20xx 年秋季 15-213
攻击实验室：了解缓冲区溢出漏洞
分配时间：9 月 29 日星期二
截止时间：美国东部时间 10 月 8 日星期四晚上 11:59
最后可能的交卷时间：美国东部夏令时 10 月 11 日星期日晚上 11:59

1 介绍

此任务涉及对具有不同安全漏洞的两个程序总共生成五次攻击。您将从本实验室获得的成果包括：

- 您将了解当程序不能很好地保护自身免受缓冲区溢出时，攻击者利用安全漏洞的不同方式。
- 通过这一点，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为降低程序易受攻击性提供的一些功能。
- 您将更深入地了解 x86-64 机器代码的堆栈和参数传递机制。
- 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：在本实验室中，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性质，以便您在编写系统代码时避免这些漏洞。我们不允许使用任何其他形式的攻击来获得对任何系统资源的未经授权访问。

您需要学习 CS:APP3e 手册的第 3.10.3 节和第 3.10.4 节，作为本实验室的参考资料。

2 后勤

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\u NAME 是运行 Attacklab 服务器的计算机。您可以在 attacklab/attacklab 中定义它。pm 和 in attacklab/src/build/driverhdrs。h 类

服务器将构建您的文件，并将它们以名为 target 的 tar 文件的形式返回到浏览器。tar，其中是目标程序的唯一编号。*kk*

注意：构建和下载目标需要几秒钟，所以请耐心等待。

保存目标。您计划在其中执行工作的（受保护的）Linux 目录中的 tar 文件。然后给出命令：tar-xvf target。焦油这将提取包含以下所述文件的目录 target。*kkk*

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

警告：如果扩展目标。在 PC 上，通过使用 Winzip 之类的实用程序，或让浏览器进行解压缩，您将面临重置可执行文件上权限位的风险。*k*

目标中的文件包括：*k*

自述。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，在攻击中用作唯一标识符。

农场 c： 目标“gadget farm”的源代码，用于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，我们假设您已将文件复制到受保护的本地目录，并且正在该本地目录中执行程序。

2.2 重要事项

这里总结了一些关于本实验室有效解决方案的重要规则。当您第一次阅读本文档时，这些要点没有多大意义。一旦开始，它们在这里作为规则的中心参考。

- 您必须在与生成目标的机器类似的机器上执行任务。
- 您的解决方案可能不会使用攻击绕过程序中的验证代码。具体而言，任何合并到攻击字符串中供 ret 指令使用的地址都应指向以下目标之一：
  - –功能 touch1、touch2 或 touch3 的地址。
  - –注入代码的地址

–小工具场中一个小工具的地址。

•您只能从文件 rtarget 构建小工具，其地址介于函数 start\u farm 和 end\u farm 的地址之间。

3 目标计划

CTARGET 和 RTARGET 都从标准输入读取字符串。它们使用下面定义的函数 getbuf 执行此操作：

```
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 get 与标准库函数 get 类似，它从标准输入中读取字符串（以“\n”或文件结尾终止），并将其（连同空终止符）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，声明为具有 BUFFER\u SIZE 字节。在生成目标时，BUFFER\u 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。它们只是复制字节序列，可能会超出在目的地分配的存储范围。

如果用户键入并由 getbuf 读取的字符串足够短，则 getbuf 将返回 1，如下执行示例所示：

```
unix>/C 目标
Cookie:0x1a7dd803
输入字符串：保持简短！
无漏洞。Getbuf 返回 0x1 正常返回
```

如果键入长字符串，通常会发生错误：

```
unix>/C 目标
Cookie:0x1a7dd803
类型字符串：这不是一个非常有趣的字符串，但它具有以下属性。。。
哎哟！：你造成了分割错误！
祝你下次好运
```

（请注意，显示的 cookie 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损坏，从而导致内存访问错误。您的任务是更巧妙地使用您为 CTARGET 和 RTARGET 提供的字符串，以便它们做更多有趣的事情。这些被称为利用字符串。

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： 不将结果发送到放坡服务器

-i 文件：提供来自文件的输入，而不是来自标准输入

利用漏洞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，请参见附录 A。

要点：

- 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字符串。
- HEX2RAW 需要两位十六进制值，由一个或多个空格分隔。因此，如果要创建十六进制值为 0 的字节，需要将其写入 00。要创建单词 0xdeadbeef，应该将“ef be ad de”传递给 HEX2RAW（注意小端字节排序所需的反转）。

正确解决其中一个级别后，目标程序将自动向放坡服务器发送通知。例如：

```
unix>/hex2raw<C 目标。l2.txt |/C 目标
Cookie:0x1a7dd803
类型字符串：Touch2！：您调用了 touch2（0x1a7dd803）
已传递目标 ctarget 的级别 2 的有效解决方案：已将漏洞字符串发送到要
验证的服务器。
干得好！
```

阶段	程序	数量	方法	作用	积分
1	C 目标	1	CI 公司	触摸 1	10
2	C 目标	2	CI 公司	触摸 2	25
3	C 目标	3	CI 公司	触摸 3	25
4	R 目标	2	ROP	触摸 2	35
5	R 目标	3	ROP	触摸 3	5

CI：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字符串，以确保其真正有效，并将更新 Attacklab 记分板页面，指示您的用户 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

http://$Attacklab：：服务器名称：15513/记分板

与炸弹实验室不同的是，在这个实验室犯错误不会受到惩罚。您可以随意使用任意字符串射击 CTARGET 和 RTARGET。

重要提示：您可以在任何 Linux 机器上使用解决方案，但为了提交解决方案，您需要在以下机器之一上运行：

讲师：插入您在 buflab/src/config 中建立的合法域名列表。c

图 1 总结了实验室的五个阶段。如图所示，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 的面向返回编程（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。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保持一致，因此堆栈上的数据可以被视为可执行代码。这些特性使程序容易受到攻击，其中漏洞字符串包含可执行代码的字节编码。

4.1 1 级

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

具有以下 C 代码的函数测试在 CTARGET 内调用函数 getbuf：

```
1    无效测试（）
2    {
3    int val；
4    val=getbuf（）；
5    printf（“无漏洞。Getbuf 返回 0x%x\n”，val）；6 }
```

当 getbuf 执行其 return 语句（getbuf 的第 5 行）时，程序通常会在函数测试中恢复执行（在该函数的第 5 行）。我们想改变这种行为。在文件 ctarget 中，有一个函数 touch1 的代码，其 C 表示形式如下：

```
1    void touch1（）
2    {
3    vlevel=1；/*验证方案的一部分*/
4    printf（“Touch1！：您调用了 Touch1（）\n”）；
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，而不是返回测试。请注意，利用漏洞字符串可能还会损坏堆栈中与此阶段不直接相关的部分，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，可以确定为该级别设计漏洞字符串所需的所有信息。使用 objdump-d 获得这个分解版本。
- 想法是定位 touch1 起始地址的字节表示，以便 getbuf 代码末尾的 ret 指令将控制权转移到 touch1。
- 注意字节顺序。
- 您可能希望使用 GDB 逐步完成 getbuf 的最后几条指令，以确保它正在做正确的事情。
- 在 getbuf 的堆栈帧中，buf 的位置取决于编译时常量 BUFFER\u SIZE 的值，以及 GCC 使用的分配策略。您需要检查反汇编代码以确定其位置。

4.2 2 级

阶段 2 涉及注入少量代码作为攻击字符串的一部分。

在文件 ctarget 中，有一个函数 touch2 的代码，其 C 表示形式如下：

```
1       无效触摸 2（未签名 val）
2       {
3       vlevel=2；/*验证方案的一部分*/
4       if（val==cookie）{
5       printf（“Touch2！：您调用了 Touch2（0x%.8x）\n”，val）；
6       验证（2）；
7       }其他{
8       printf（“失火：您调用了 touch2（0x%.8x）\n”，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。然而，在这种情况下，您必须让它看起来像是 touch2，就好像您传递了 cookie 作为其参数一样。

一些建议：

- 您需要定位注入代码地址的字节表示，以便 getbuf 代码末尾的 ret 指令将控制权转移给它。
- 回想一下，函数的第一个参数是在寄存器%rdi 中传递的。
- 注入的代码应将寄存器设置为 cookie，然后使用 ret 指令将控制转移到 touch2 中的第一条指令。
- 不要试图在利用漏洞代码中使用 jmp 或调用指令。这些指令的目标地址编码很难制定。对所有控制权转移使用 ret 指令，即使您没有从呼叫中返回。
- 参见附录 B 中关于如何使用工具生成指令序列字节级表示的讨论。

4.3 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，有函数 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形式*/
2        int hexmatch（无符号 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 返回 strncmp（sval，s，9）==0；9 }
10
11               无效触摸 3（字符*sval）
12               {
13               vlevel=3；/*验证方案的一部分*/
14               if（hexmatch（cookie，sval））{
15               printf（“Touch3！：您调用了 Touch3（\%s）”）\n，“sval）；
16               验证（3）；
17               }其他{
18               printf（“失火：您调用了 touch3（\%s）”）\n，“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。您必须使其看起来像是 touch3，就好像您传递了 cookie 的字符串表示形式作为其参数一样。

一些建议：

- 您需要在利用漏洞字符串中包含 cookie 的字符串表示形式。字符串应由八个十六进制数字组成（从最高位到最低位排序），无前导“0x”
- 回想一下，字符串在 C 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形式。
- 注入的代码应将寄存器%rdi 设置为此字符串的地址。
- 调用 hexmatch 和 strncmp 函数时，它们会将数据推送到堆栈上，覆盖保存 getbuf 使用的缓冲区的内存部分。因此，您需要小心地放置 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 执行代码注入攻击困难得多，因为它使用两种技术来阻止此类攻击：

- 它使用随机化，以便堆栈位置在不同的运行中有所不同。这使得无法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，聪明的人已经设计了一些策略，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的事情。最普遍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，该程序由一条或多条指令组成，后跟指令 ret。这种段称为

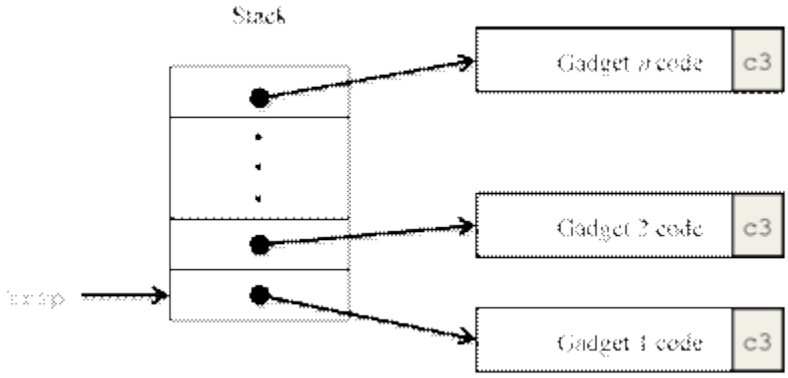

图 2：设置小工具的执行顺序。字节值 0xc3 对 ret 指令进行编码。

小工具. 图 2 说明了如何设置堆栈以执行一系列小工具。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，最后一个字节是 0xc3，用于编码 ret 指令。当程序执行从该配置开始的 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 指令将导致程序跳转到下一个小工具的开头。*n*

小工具可以使用编译器生成的汇编语言语句对应的代码，尤其是函数末尾的语句。实际上，可能有一些这种形式的有用小工具，但不足以实现许多重要操作。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，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\u 210（无符号*p）{
    *p=3347663060U；
}
```

此函数用于攻击系统的可能性似乎很小。但是，此函数的反汇编机器代码显示了一个有趣的字节序列：

0000000000 400F15：

400f15:c7 07 d4 48 89 c7	movl 公　$0xc78948d4，（%rdi） 司
400f1b:c3	retq 公司

字节序列 48 89 c7 对指令 movq%rax，%rdi 进行编码。（有关有用 movq 指令的编码，请参见图 3A。）该序列后面是字节值 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，此代码包含一个小工具，其起始地址为 0x400f18，它将寄存器%rax 中的 64 位值复制到寄存器%rdi。

您的 RTARGET 代码在我们称为 gadget farm 的区域中包含许多类似于上面所示的 setval\u 210 函数的函数。您的工作将是识别小工具场中有用的小工具，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：小工具场由 rtarget 副本中的函数 start\u farm 和 end\u farm 来划分。不要试图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，但在使用 gadget farm 中的 gadget 对程序 RTARGET 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：这些代码如图 3B 所示。ret：此指令由单字节 0xc3 编码。

nop：该指令（发音为“no op”，是“no operation”的缩写）由单字节 0x90 编码。其唯一作用是使程序计数器增加 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\u farm 和 mid\u 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，您的漏洞字符串将包含小工具地址和数据的组合。

5.2 3 级

在开始第 5 阶段之前，请暂停思考您迄今为止所取得的成就。在第 2 和第 3 阶段中，您使一个程序执行您自己设计的机器代码。如果 CTARGET 是一个网络服务器，那么您可以将自己的代码注入远程机器。在阶段 4 中，您绕过了现代系统用来阻止缓冲区溢出攻击的两种主要设备。虽然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。你在实验室也得到了 95/100 分。这是一个很好的分数。如果你有其他紧迫的义务，请考虑立即停止。

阶段 5 要求您对 RTARGET 执行 ROP 攻击，以使用指向 cookie 的字符串表示形式的指针调用函数 touch3。这似乎并不比使用 ROP 攻击调用 touch2 困难多少，但我们已经做到了这一点。此外，第 5 阶段仅计算 5 分，这不是衡量其所需努力的真正标准。对于那些想超越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 指令编码

movq，*SD*

来源 *S*	目的地 *D*							
	%rax 公司	%rcx 公司	%rdx 公司	%rbx 公司	%rsp	%限制性商业惯例	%rsi 指数	%rdi 公司
%rax 公司	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 公司	48 89 c8	48 89 c9	48 89 加利福尼亚州	48 89 cb	48 89 立方厘米	48 89 cd	48 89 ce	48 89 cf
%rdx 公司	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 公司	48 89 d8	48 89 d9	48 89 da	48 89 分贝	48 89 dc	48 89 日	48 89 de	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
%限制性商业惯例	48 89 e8	48 89 e9	48 个 89 个	48 89 eb	48 89 ec	48 89 版	48 89 ee	48 89 ef
%rsi 指数	48 89 f0	48 89 f1	48 89 层	48 89 f3	48 89 f4	48 89 f5	48 89 f6	48 89 层 7
%rdi 公司	48 89 f8	48 89 f9	48 89 英尺	48 89 fb	48 89 fc	48 89 fd	48 89 fe	48 89 ff

B、 popq 指令编码

活动	登记 *R*							
	%rax 公司	%rcx 公司	%rdx 公司	%rbx 公司	%rsp	%限制性商业惯例	%rsi 指数	%rdi 公司
popq*R*	58	59	5a 级	5b	5c 级	5 天	5e	5f 级

C、 movl 指令编码

movl，*SD*

来源 *S*	目的地 *D*							
	%eax 公司	%ecx 公司	%edx 公司	%ebx 公司	%esp	%ebp 公司	%esi 公司	%电子数据交换

%eax 公司	89 c0	89 c1	89 c2	89 c3	89 c4	89 c5	89 c6	89 c7
%ecx 公司	89 c8	89 c9	89 加利福尼亚州	89 cb	89 立方厘米	89 cd	89 ce	89 立方英尺
%edx 公司	89 d0	89 d1	89 d2	89 d3	89 d4	89 d5	89 d6	89 d7
%ebx 公司	89 d8	89 d9	89 da	89 分贝	89 dc	89 日	89 德国	89 df
%esp	89 e0	89 e1	89 e2	89 e3	89 e4	89 e5	89 e6	89 e7
%ebp 公司	89 e8	89 e9	89 个	89 eb	89 欧共体	89 版	89 ee	89 ef
%esi 公司	89 f0	89 f1	89 f2	89 f3	89 f4 层	89 f5	89 层 6	89 楼 7 层
%电子数据交换	89 f8	89 层 9	89 英尺	89 fb	89 fc	89 fd	89 铁	89 ff

D、 2 字节函数 nop 指令的编码

操作员离子	登记 *R*			
	%al 公司	%cl 公司	%dl	%基本法
andb *R*, *R*	20 c0	20 c9	20 d2	20 分贝
圆球 *R*, *R*	08 c0	08 c9	08 d2	08 分贝
cmpb 公司 *R*, *R*	38 c0	38 c9	38 d2	38 分贝
测试 B *R*, *R*	84 c0	84 c9	84 d2	84 分贝

图 3：指令的字节编码。所有值均以十六进制显示。

要解决阶段 5，可以在 rtarget 中由函数 start\u farm 和 end\u farm 划分的代码区域中使用小工具。除了阶段 4 中使用的小工具外，此扩展场还包括不同 movl 指令的编码，如图 3C 所示。场的这一部分中的字节序列还包含用作功能 NOP 的 2 字节指令，即它们不会更改任何寄存器或内存值。这些指令包括图 3D 所示的指令，例如 andb%al，%al，它们对某些寄存器的低位字节进行操作，但不会更改其值。

一些建议：

- 您需要查看 movl 指令对寄存器上部 4 字节的影响，如本文第 183 页所述。
- 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，十进制数字的 ASCII 码是 0x3，字符串的结尾由空字节表示。）x 个x 个

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字符串时，使用换行符分隔其不同部分。HEX2RAW 支持 C 风格的块注释，因此您可以标记利用漏洞字符串的部分。例如：

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

请确保在开头和结尾的注释字符串（“/*”、“*/”）周围留出空间，以便正确忽略注释。

如果在文件漏洞利用中生成十六进制格式的漏洞利用字符串。txt 中，可以通过几种不同的方式将原始字符串应用于 CTARGET 或 RTARGET：

1. 可以设置一系列管道，使管柱通过 HEX2RAW。

   unix>cat 攻击。txt |/hex2raw |/C 目标

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<利用漏洞。txt>利用原始漏洞。txt unix>/ctarget<利用原始资源。txt

   当从 GDB 内运行时，也可以使用此方法：

   unix>gdb ctarget
   （gdb）运行<利用原始资源。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>/hex2raw<利用漏洞。txt>利用原始漏洞。txt unix>/ctarget-我利用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 作为汇编程序，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，假设您编写了一个文件示例。包含以下程序集代码的：

```
#手工生成的汇编代码示例
        pushq 公司$0xabcdef             #将值推送到堆栈上
        addq 公司 17 美元，%rax          #将 17 添加到%rax
        movl 公司 %eax，%edx             #将低位 32 位复制到%edx
```

代码可以包含指令和数据的混合物。“#”字符右侧的任何内容都是注释。
现在可以组装和反汇编此文件：

```
unix>gcc-c 示例。s unix>objdump-d 示例。o>示例。d
```

生成的文件示例。d 包含以下内容：

```
实例 o： 文件格式 elf64-x86-64

节段的拆卸。文本：

0000000000000000 <.文本>：
        0:68 ef cd ab 00 pushq$0xabcdef
        5： 48 83 c0 11 增加$0x11，%rax
        9： 89 c2 mov%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 具有十六进制格式的字节码 68 ef cd ab 00。
从该文件中，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，为目标程序生成输入字符串。。或者，您可以编辑示例。d 省略无关值并包含 C 风格注释以提高可读性，从而产生：

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

这也是一个有效的输入，您可以在发送到一个目标程序之前通过 HEX2RAW。

工具书类

[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）：2:1–2:342012 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。