15-213 年，20xx 年秋季

攻击实验室：了解缓冲溢出虫

9 月 29 日星期二

截止日期：10 月 8 日 11 时 59 分

最后可能的上交时间：10 月 11 日 11 时 59 分

1 导言

此任务涉及对具有不同安全漏洞的两个程序总共生成五个攻击。 你将从这个实验室获得的结果包括：

- 当程序不能很好地保护自己以防止缓冲区溢出时，攻击者可以学习不同的方法来利用安全漏洞。
- 通过这一点，您将更好地理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，以使程序不那么脆弱。
- 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 指令是如何编码的。
- 您将获得更多的调试工具的经验，如 GDB 和 OBJDUMP。

注意 :在本实验室中，您将获得用于利用操作系统和网络服务器中的安全弱点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，并了解这些安全弱点的性质，以便您在编写系统代码时可以避免它们。 我们不允许使用任何其他形式的攻击来获得未经授权的访问任何系统资源。

您将希望研究 CS：APP3E 书的 3.10.3 和 3.10.4 节作为本实验室的参考材料。

2 后勤

和往常一样，这是一个单独的项目。 您将为为您自定义生成的目标程序生成攻击。

2.1 获取文件

您可以通过将网页浏览器指向：

http://$Attacklab：：SERVER_NAME：15513/

$攻击：SERVER_NAME 是运行攻击服务器的机器。 您可以在 atamlab/Attacklab.pm 和 atamlab/src/build/driverhdrs.h 中定义它

服务器将构建您的文件并将它们返回到名为 targetfc 的 tar 文件中。 焦油，其中 k 是目标程序的唯一数目。

注意：构建和下载目标需要几秒钟，所以请耐心等待。

保存 targetfc.. 在（受保护的）Linux 目录中的 tar 文件，您计划在其中执行您的工作。 然后给出命令：tar-xvftargetfc.. 焦油。这将提取包含下面描述的文件的目录 targetfc。

你应该只下载一组文件。 如果出于某种原因下载了多个目标，请选择一个目标来处理并删除其余的目标。

警告：如果您扩展了目标 FC。 在 PC 上的 tar，通过使用 Winzip 这样的实用程序，或者让浏览器进行提取，您将面临在可执行文件上重置权限位的风险。

targetfc 的文件包括：

自述文件。 txt：描述目录内容的文件

carget：易受代码注入攻击的可执行程序

可执行程序易受面向返回的编程攻击

饼干。 txt：一个 8 位十六进制代码，您将在攻击中用作唯一标识符。

农场。 c：目标的“小工具表单”的源代码，您将使用它生成面向返回的编程攻击。

十六进制 2raw：生成攻击字符串的实用程序。

在下面的说明中，我们将假设您已经将文件复制到受保护的本地目录，并且您正在执行该本地目录中的程序。

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摘要。当你第一次阅读这份文件时，这些观点将不太有意义。 在这里，一旦你开始，它们将作为规则的中心参考。

- 您必须在类似于生成目标的机器上执行任务。
- 您的解决方案可能不会使用攻击来规避程序中的验证代码。 具体来说，您将任何地址合并到攻击字符串中供 RET 指令使用，都应该指向以下目的之一：

  —函数 touchl、touch2 或 touch3 的地址。

  —你注入代码的地址

  —你的一个小工具从小工具农场的地址。
- 您只能从文件 r Target 中构造小工具，其地址在功能 start_f ARM 和 end_f ARM 之间。

3 目标方案

CTarget 和 RTarget 都从标准输入读取字符串。 他们这样做的函数 getbuf 定义如下：

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

函数获取类似于标准库函数 gets — it 从标准输入(由‘n’或文件结束终止)读取字符串，并将其（连同空终止符）存储在指定的目的地。 在此代码中，您可以看到目标是数组 buf，声明为具有 BUFFER_SIZE 字节。 在生成目标时，BUFFER_SIZE 是特定于程序版本的编译时常量。

函数获取（)并获取()）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的边界。

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：

统一>/目标

Cookie：0xla7dd803

类型字符串：保持短！

不是剥削。 返回 0x1

正常返回

通常，如果键入长字符串，则会出现错误：

统一>/目标

Cookie：0xla7dd803

类型字符串：这不是一个非常有趣的字符串^但我没有属性...哎哟！你造成了分割错误！

祝你下次好运

(注意，所显示的 cookie 的值将与您的值不同。) 程序 RTarget 将具有相同的行为。 正如错误消息所指示的，溢出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给 CTarget 和 RTarget 的字符串，这样他们就能做更有趣的事情。 这些被称为剥削字符串。

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：不要将结果发送给分级服务器

-i 文件：提供文件输入，而不是标准输入

您的开发字符串通常包含不对应于用于打印字符的 ASCII 值的字节值。 程序 HEX2RAW 将使您能够生成这些原始字符串。 有关如何使用 HEX2RAW 的更多信息，请参见附录 A。

要点：

- 您的开发字符串不能在任何中间位置包含字节值 0x0a，因为这是获取此字节时换行的 ASCII 代码，它将假定您打算终止字符串。
- HEX2RAW 期望用一个或多个空格分隔两位数字的十六进制值。因此，如果要创建一个十六进制值为 0 的字节，则需要将其写入 0。要创建 Oxdeadbeef 这个词，你应该通过你好EF 是广告 d㊀”到 HEX2RAW（注意小端字节排序所需的反转）。

当您正确解决了其中一个级别时，您的目标程序将自动向分级服务器发送通知。 例如：

Unix>./hex2raw<carget。 12.txt/。/ctget

Cookie：0xla7dd803

类型字符串：Touch2！你叫 Touch2(0xla7dd803)

具有目标 c 目标的 2 级有效解决方案

通过：发送利用字符串到服务器进行验证。

尼斯工作！

阶段	方案	级别	方法	功能	要点
1	CTARGET	1	CIC	触摸式	10
2	CTARGET	2	CIC	触摸	25
3	CTARGET	3	CIC	触摸 3	25
4	RTARGET	2	ROP	触摸	35
5	RTARGET	3	ROP	触摸 3	5

CI： 注射

ROP：面向返回的编程

图 1：攻击实验室阶段概述

服务器将测试您的剥削字符串，以确保它真的工作，它将更新 Attacklab 评分板页，指示您的 userid（按匿名目标编号列出）已经完成此阶段。

您可以通过将您的 Web 浏览器指向

http://$Attacklab：：S ERVE R_NAME：15513/成绩板

与炸弹实验室不同，在这个实验室里犯错误是没有惩罚的。 你可以随意用任何你喜欢的弦来点燃 CTarget 和 RTarget。

重要注意事项：您可以在任何 Linux 机器上处理解决方案，但为了提交解决方案，您需要在下列机器上运行：

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

图 1 总结了实验室的五个阶段。 可以看出，前三个涉及对 CTA RGET 的代码注入(C I)攻击，而最后两个涉及对 RTA RGET 的面向返回编程(ROP)攻击。

4 第一部分：代码注入攻击

对于前三个阶段，您的开发字符串将攻击 CTARGET。此程序的设置方式是，堆栈位置将从一次运行到下一次运行保持一致，以便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漏洞字符串包含可执行代码字节编码的攻击。

4.1 1 级

对于第一阶段，您不会注入新代码。 相反，您的开发字符串将重定向程序以执行现有过程。

函数 getbuf 是通过具有以下 C 代码的函数测试在 ctarget 中调用的：

```
1  空隙试验()
2  {
3          英特瓦尔；
4          瓦尔=格布夫()；
5          打印(“没有剥削。 Getbuf 返回 Ox%x\n“，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 c Target 中，有一个函数 touchl 的代码具有以下 C 表示：

```
1  虚空触摸()
2  {
3          五级=1；                          /*部分验证协议大/
4          打印(“Touchl！ 你叫 Touchl()\n“)；
5          验证（1）；
6          退出（0）；
7  }
```

您的任务是在 getbuf 执行其返回语句时，让 ctarget 执行 touchl 的代码，而不是返回测试。 请注意，您的开发字符串也可能损坏与此阶段无关的堆栈部分，但这不会引起问题，因为 touchl 会导致程序直接退出。

一些建议：

- 您为此级别设计开发字符串所需的所有信息都可以通过检查 CTARGET 的拆卸版本来确定。 使用 obj 转储-d 来获取这个已拆卸的版本。
- 其思想是为 touchl 定位起始地址的字节表示，以便 getbuf 代码末尾的 ret 指令将控制转移到 touchl。
- 小心字节排序。
- 您可能希望使用 GDB 通过 getbuf 的最后几个指令来步骤程序，以确保它正在做正确的事情。
- 在 getbuf 的堆栈帧内放置 buf 取决于编译时常数 BUFFER_SIZE 的值，以及 gcc 使用的分配策略.. 您需要检查拆卸的代码以确定其位置。

4.2 二级

第二阶段包括注入少量代码作为开发字符串的一部分。

在文件 c Target 中有一个函数 touch2 的代码，它具有以下 C 表示：

```
1voidtouch2(无符号 val)
```

```
五级=2；                                /*    部分验证协议*
如果(Val==cookie){printf(“Touch2！
        *验证（2）；                            你叫 Touch2(Ox%)。 8x)n“，  val)；
其他{
        printf(“Misfire：失败（2）；            你叫 Touch2(Ox%)。 8x)n“，  val)；
```

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，您必须使它看起来像触摸 2，就好像您已将 cookie 作为其参数传递一样。

一些建议：

- 您将希望以这样的方式定位注入代码的地址的字节表示，在 getbuf 的代码末尾的 ret 指令将控制权转移到它。
- 回想一下，函数的第一个参数是在寄存器%RDI 中传递的。
- 注入的代码应该将寄存器设置为 cookie，然后使用 RET 指令将控件传输到 Touch2 中的第一个指令。
- 不要尝试在您的开发代码中使用 jmp 或调用指令。 这些指令的目的地地址编码很难制定。 对所有控件的传输使用 ret 指令，即使您没有从调用返回。
- 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

第三阶段还涉及代码注入攻击，但将字符串作为参数传递。

在文件中，cTarget 有函数 hexmatch 和 touch3 的代码，具有以下 C 表示：

```
/*将字符串与无符号值*/的十六进制表示进行比较
十六进制匹配(无符号 Val，char*sval)
{
        char cbuf[110]；
         /*使检查字符串的位置不可预测大/
         char*s=cbuf+随机()100%；
        冲刺 f(s，“%。 8x“，  val)；
        返回 strncmp(sval，s，9)==0；
```

```
10
11  空白触摸 3(char*sval)
12  {
13          五级=3；                        /*    部分验证协议*/
14          如果...                                )｛
15                  打印(“Touch3！ ：              你叫 touch3(%s)n，sval)；
16                  验证（3）；
17          其他{                                  你叫 touch3(%s)n，sval)；
18                  (“Misfire：
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 touch3，就好像您已将 cookie 的字符串表示形式作为其参数一样。

一些建议：

- 您需要在您的开发字符串中包含 cookie 的字符串表示形式。字符串应该由八个十六进制数字（从最重要到最不重要的顺序）组成，而没有前导的“Ox”。
- 回想一下，字符串在 C 中表示为字节序列，然后是值为 0 的字节。 类型[你好]在任何 Linux 机器上都可以看到您需要的字符的字节表示。
- 注入的代码应该将寄存器%rdi 设置为此字符串的地址。
- 当函数 hexmatch 和 strncmp 被调用时，它们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，您需要小心放置 cookie 的字符串表示形式的位置。

5 第二部分：面向回报的方案编制

对 RTA RGET 程序进行代码注入攻击比对 CTA RGET 来说要困难得多，因为它使用两种技术来阻止这种攻击：

- 它使用随机化，使堆栈位置在一个运行到另一个运行时有所不同。 这使得无法确定注入的代码将位于何处。
- 它将保持堆栈的内存部分标记为不可执行的，因此即使可以将程序计数器设置为注入代码的开始，程序也会因分割故障而失败。

幸运的是，聪明的人已经设计了策略，通过执行现有的代码而不是注入新的代码来在程序中完成有用的事情。 其中最一般的形式被称为面向返回的编程(ROP)[1，2]，ROP 的策略是在现有程序中识别由一个或多个指令组成的字节序列，然后是指令 RET。 这一部分称为

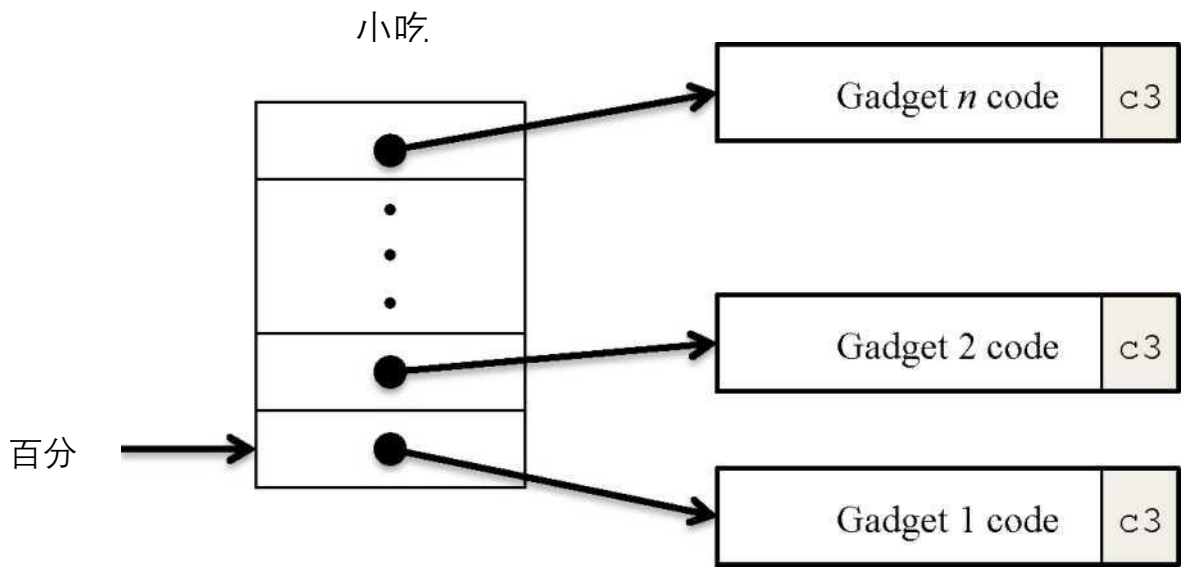

图 2：设置用于执行的小工具序列。 字节值 0xc3 编码 ret 指令。

小玩意。 图 2 说明了如何设置堆栈以执行 n 个小工具序列。 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，最后一个字节为 0xc3，编码 ret 指令。 当程序从这个配置开始执行 ret 指令时，它将启动一个小工具执行链，每个小工具末尾的 ret 指令导致程序跳转到下一个小工具的开始。

小工具可以使用编译器生成的汇编语言语句对应的代码，特别是函数末尾的语句。 在实践中，可能有一些有用的这种形式的小工具，但不足以实现许多重要的操作。 例如，编译后的函数在 ret 之前不太可能将 popq%rdi 作为最后一条指令。 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，一个版本的 rtarget 包含为以下 C 函数生成的代码：

```
无效 setval_210(无符号大 p)
(
        *p=334766306OU；
}
```

这种功能对攻击系统有用的可能性似乎很小。 但是，这个函数的拆卸机器代码显示了一个有趣的字节序列：

```
0000000000400fl5<setval_210>：
    400fl5：        c707d44889c7                移动    $0xc78948d4，(%rdi)
    40Oflb：        c3                          retq
```

字节序列 4889c7 编码指令 movq%rax$_{zz}$ %R DI。(有关有用 movq 指令的编码，请参见图 3A) 这个序列后面是字节值 c3，它编码 ret 指令。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，该小工具将在寄存器%Rax 中复制 64 位值以注册%RDI。

您的 RTA RGET 代码包含许多类似于上面所示的 setval_210 函数的函数，在我们所称的小工具农场区域中。 您的工作将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 和第 3 阶段所做的攻击。

重要事项：小工具农场是由功能 start_f 手臂和 end_f 手臂在您的副本 R Target。 不要试图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，但使用来自小工具农场的小工具执行 RTarget 程序。 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)。

移动：这些代码如图 3A 所示。

这些代码如图 3B 所示。

重试：此指令由单字节 0xc3 编码。

这条指令（发音）[你好] 没有行动，“这是短期的[你好]没有操作，)由单个字节 0x90 编码。 它的唯一效果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由 ARM 和 ARM 函数标定的 rTarget 代码 start_f 区域中找到 mid_f。
- 你只需要两个小工具就可以完成这次攻击。
- 当小工具使用 popq 指令时，它将从堆栈中弹出数据。 因此，您的开发字符串将包含小工具地址和数据的组合。

5.2 3 级

在进入第五阶段之前，请暂停考虑到你迄今所取得的成就。 在第 2 和第 3 阶段，您导致了一个程序来执行您自己设计的机器代码。 如果 CTARGET 是一个网络服务器，您可以将您自己的代码注入远程机器。 在第四阶段，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。 虽然您没有注入您自己的代码，但您可以通过拼接现有代码的序列来注入一种程序类型。 你还得到了 95/100 分的实验室。 这是个好分数。 如果你有其他紧迫的义务，请立即停止。

第 5 阶段要求您对 RTARGET 进行 ROP 攻击，以调用函数 Touch3，指针指向 cookie 的字符串表示形式。 这似乎并不比使用 ROP 攻击调用 touch2 更困难，只是我们已经做到了。 此外，第五阶段只有 5 个点，这不是衡量它所需努力的真正标准。 对于那些想超越对课程的正常期望的人来说，这更像是一个额外的信用问题。

A.Movq 指令的编码

movqs*rr D.D.*

资料来源	目的地 D							
S.S.	rax%	百分比	%rdx	百分比	百分比	%	%	%rdi
rax%	4889c	4889cl	4889c2	4889c3	4889c4	4889c5	4889c6	4889c7
百分比	4889c8	4889c9	4889ca	489CB	4889cc	4889CD	4889 行政长官	4889cf
%rdx	4889d	4889 公升	4889d2	4889d3	4889d4	4889d5	4889d6	4889d7
百分比	4889d8	4889d9	4889da	4889 分贝	4889	4889dd	4889	4889d
百分比	4889eO	4889	4889e2	4889e3	4889e4	4889e5	4889e6	4889e7
%	4889e8	4889e9	489e	4889 退潮	4889c	4889	4889e	4889e
%	4889f	4889 升	4889f2	4889f3	4889f4	4889f5	4889f6	4889f7
%rdi	4889f8	4889f9	4889fa	4889 个 FBI	4889f	4889FD	4889 发	4889ff

B.POPQ 指令的编码

行动	登记册							
	rax%	百分比	%rdx	百分比	百分比	%	%	%rdi
popqr	58	59	5a	5b	5c	5d	5e	5f

C.移动指令的编码

电影 S*rr D.D.*

资料来源	目的地 D							
S.S.	百分比	百分比	百分比	百分比	百分比	百分比	%	%
百分比	89c	89 政	89c2	89c3	89c4	89c5	89c6	89c7
百分比	89c8	89c9	89ca	89CB	89cc	89CD	89 政	89cf
百分比	89d	89dl	89d2	89d3	89d4	89d5	89d6	89d7
百分比	89d8	89d9	89da	89 分贝	89	8,9d	89	89DF
百分比	89eO	89el	89e2	89e3	89e4	89e5	89e6	89e7
百分比	89e8	89e9	89e	89 度	89EC	89	89e	89e
%	89f	89 毫升	89f2	89f3	89f4	89f5	89f6	89f7
%	89f8	89f9	89fa	89 架	89fc	89FD	89 铁	89ff

D.2 字节功能 nop 指令的编码

行动	登记册			
	%	%	%	百分比
和 b*rr R.*	20c	20c9	20d2	20 分贝
或 *R.rr R.*	08c	08c9	08d2	08 分贝
空 *rr R.*	38c	38c9	38d2	38 分贝
试验员 *rr R.*	84c	84c9	84d2	84 分贝

图 3：字节编码指令。 所有值都以十六进制显示。

为了解决第 5 阶段，您可以使用由函数 start_farm 和 end_farm 标定的 rTarget 中代码区域中的小工具。 除了在第四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。农场这一部分中的字节序列也包含作为功能 NOPs 的 2 字节指令，即它们不改变任何寄存器或内存值。 这些包括指令，如图 3D 所示，例如 andb%al，%al，它们对一些寄存器的低阶字节进行操作，但不改变它们的值。

一些建议：

- 您将要查看 movl 指令对寄存器上 4 个字节的影响，正如文本第 183 页所描述的那样。
- 官方解决方案需要八个小工具（并不是所有这些都是唯一的）。

祝你好运，玩得开心！

使用 Hex2raw

HEX2RAW 以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“3031323334350”(回想一下十进制数字 x 的 ASCII 代码是 0x3rr，字符串的末尾由空字节表示)。

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。 我们建议您在工作时使用换行符将您的开发字符串的不同部分分开。 HEX2RAW 支持 C 样式的块注释，因此可以标记开发字符串的部分。 例如：

48c7clf011400/*mov$0x40011f0，%rcx 大/

请务必在开始和结束注释字符串（“/*”、“大/”）周围留出空间，以便注释将被适当忽略。

如果在文件漏洞中生成十六进制格式的漏洞字符串。txt，您可以将原始字符串应用于 CTARGET 或 RTARGET 的几种不同方式：

1. 您可以设置一系列管道来通过 HEX2RAW 传递字符串。

   Unix>猫剥削.txt/。/hex2raw/。/ctget

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<proplication.txt>proplication-raw.txtunix>./ctarget<proplication-raw.txt

   在从 GDB 内部运行时也可以使用这种方法：

Unix>GDBCTarget
(gdb)运行<explain-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>./hex2raw<proplication.txt>proplication-raw.txtunix>。ctget-iplication-raw.txt

这种方法也可以在 GDB 内运行时使用。

B 产生字节码

使用 GCC 作为汇编程序，OBJDUMP 作为拆卸器，可以方便地生成指令序列的字节代码。 例如，假设您编写了一个文件示例。 包含下列程序集代码：

```
手工生成的装配代码示例
推              $0xabc                          #推          堆栈的值
addq            第 17$，%rax                    #加入        17%至%
移动            %eax，%edx                      #副本        低 32 位到%EDX
```

代码可以包含指令和数据的混合。 任何与角色右边的东西都是评论。

你现在可以组装和拆卸这个文件：

unix>gcc-c 示例
unix>objump-d 示例.o>示例.d

生成的文件示例。 d 包括以下内容：

```
例子.o：                        文件格式

拆卸节.文本：
000000000000<文本>：
                0:    68ef CDab00                   推              $Oxabc
                5：   4883c                         添加            $0x11，%rax
                9:    89c2                          电影            %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字，表示指令的起始地址（从 0 开始），而

在‘：’字符后面的十六进制数字表示实例的字节代码。 因此，我们可以看到，指令推送$0x ABCDEF 有十六进制字节代码 68efcdab00。

从这个文件中，您可以得到代码的字节序列：

68ef cd ab004883cO1189c2

然后，该字符串可以通过 HEX2RAW 传递，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 样式的可读性注释，生成：

68ef CDab00	推	$Oxabc	大/
4883c	添加	$0x11，%rax	大/
89c2	电影	%eax，%edx	大/

这也是一个有效的输入，您可以在发送到目标程序之一之前通过 HEX2RAW。

参考资料

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

[2] E.J.Schwartz，T.Avgerinos 和 D.Brumley。 Q：开发硬化变得容易。 在 USEN IX 安全研讨会上，2011 年。