4 89 f5 89 f6 89 f7 %edi 89 f8 89 f9 89 fa 89 fb 89 fc 89 fd 89 fe 89 ff								

D. 2 字节功能 nop 指令的编码

手术	寄存器 R			
	%al	%cl %dl		%bl
andb R, R 20 c0 20 c9 20 d2 20 db orb 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。所有值都以十六进制显示。

要解决第 5 阶段,您可以在 rtarget 中由函数 start_farm 和 end_farm 划分的代码区域中使用小工具。除了第 4 阶段中使用的小工具之外，这个扩展的农场还包括不同 movl 指令的编码,如图 3C 所示。这部分场中的字节序列还包含用作功能性 nop 的 2 字节指令,即它们不更改任何寄存器或内存值。这些包括指令,如图 3D 所示,例如 andb %al,%al,它们对某些寄存器的低位字节进行操作但不更改它们的值。

一些忠告：

- 您需要查看 movl 指令对寄存器的高 4 字节的影响,如下所示
  正文第 183 页中描述。
- 官方解决方案需要八个小工具（并非所有小工具都是独一无二的）。

祝好运并玩得开心点！

A 使用HEX2RAW

HEX2RAW将十六进制格式的字符串作为输入。在这种格式中,每个字节值由两个十六进制数字表示。例如,字符串“012345”可以十六进制格式输入为“30 31 32 33 34 35 00”。 （回想一下,十进制数字 x 的 ASCII 代码是 0x3x,字符串的结尾由空字节指示。）

您传递给HEX2RAW 的十六进制字符应该用空格（空格或换行符)分隔。我们建议您在处理漏洞利用字符串时用换行符分隔不同部分。

HEX2RAW支持 C 风格的块注释,因此您可以标记出漏洞利用字符串的各个部分。例如：

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

请务必在开始和结束注释字符串（“/*”、“*/”)两边留出空格,以便正确忽略注释。

如果您在文件 exploit.txt 中生成一个十六进制格式的漏洞利用字符串,您可以通过几种不同的方式将原始字符串应用于CTARGET或RTARGET：

1.可以设置一系列的管道让字符串通过HEX2RAW。

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

从 GDB 中运行时也可以使用这种方法：

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

3. 您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
```

当从 GDB 中运行时也可以使用这种方法。

B 生成字节码

使用GCC作为汇编器和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 pushq $0xabcdef
                                                    # 将值压入堆栈 addq $17,%rax
                                                    # 将 17 添加到 %rax movl
        %eax,%edx                                   # 复制低 32 位到 %edx
```

代码可以包含指令和数据的混合。 “#”字符右侧的任何内容都是注释。

您现在可以汇编和反汇编此文件:

```
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:

```
例子.o:     文件格式 elf64-x86-64
```

.text 部分的反汇编:

```
0000000000000000 <.文本>:
   0: 68 ef cd ab 00                   pushq $0xabcdef
   5: 48 83 c0 11                      加    $0x11,%rax 添
   9: 89 立方米                        ＊ %eax,%edx
```

底部的行显示从汇编语言指令生成的机器代码。每行左边有一个十六进制数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码 68 ef cd ab 00。

从此文件中，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传递此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 以省略无关值并包含 C 风格的注释以提高可读性，从而产生：

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

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传递。

参考

[1] R. Roemer、E. Buchanan、H. Shacham 和 S. Savage。面向返回的编程：系统、语言和应用程序。 ACM 信息系统安全交易，15(1):2:1–2:34，2012 年 3 月。

[2] EJ Schwartz、T. Avgerinos 和 D. Brumley。问：利用强化变得容易。在 USENIX 安全研讨会，2011 年。