15-213, Fall 20xx
15-213，秋季 20xx

The Attack Lab: Understanding Buffer Overflow Bugs
攻击实验室: 理解缓冲区溢出错误

Assigned: Tue, Sept. 29
作业: 星期二，9 月 29 日

Due: Thu, Oct. 8, 11:59PM EDT
截止日期: 10 月 8 日，星期四，美国东部时间 11:59 PM

Last Possible Time to Turn in: Sun, Oct. 11, 11:59PM EDT
最后一次上床时间: 10 月 11 日，星期日，美国东部时间下午 11:59

1 Introduction
引言

This assignment involves generating a total of five attacks on two programs having different security vul-nerabilities. Outcomes you will gain from this lab include:
这个任务包括对两个具有不同安全可靠性的程序产生总共五个攻击。你将从这个实验室获得的结果包括:

- You will learn different ways that attackers can exploit security vulnerabilities when programs do not safeguard themselves well enough against buffer overflows.
  您将了解当程序不能很好地防止缓冲区溢出时，攻击者可以利用安全漏洞的不同方法。

- Through this, you will get a better understanding of how to write programs that are more secure, as well as some of the features provided by compilers and operating systems to make programs less vulnerable.
  通过这些，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特性，以使程序不那么容易受到攻击。

- You will gain a deeper understanding of the stack and parameter-passing mechanisms of x86-64 machine code.
  您将更深入地理解 x86-64 机器代码的堆栈和参数传递机制。

You will gain a deeper understanding of how x86-64 instructions are encoded.
您将更深入地了解 x86-64 指令是如何编码的。

You will gain more experience with debugging tools such as GDB and OBJDUMP.
您将获得更多使用 GDB 和 OBJDUMP 等调试工具的经验。

Note: In this lab, you will gain firsthand experience with methods used to exploit security weaknesses in operating systems and network servers. Our purpose is to help you learn about the runtime operation of programs and to understand the nature of these security weaknesses so that you can avoid them when you write system code. We do not condone the use of any other form of attack to gain unauthorized access to any system resources.
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，并了解这些安全缺陷的本质，以便您在编写系统代码时能够避免它们。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的访问。

You will want to study Sections 3.10.3 and 3.10.4 of the CS:APP3e book as reference material for this lab.
你需要学习 CS: APP3e 书的 3.10.3 和 3.10.4 部分作为本实验室的参考资料。

2 Logistics
物流

As usual, this is an individual project. You will generate attacks for target programs that are custom gener-ated for you.
像往常一样，这是一个单独的项目。您将为自定义的目标程序生成攻击。

2.1 Getting Files
2.1 获取文件

You can obtain your files by pointing your Web browser at:
你可以通过将你的网页浏览器指向以下地址来获取你的文件:

http://$Attacklab::SERVER_NAME:15513/
Http://$attacklab::server_name:15513/

INSTRUCTOR: $Attacklab::SERVER_NAME is the machine that runs the attacklab servers. You define it in attacklab/Attacklab.pm and in attacklab/src/build/driverhdrs.h
讲师: $Attacklab: : SERVER _ name 是运行攻击实验室服务器的机器，您可以在攻击实验室/攻击实验室. pm 和攻击实验室/src/build/driverhdrs.h 中定义它

The server will build your files and return them to your browser in a tar file called targetk.tar, where k is the unique number of your target programs.
服务器将构建您的文件，并将它们返回到您的浏览器中的 tar 文件 targetk.tar 中，其中 k 是目标程序的唯一编号。

Note: It takes a few seconds to build and download your target, so please be patient.
注意: 建立和下载你的目标需要几秒钟，所以请耐心等待。

Save the targetk.tar file in a (protected) Linux directory in which you plan to do your work. Then give the command: tar -xvf targetk.tar. This will extract a directory targetk containing the files described below.
将 targetk.tar 文件保存在(受保护的) Linux 目录中，您计划在该目录中执行工作。然后输入命令: tar-xvf targetk.tar。这将提取包含下面描述的文件的目录 targetk。

You should only download one set of files. If for some reason you download multiple targets, choose one target to work on and delete the rest.
您应该只下载一组文件。如果由于某种原因下载了多个目标，请选择一个目标进行处理，然后删除其余目标。

Warning: If you expand your targetk.tar on a PC, by using a utility such as Winzip, or letting your browser do the extraction, you'll risk resetting permission bits on the executable files.
警告: 如果你在 PC 上展开 targetk.tar，使用 Winzip 这样的工具，或者让浏览器进行提取，你将冒险重置可执行文件的权限位。

The files in targetk include:

目标文件包括:

README.txt: A file describing the contents of the directory
README.txt: 描述目录内容的文件

ctarget: An executable program vulnerable to code-injection attacks
Ctarget: 一个容易受到代码注入攻击的可执行程序

rtarget: An executable program vulnerable to return-oriented-programming attacks
Rtarget: 一个可执行程序，容易受到面向返回的编程攻击

cookie.txt: An 8-digit hex code that you will use as a unique identifier in your attacks.
Txt: 一个 8 位十六进制代码，你可以用它作为攻击的唯一标识符。

farm.c: The source code of your target's "gadget farm," which you will use in generating return-oriented programming attacks.
C: 目标的"小工具场"的源代码，你可以用它来产生 Return-to-libc 攻击攻击。

hex2raw: A utility to generate attack strings.
Hex2raw: 生成攻击字符串的工具。

In the following instructions, we will assume that you have copied the files to a protected local directory, and that you are executing the programs in that local directory.
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，并且您正在该本地目录中执行程序。

2.2 Important Points
2.2 重点

Here is a summary of some important rules regarding valid solutions for this lab. These points will not make much sense when you read this document for the first time. They are presented here as a central reference of rules once you get started.
下面是关于本实验室有效解决方案的一些重要规则的总结。当您第一次阅读本文档时，这些要点将没有多大意义。一旦开始，它们就作为规则的中心参考呈现在这里。

You must do the assignment on a machine that is similar to the one that generated your targets.
必须在与生成目标的机器类似的机器上执行任务。

Your solutions may not use attacks to circumvent the validation code in the programs. Specifically, any address you incorporate into an attack string for use by a ret instruction should be to one of the following destinations:
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，任何地址，你纳入一个攻击字符串使用一个秘密指令应该是以下目的地之一:

– The addresses for functions touch1, touch2, or touch3.
- touch1、 touch2 或 touch3 函数的地址。

– The address of your injected code
- 注入代码的地址

– The address of one of your gadgets from the gadget farm.
- 你的一个小工具的地址从小工具场。

You may only construct gadgets from file rtarget with addresses ranging between those for func-tions start_farm and end_farm.
您只能从文件 rtarget 构造 gadgets，其地址在 func-tions start _ farm 和 end _ farm 之间。

3 Target Programs
目标项目

Both CTARGET and RTARGET read strings from standard input. They do so with the function getbuf defined below:
和 RTARGET 都从标准输入中读取字符串，它们使用下面定义的函数 getbuf 进行读取:

```
1  unsigned getbuf()
1 unsigned getbuf ()
2  {
3          char buf[BUFFER_SIZE];
查找缓冲区大小;
4          Gets(buf);
得(buf) ;
5return 1;
5return 1;
6  }
```

The function Gets is similar to the standard library function gets—it reads a string from standard input (terminated by '\n' or end-of-file) and stores it (along with a null terminator) at the specified destination. In this code, you can see that the destination is an array buf, declared as having BUFFER_SIZE bytes. At the time your targets were generated, BUFFER_SIZE was a compile-time constant specific to your version of the programs.
Gets 函数类似于标准库函数 Gets 一它从标准输入读取一个字符串(以" n"或文件结束符结束)，并将其(连同一个空终止符)存储在指定的目的地。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 _ 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。

Functions Gets() and gets() have no way to determine whether their destination buffers are large enough to store the string they read. They simply copy sequences of bytes, possibly overrunning the bounds of the storage allocated at the destinations.
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，可能会超出在目标上分配的存储的边界。

If the string typed by the user and read by getbuf is sufficiently short, it is clear that getbuf will return 1, as shown by the following execution examples:
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，如下面的执行示例所示:

```
unix> ./ctarget
Unix > ./ctarget
```

Cookie: 0x1a7dd803
0x1a7dd803
Type string: Keep it short!
键入字符串: 保持简短！
No exploit. Getbuf returned 0x1
没有利用价值，Getbuf 返回 0x1
Normal return
正常回报

Typically an error occurs if you type a long string:
如果键入长字符串，通常会发生错误:

unix> ./ctarget
Unix > ./ctarget
Cookie: 0x1a7dd803
0x1a7dd803
Type string: This is not a very interesting string, but it has the property ...
Type string: 这不是一个非常有趣的字符串，但它具有... 属性。
Ouch!: You caused a segmentation fault!
哎哟! : 你引起了内存区段错误！
Better luck next time
祝你下次好运

(Note that the value of the cookie shown will differ from yours.) Program RTARGET will have the same behavior. As the error message indicates, overrunning the buffer typically causes the program state to be corrupted, leading to a memory access error. Your task is to be more clever with the strings you feed CTARGET and RTARGET so that they do more interesting things. These are called exploit strings.
(请注意，所显示的 cookie 的值将与您的不同。)程序 RTARGET 将具有相同的行为。正如错误消息所指出的，超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。您的任务是更聪明地处理提供给 CTARGET 和 RTARGET 的字符串，这样它们就能做更多有趣的事情。这些被称为利用字符串。

Both CTARGET and RTARGET take several different command line arguments:
和 RTARGET 都使用几个不同的命令行参数:

-h: Print list of possible command line arguments
- h: 打印可能的命令行参数列表

-q: Don’t send results to the grading server
- q: 不要将结果发送到评分服务器

-i FILE: Supply input from a file, rather than from standard input
I FILE: 从文件提供输入，而不是从标准输入

Your exploit strings will typically contain byte values that do not correspond to the ASCII values for printing characters. The program HEX2RAW will enable you to generate these raw strings. See Appendix A for more information on how to use HEX2RAW.
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 程序将允许您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。

Important points:
重点:

Your exploit string must not contain byte value 0x0a at any intermediate position, since this is the ASCII code for newline (‘\n’). When Gets encounters this byte, it will assume you intended to terminate the string.
利用字符串在任何中间位置都不能包含字节值 0x0a，因为这是换行符(‘ n’)的 ASCII 码。当遇到此字节时，它将假定您打算终止该字符串。

HEX2RAW expects two-digit hex values separated by one or more white spaces. So if you want to create a byte with a hex value of 0, you need to write it as 00. To create the word 0xdeadbeef you should pass “ef be ad de” to HEX2RAW (note the reversal required for little-endian byte ordering).
HEX2RAW 期望两位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。要创建单词 0xdeadbeef，您应该将“ ef be ad de”传递给 HEX2RAW (注意 little-endian 字节排序所需的逆转)。

When you have correctly solved one of the levels, your target program will automatically send a notification to the grading server. For example:
当你正确地解决了其中一个水平，你的目标程序将自动发送一个通知给分级服务器。例如:

unix> ./hex2raw < ctarget.l2.txt | ./ctarget Cookie:
0x1a7dd803
Unix > ./hex2raw < ctarget.l2.txt | ./ctarget Cookie:
0x1a7dd803
Type string:Touch2!: You called touch2(0x1a7dd803)
输入 string: Touch2! : You called Touch2(0x1a7dd803)
Valid solution for level 2 with target ctarget
带有目标 ctarget 的级别 2 的有效解决方案
PASSED: Sent exploit string to server to be validated.
传递: 将利用字符串发送到服务器进行验证。
NICE JOB!
干得好！

Phase 第二阶段	Program 计划	Level 水平	Method 方法	Function 功能	Points 积分
1	CTARGET CTARGET CTARGET	1	CI 国际竞争力	touch1 触摸	10
2	CTARGET CTARGET CTARGET	2	CI 国际竞争力	touch2 触摸 2	25
3	CTARGET CTARGET	3	CI 国际竞争力	touch3 触摸 3	25
4	RTARGET RTARGET RTARGET	2	ROP 机械钻头	touch2 触摸 2	35
5	RTARGET RTARGET	3	ROP 机械钻头	touch3 触摸 3	5

CI:
主持人: Code injection
代码注入

ROP:
原产地规划: Return-oriented programming
Return-to-libc 攻击

Figure 1: Summary of attack lab phases
图 1: 攻击实验阶段的总结

The server will test your exploit string to make sure it really works, and it will update the Attacklab score-board page indicating that your userid (listed by your target number for anonymity) has completed this phase.
服务器将测试您的利用字符串，以确保它真正的工作，它将更新攻击实验室分数板页面，指出您的用户标识(列出您的目标数字为匿名)已经完成了这一阶段。

You can view the scoreboard by pointing your Web browser at
你可以通过将你的网页浏览器指向

http://$Attacklab::SERVER_NAME:15513/scoreboard
Http://$attacklab::server_name:15513/scoreboard

Unlike the Bomb Lab, there is no penalty for making mistakes in this lab. Feel free to fire away at CTARGET and RTARGET with any strings you like.

与炸弹实验室不同，在这个实验室里犯错是不会受到惩罚的。请随意使用任何你喜欢的字符串向 CTARGET 和 RTARGET 开火。

IMPORTANT NOTE: You can work on your solution on any Linux machine, but in order to submit your solution, you will need to be running on one of the following machines:
重要提示: 你可以在任何一台 Linux 机器上运行你的解决方案，但是为了提交你的解决方案，你需要在以下机器上运行:

INSTRUCTOR: Insert the list of the legal domain names that you established in buflab/src/config.c.
讲师: 插入您在 buflab/src/config.c 中建立的合法域名列表。

Figure 1 summarizes the five phases of the lab. As can be seen, the first three involve code-injection (CI) attacks on CTARGET, while the last two involve return-oriented-programming (ROP) attacks on RTARGET.
图 1 总结了实验室的五个阶段。可以看出，前三种攻击涉及对 CTARGET 的代码注入(CI)攻击，而后两种攻击涉及对 RTARGET 的面向返回的编程(ROP)攻击。

4 Part I: Code Injection Attacks
第一部分: 代码注入攻击

For the first three phases, your exploit strings will attack CTARGET. This program is set up in a way that the stack positions will be consistent from one run to the next and so that data on the stack can be treated as executable code. These features make the program vulnerable to attacks where the exploit strings contain the byte encodings of executable code.
在前三个阶段，您的开发字符串将攻击 CTARGET。这个程序的设置方式是，从一次运行到下一次运行，堆栈的位置将保持一致，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，因为利用字符串包含可执行代码的字节编码。

4.1 Level 1
4.1 第 1 层

For Phase 1, you will not inject new code. Instead, your exploit string will redirect the program to execute an existing procedure.
对于第 1 阶段，您将不会注入新代码。相反，利用字符串将重定向程序以执行现有的过程。

Function getbuf is called within CTARGET by a function test having the following C code:
函数 getbuf 通过以下 c 代码的函数测试在 CTARGET 中调用:

```
1 void test()
1 孔洞测试()
2 {
3       int val;
整数值;
4       val = getbuf();
1.1.1.1.1.1.1.1.1.1.2.1.2.1.1.2.1.2.1.2.1.2.1.2.1.2.1.2.1.2.2
5       printf("No exploit.  Getbuf returned 0x%x\n", val);
Printf (“ No exploit. Getbuf returns 0x% xn” ， val) ;
6 }
```

When getbuf executes its return statement (line 5 of getbuf), the program ordinarily resumes execution within function test (at line 5 of this function). We want to change this behavior. Within the file ctarget, there is code for a function touch1 having the following C representation:
当 getbuf 执行它的 return 语句(getbuf 的第 5 行)时，程序通常会在函数测试中恢复执行(这个函数的第 5 行)。我们想要改变这种行为。在文件 ctarget 中，有一个函数 touch1 的代码，该函数具有以下 c 表示:

```
1 void touch1()
1 void touch1()
2 {
3       vlevel = 1;/* Part of validation protocol */
Vlevel = 1;/* 验证协议的一部分 */
4printf("Touch1!: You called touch1()\n");
4printf (“ Touch1! : You called Touch1() n”) ;
5       validate(1);
(1) ;
6       exit(0);
出口(0) ;
7 }
```

Your task is to get CTARGET to execute the code for touch1 when getbuf executes its return statement, rather than returning to test. Note that your exploit string may also corrupt parts of the stack not directly related to this stage, but this will not cause a problem, since touch1 causes the program to exit directly.
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，利用字符串还可能损坏与这个阶段没有直接关系的堆栈部分，但这不会造成问题，因为 touch1 会导致程序直接退出。

Some Advice:
一些建议:

All the information you need to devise your exploit string for this level can be determined by examining a disassembled version of CTARGET. Use objdump -d to get this dissembled version.
所有你需要为这个级别设计开发字符串的信息都可以通过测试一个反汇编版本的 CTARGET 来确定。使用 objdump-d 获得这个伪装版本。

The idea is to position a byte representation of the starting address for touch1 so that the ret instruction at the end of the code for getbuf will transfer control to touch1.

其思想是定位 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。

Be careful about byte ordering.
注意字节排序。

You might want to use GDB to step the program through the last few instructions of getbuf to make sure it is doing the right thing.
您可能希望使用 GDB 逐步执行 getbuf 的最后几条指令，以确保程序正在执行正确的操作。

The placement of buf within the stack frame for getbuf depends on the value of compile-time constant BUFFER_SIZE, as well the allocation strategy used by GCC. You will need to examine the disassembled code to determine its position.
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。您将需要检查反汇编的代码以确定其位置。

4.2 Level 2
4.2 第二层

Phase 2 involves injecting a small amount of code as part of your exploit string.
阶段 2 包括注入少量代码作为开发字符串的一部分。

Within the file ctarget there is code for a function touch2 having the following C representation:
在文件 ctarget 中有一个函数 touch2 的代码，该函数具有以下 c 表示:

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

```
2  {
3          vlevel = 2;/* Part of validation protocol */
Vlevel = 2;/* 验证协议的一部分 */
4if (val == cookie) {
4if (val = cookie){
5printf("Touch2!: You called touch2(0x%.8x)\n", val);
5printf (“ Touch2! : You called Touch2(0x% . 8 x) n”， val) ;
6                  validate(2);
(2) ;
7} else {
7} else {
8printf("Misfire: You called touch2(0x%.8x)\n", val);
8printf (“ Misfire: You called touch2(0x% . 8 x) n”， val) ;
9                  fail(2);
失败(2) ;
10         }
11         exit(0);
出口(0) ;
12  }
```

Your task is to get CTARGET to execute the code for touch2 rather than returning to test. In this case, however, you must make it appear to touch2 as if you have passed your cookie as its argument.
您的任务是让 CTARGET 执行 touch2 的代码，而不是返回到 test。但是，在这种情况下，您必须使它看起来像 touch2，好像已经将 cookie 作为参数传递给了它。

Some Advice:
一些建议:

> You will want to position a byte representation of the address of your injected code in such a way that ret instruction at the end of the code for getbuf will transfer control to it.
> 您需要定位注入代码地址的字节表示形式，以便 getbuf 代码末尾的 ret 指令将控制权传递给它。

Recall that the first argument to a function is passed in register %rdi.
回想一下，函数的第一个参数是在寄存器% rdi 中传递的。

> Your injected code should set the register to your cookie, and then use a ret instruction to transfer control to the first instruction in touch2.
> 注入的代码应该将寄存器设置为 cookie，然后使用 ret 指令将控制权转移到 touch2 中的第一条指令。

> Do not attempt to use jmp or call instructions in your exploit code. The encodings of destination addresses for these instructions are difficult to formulate. Use ret instructions for all transfers of control, even when you are not returning from a call.
> 不要试图在你的利用代码中使用 jmp 或调用指令。这些指令的目的地址编码很难公式化。对所有的控制转移使用可靠的指令，即使你没有从一个呼叫返回。

> See the discussion in Appendix B on how to use tools to generate the byte-level representations of instruction sequences.
> 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 Level 3
4.3 第 3 层

Phase 3 also involves a code injection attack, but passing a string as argument.
阶段 3 还涉及代码注入攻击，但是传递一个字符串作为参数。

Within the file ctarget there is code for functions hexmatch and touch3 having the following C representations:
在文件 ctarget 中有 hexmatch 和 touch3 函数的代码，它们具有以下 c 表示:

```
1  /* Compare string to hex represention of unsigned value */
1/* 比较字符串与 unsigned value */的十六进制表示
2  int hexmatch(unsigned val, char *sval)
2 int hexmatch (unsigned val，char * sval)
3  {
4          char cbuf[110];
[110] ;
5          /* Make position of check string unpredictable */
/* 使检查字符串的位置不可预测 */
6char *s = cbuf + random() % 100;
6char * s = cbuf + random ()% 100;
7sprintf(s, "%.8x", val);
7sprintf (s，"% . 8 x" ，val) ;
8return strncmp(sval, s, 9) == 0;
8return strncmp (sval，s，9) = = 0;
9  }
```

10
图 10

```
11 void touch3(char *sval)
11 void touch3(char * sval)
12 {
13          vlevel = 3;/* Part of validation protocol */
Vlevel = 3;/* 验证协议的一部分 */
14          if (hexmatch(cookie, sval)) {
如果(hexmatch (cookie，sval)){
15                  printf("Touch3!: You called touch3(\"%s\")\n", sval);
Printf (“ Touch3! : You called Touch3(”% s”) n”， sval) ;
16                  validate(3);
(3) ;
17          } else {
1
18                  printf("Misfire: You called touch3(\"%s\")\n", sval);
Printf (“ Misfire: You called touch3(”% s”) n”， sval) ;
19                  fail(3);
失败(3) ;
20          }
21          exit(0);
出口(0) ;
22 }
```

Your task is to get CTARGET to execute the code for touch3 rather than returning to test. You must make it appear to touch3 as if you have passed a string representation of your cookie as its argument.
您的任务是让 CTARGET 执行 touch3 的代码，而不是返回到 test。你必须让它看起来像 touch3，好像你已经传递了你的 cookie 的字符串表示作为它的参数。

Some Advice:
一些建议:

You will need to include a string representation of your cookie in your exploit string. The string should consist of the eight hexadecimal digits (ordered from most to least significant) without a leading “0x.”
您需要在利用字符串中包含 cookie 的字符串表示形式。字符串应该由八个十六进制数字组成(从最大到最小有效位排序)，前面没有前导的“0x”

Recall that a string is represented in C as a sequence of bytes followed by a byte with value 0. Type “man ascii” on any Linux machine to see the byte representations of the characters you need.
回想一下，字符串在 c 中表示为一个字节序列，后跟一个字节，值为 0。在任何 Linux 机器上键入“ manascii”，以查看所需字符的字节表示形式。

Your injected code should set register %rdi to the address of this string.
注入的代码应该将寄存器% rdi 设置为此字符串的地址。

When functions hexmatch and strncmp are called, they push data onto the stack, overwriting portions of memory that held the buffer used by getbuf. As a result, you will need to be careful where you place the string representation of your cookie.
当函数 hexmatch 和 strncmp 被调用时，它们将数据推送到堆栈上，覆盖存放 getbuf 使用的缓冲区的内存部分。因此，您需要注意将 cookie 的字符串表示放在何处。

5 Part II: Return-Oriented Programming
第二部分: Return-to-libc 攻击

Performing code-injection attacks on program RTARGET is much more difficult than it is for CTARGET, because it uses two techniques to thwart such attacks:
对程序 RTARGET 执行代码注入攻击要比 CTARGET 困难得多，因为它使用两种技术来阻止这种攻击:

- It uses randomization so that the stack positions differ from one run to another. This makes it impos-sible to determine where your injected code will be located.
  它使用随机化，以便堆栈的位置不同的运行另一个。这使得无法确定注入的代码将位于何处。
- It marks the section of memory holding the stack as nonexecutable, so even if you could set the program counter to the start of your injected code, the program would fail with a segmentation fault.
  它将保存堆栈的内存区域标记为非可执行区域，因此，即使你可以将程序计数器设置为注入代码的开始，程序也会失败，并且会有一个内存区段错误。

Fortunately, clever people have devised strategies for getting useful things done in a program by executing existing code, rather than injecting new code. The most general form of this is referred to as return-oriented programming (ROP) [1, 2]. The strategy with ROP is to identify byte sequences within an existing program that consist of one or more instructions followed by the instruction ret. Such a segment is referred to as a
幸运的是，聪明的人已经设计出了一些策略，通过执行现有代码而不是注入新代码来完成程序中有用的事情。这种情况最普遍的形式被称为 Return-to-libc 攻击。ROP 的策略是在现有程序中识别字节序列，该程序由一个或多个指令组成，后面跟着指令 ret。这样的一个片段被称为

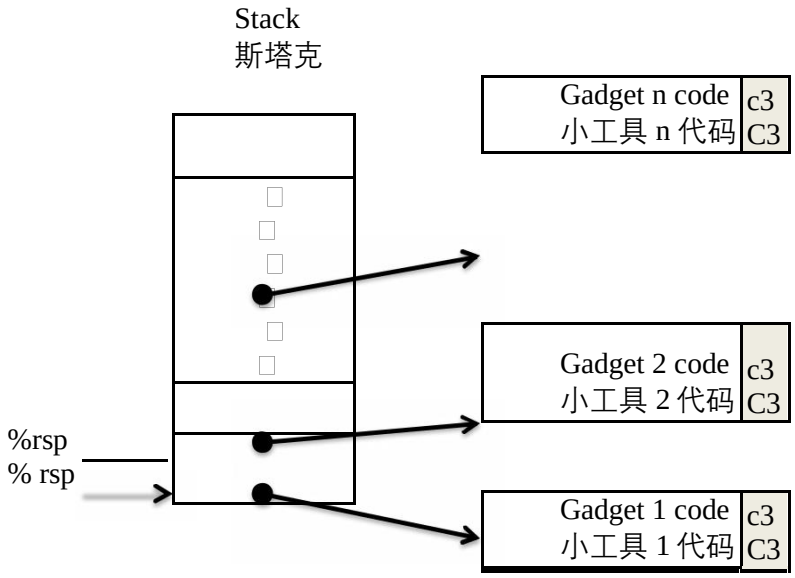

Figure 2: Setting up sequence of gadgets for execution. Byte value 0xc3 encodes the ret instruction.
图 2: 设置小工具的执行顺序。字节值 0xc3 对密码指令进行编码。

gadget. Figure 2 illustrates how the stack can be set up to execute a sequence of n gadgets. In this figure, the stack contains a sequence of gadget addresses. Each gadget consists of a series of instruction bytes, with the final one being 0xc3, encoding the ret instruction. When the program executes a ret instruction starting with this configuration, it will initiate a chain of gadget executions, with the ret instruction at the end of each gadget causing the program to jump to the beginning of the next.
小工具。图 2 说明了如何设置堆栈来执行 n 个小工具的序列。在此图中，堆栈包含一系列小部件地址。每个 gadget 由一系列指令字节组成，最后一个字节是 0xc3，对 ret 指令进行编码。当程序执行以此配置开始的 ret 指令时，它将启动一系列 gadget 执行，每个 gadget 末尾的 ret 指令导致程序跳到下一个 gadget 的开始。

A gadget can make use of code corresponding to assembly-language statements generated by the compiler, especially ones at the ends of functions. In practice, there may be some useful gadgets of this form, but not enough to implement many important operations. For example, it is highly unlikely that a compiled function would have popq %rdi as its last instruction before ret. Fortunately, with a byte-oriented instruction set, such as x86-64, a gadget can often be found by extracting patterns from other parts of the instruction byte sequence.
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。实际上，可能有一些这种形式的有用的小工具，但是不足以实现许多重要的操作。例如，一个已编译的函数不太可能将 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 gadget。

For example, one version of rtarget contains code generated for the following C function:
例如，一个版本的 rtarget 包含为下面的 c 函数生成的代码:

```
void setval_210(unsigned *p)
210(unsigned * p)
{
```

```
    *p = 3347663060U;
    * p = 3347663060U;
}
```

The chances of this function being useful for attacking a system seem pretty slim. But, the disassembled machine code for this function shows an interesting byte sequence:
这个函数用于攻击系统的可能性似乎很小。但是，这个函数的分解机器代码显示了一个有趣的字节序列:

```
0000000000400f15 <setval_210>:
000000000400f15 < setval _ 210 > > :
400f15:    c7    07 d4 48 89 c7    movl    $0xc78948d4,(%rdi)
400f15:    C7    07 d44889 C7      移动    $0xc78948d4，(% rdi)
400f1b:    c3                      retq
400f1b:    C3                      Retq
```

The byte sequence 48 89 c7 encodes the instruction movq %rax, %rdi. (See Figure 3A for the encodings of useful movq instructions.) This sequence is followed by byte value c3, which encodes the ret instruction. The function starts at address 0x400f15, and the sequence starts on the fourth byte of the function. Thus, this code contains a gadget, having a starting address of 0x400f18, that will copy the 64-bit value in register %rax to register %rdi.
字节序列 4889 c7 编码指令 movq% rax,% rdi。(有用的 movq 指令的编码见图 3A。)这个序列后面是字节值 c3，它对 ret 指令进行了编码。函数从地址 0x400f15 开始，序列从函数的第四个字节开始。因此，这段代码包含一个小工具，其起始地址为 0x400f18，它将在 register% rax 中复制 64 位值以注册% rdi。

Your code for RTARGET contains a number of functions similar to the setval_210 function shown above in a region we refer to as the gadget farm. Your job will be to identify useful gadgets in the gadget farm and use these to perform attacks similar to those you did in Phases 2 and 3.
RTARGET 的代码包含许多类似于上面显示的 setval _ 210 函数的函数，该函数位于我们称为小工具场的区域中。你的工作将是识别小工具场中有用的小工具，并使用它们执行类似于第 2 和第 3 阶段的攻击。

Important: The gadget farm is demarcated by functions start_farm and end_farm in your copy of rtarget. Do not attempt to construct gadgets from other portions of the program code.
重要提示: 在 rtarget 的副本中，gadget farm 由 start _ farm 和 end _ farm 函数划分。不要试图从程序代码的其他部分构造 gadget。

5.1 Level 2
5.1 第二层

For Phase 4, you will repeat the attack of Phase 2, but do so on program RTARGET using gadgets from your gadget farm. You can construct your solution using gadgets consisting of the following instruction types, and using only the first eight x86-64 registers (%rax–%rdi).
对于第 4 阶段，您将重复第 2 阶段的攻击，但是使用您的小工具场中的小工具在程序 RTARGET 上进行。您可以使用由以下指令类型组成的 gadget 构造您的解决方案，并且只使用前八个 x86-64 寄存器(% rax -% rdi)。

movq : The codes for these are shown in Figure 3A.
Movq: 这些代码如图 3A 所示。

popq : The codes for these are shown in Figure 3B.
Popq: 这些代码如图 3B 所示。

ret : This instruction is encoded by the single byte 0xc3.
Ret: 此指令由单个字节 0xc3 编码。

nop : This instruction (pronounced “no op,” which is short for “no operation”) is encoded by the single byte 0x90. Its only effect is to cause the program counter to be incremented by 1.
Nop: 这条指令(发音为“ no op” ，是“ no operation”的缩写)由单个字节 0x90 编码。它的唯一效果是使程序计数器增加 1。

Some Advice:
一些建议:

All the gadgets you need can be found in the region of the code for rtarget demarcated by the functions start_farm and mid_farm.
所有你需要的小工具都可以在 rtarget 的代码区域中找到，这些区域由函数 start farm 和 mid farm 划分。

You can do this attack with just two gadgets.
你只需要两个小工具就可以完成这个攻击。

When a gadget uses a popq instruction, it will pop data from the stack. As a result, your exploit string will contain a combination of gadget addresses and data.
当小工具使用 popq 指令时，它将从堆栈中弹出数据。因此，利用字符串将包含小工具地址和数据的组合。

5.2 Level 3
5.2 第三层

Before you take on the Phase 5, pause to consider what you have accomplished so far. In Phases 2 and 3, you caused a program to execute machine code of your own design. If CTARGET had been a network server, you could have injected your own code into a distant machine. In Phase 4, you circumvented two of the main devices modern systems use to thwart buffer overflow attacks. Although you did not inject your own code, you were able inject a type of program that operates by stitching together sequences of existing code. You have also gotten 95/100 points for the lab. That's a good score. If you have other pressing obligations consider stopping right now.
在你开始第五阶段之前，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 是一个网络服务器，那么您可以将自己的代码注入到一台远程机器中。在第 4 阶段，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。虽然没有注入自己的代码，但是可以注入一种通过拼接现有代码序列来操作的程序。你的实验室也得了 95/100 分。这分数不错。如果你还有其他紧迫的义务，考虑现在就停止。

Phase 5 requires you to do an ROP attack on RTARGET to invoke function touch3 with a pointer to a string representation of your cookie. That may not seem significantly more difficult than using an ROP attack to invoke touch2, except that we have made it so. Moreover, Phase 5 counts for only 5 points, which is not a true measure of the effort it will require. Think of it as more an extra credit problem for those who want to go beyond the normal expectations for the course.
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 touch3 函数。这似乎并不比使用 ROP 攻击来调用 touch2 困难得多，只是我们已经这样做了。此外，第五阶段只计算了 5 分，这并不能真正衡量它所需要的努力。对于那些想要超越正常期望的课程的学生来说，这更像是一个额外的学分问题。

A. Encodings of movq instructions
移动指令的编码

movq S, D
移动 s，d

Source 资料来源 S	Destination D 目的地 d							
	%rax % rax	%rcx % rcx	%rdx % rdx	%rbx % rbx	%rsp % rsp	%rbp % rbp	%rsi % rsi	%rdi % 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
% rax								
%rcx	48 89 C8	48 89 C9	48 89 Ca	48 89 cb “cb”的复数	48 89 Cc	48 89 Cd	48 89 Ce	48 89 Cf
% rcx								
%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
% rdx								
%rbx	48 89 D8	48 89 D9	48 89 da 爸爸	48 89 db 分贝	48 89 dc 直流电	48 89 dd	48 89 de 德	48 89 df
% rbx								
%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
% rsp								
%rbp	48 89 E8	48 89 E9	48 89 ea	48 89 eb 电子束	48 89 ec 欧共体	48 89 ed 埃德	48 89 ee 看	48 89 ef 英孚
% rbp								
%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
% rsi								
%rdi	48 89 f8	48 89 f9	48 89 fa 发	48 89 fb	48 89 fc 男名男子名	48 89 fd 消防局	48 89 fe 生命	48 89 ff
% rdi								

B. Encodings of popq instructions
电子指令的编码

Operation 操作	Register R 登记册 r							
	%rax % rax	%rcx % rcx	%rdx % rdx	%rbx % rbx	%rsp % rsp	%rbp % rbp	%rsi % rsi	%rdi % rdi

popq R 泡泡	58	59	5a 5a	5b 5b	5c 5c	5d 5d	5e 5e	5f 5f

C. Encodings of movl instructions
运动指令的编码

movl S, D
S，d

Source 资料来源 S	Destination D 目的地 d							
	%eax X	%ecx X	%edx X	%ebx X	%esp % esp	%ebp P	%esi % esi	%edi 我
%eax X	c0 89 C0	c1 89 C1	c2 89 C2	c3 89 C3 cb “ c b” 的	c4 89 C4	c5 89 C5	c6 89 C6	c7 89 C7
%ecx X	c8 89 C8	c9 89 C9	ca 89 Ca	复 89 数	cc 89 Cc	cd 89 Cd	ce 89 Ce	cf 89 Cf
%edx X	d0 89 D0	d1 89 D1	d2 89 D2	d3 89 D3	d4 89 D4	d5 89 D5	d6 89 D6	d7 89 D7
%ebx X	d8 89 D8	d9 89 D9	da 爸 89 爸	db 分 89 贝	dc 直 流 89 电	dd 89 Dd	de 89 德	df 89 Df
%esp % esp	e0 89 E0	e1 89 E1	e2 89 E2	e3 89 E3	e4 89 E4	e5 89 E5	e6 89 E6	e7 89 E7
%ebp P	e8 89 E8	e9 89 E9	ea 89 Ea	eb 电 子 89 束	ec 欧 共 89 体	ed 埃 89 德	ee 89 看	ef 英 89 孚
%esi % esi	f0 89 0	f1 89 F1	f2 89 F2	f3 89 F3	f4 89 F4	f5 89 F5	f6 89 F6	f7 89 F7
%edi 我	f8 89 F8	f9 89 F9	fa 89 发	fb 89 Fb	fc 男 名 男 子 89 名	fd 消 防 89 局	fe 生 89 命	ff 89 Ff

D. Encodings of 2-byte functional nop instructions
2 字节函数 nop 指令的编码

Operation 操作		Register R 登记册 r			
		%al % al	%cl % cl	%dl % dl	%bl % bl
andb 还有	R, R R，r	c0 20 C0	c9 20 C9	d2 20 D2	db 分 20 贝
orb 圆球	R, R R，r	c0 08 C0	c9 08 C9	d2 08 D2	db 分 08

cmpb Cmpb	R, R R，r	c0 38 C0	c9 38 C9	d2 38 D2	贝 db 分 38 贝
testb 测试	R, R R，r	c0 84 C0	c9 84 C9	d2 84 D2	db 分 84 贝

Figure 3: Byte encodings of instructions. All values are shown in hexadecimal.
图 3: 指令的字节编码。所有值都以十六进制显示。

To solve Phase 5, you can use gadgets in the region of the code in rtarget demarcated by functions start_farm and end_farm. In addition to the gadgets used in Phase 4, this expanded farm includes the encodings of different movl instructions, as shown in Figure 3C. The byte sequences in this part of the farm also contain 2-byte instructions that serve as functional nops, i.e., they do not change any register or memory values. These include instructions, shown in Figure 3D, such as andb %al,%al, that operate on the low-order bytes of some of the registers but do not change their values.
为了解决第 5 阶段的问题，您可以使用目标区域中由 start _ farm 和 end _ farm 函数划分的代码区域中的 gadget。除了第 4 阶段中使用的小工具之外，这个扩展场还包括不同 movl 指令的编码，如图 3C 所示。场的这一部分的字节序列还包含作为函数 nops 的 2 字节指令，也就是说，它们不更改任何寄存器或内存值。其中包括如图 3D 所示的指令，例如 andb% al,% al，这些指令对某些寄存器的低序字节进行操作，但不更改它们的值。

Some Advice:
一些建议:

> You'll want to review the effect a movl instruction has on the upper 4 bytes of a register, as is described on page 183 of the text.
> 你需要检查一下 movl 指令对寄存器上面 4 个字节的影响，正如文本的第 183 页所描述的。

The official solution requires eight gadgets (not all of which are unique).
官方的解决方案需要八个小工具(并非所有都是独一无二的)。

Good luck and have fun!
祝你好运，玩得开心！

A Using HEX2RAW
使用 HEX2RAW

HEX2RAW takes as input a hex-formatted string. In this format, each byte value is represented by two hex digits. For example, the string "012345" could be entered in hex format as "30 31 32 33 34 35 00." (Recall that the ASCII code for decimal digit x is 0x3x, and that the end of a string is indicated by a null byte.)
HEX2RAW 接受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"可以以十六进制格式输入为"30313233343500"(回想一下，十进制数字 x 的 ASCII 代码是 0x3x，字符串的末尾是一个空字节。)

The hex characters you pass to HEX2RAW should be separated by whitespace (blanks or newlines). We recommend separating different parts of your exploit string with newlines while you're working on it. HEX2RAW supports C-style block comments, so you can mark off sections of your exploit string. For example:
传递给 HEX2RAW 的十六进制字符应该用空格(空格或换行符)分隔。我们建议您在处理利用字符串时使用换行分隔它的不同部分。HEX2RAW 支持 c 风格的块注释，因此您可以标记出利用字符串的部分。例如:

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

48 c7 c1 f0114000/* mov $0x40011f0,% rcx */

Be sure to leave space around both the starting and ending comment strings (“/*”, “*/”), so that the comments will be properly ignored.
一定要在开始和结束的注释字符串(“/* ”、“ */”)周围留出空间，这样注释就会被正确地忽略。

If you generate a hex-formatted exploit string in the file exploit.txt, you can apply the raw string to
如果在 exploit.txt 文件中生成十六进制格式的利用字符串，则可以将原始字符串应用到
CTARGET or RTARGET in several different ways:
CTARGET 或 RTARGET 有几种不同的方式:

1. You can set up a series of pipes to pass the string through

    HEX2RAW. unix> cat exploit.txt | ./hex2raw | ./ctarget

   您可以设置一系列管道，将字符串传递到 HEX2RAW。Unix >

    cat exploit.txt | ./hex2raw | .目标

2. You can store the raw string in a file and use I/O redirection:
   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
    Unix > ./hex2raw < exploit.txt > exploit-raw.txt unix > ./ctant < exploit-raw. txt

   This approach can also be used when running from within GDB:
   这种方法也可以在 GDB 内运行时使用:

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

3. You can store the raw string in a file and provide the file name as a command-line argument:
3. 可以将原始字符串存储在一个文件中，并以命令行参数的形式提供文件名:

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

This approach also can be used when running from within GDB.
当从 GDB 内部运行时，也可以使用这种方法。

B Generating Byte Codes
生成字节码

Using GCC as an assembler and OBJDUMP as a disassembler makes it convenient to generate the byte codes for instruction sequences. For example, suppose you write a file example.s containing the following assembly code:
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，假设您编写了一个包含以下汇编代码的文件 example.s:

```
# Example of hand-generated assembly code
                # 手工生成的组装示例 代码
pushq   $0xabcdef       Push value onto stack
普什图  $0xdef          # 将值推入堆栈
addq    $17,%rax        Add 17 to %rax
Addq    17 美元,% rax   # 添加 17% 到% rax
movl    %eax,%edx       Copy lower 32 bits to %edx
移动    X，x            # 复制较低的 32 位到 x
```

The code can contain a mixture of instructions and data. Anything to the right of a '#' character is a comment.
代码可以包含指令和数据的混合。“ # ”字符右边的任何字符都是注释。

You can now assemble and disassemble this file:
现在，您可以组装和反汇编这个文件:

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

The generated file example.d contains the following:

生成的文件 example.d 包含以下内容:

example.o: file format elf64-x86-64
文件格式 elf64-x86-64

Disassembly of section .text:
拆卸 部分，文字:
0000000000000000 <.text>:
000000000000000000 < . text > :

		ef		
0:	68	英 cd ab 00	pushq	$0xabcdef
0:	68	孚 Cd ab 00	普什图	$0xdef
5:		c0 11	add	$0x11,%rax
图 5:	48	83 C011	加	$0x11,% rax
9:		c2	mov	%eax,%edx
九:	89	C2	动	X，x

The lines at the bottom show the machine code generated from the assembly language instructions. Each line has a hexadecimal number on the left indicating the instruction's starting address (starting with 0), while
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(以 0 开始)，而

the hex digits after the ‘:’ character indicate the byte codes for the instruction. Thus, we can see that the instruction push $0xABCDEF has hex-formatted byte code 68 ef cd ab 00.
“：”字符后面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。

From this file, you can get the byte sequence for the code:
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
68 ef cd ab 004883 c01189 c2

This string can then be passed through HEX2RAW to generate an input string for the target programs.. Alter-natively, you can edit example.d to omit extraneous values and to contain C-style comments for readability, yielding:
然后可以通过 HEX2RAW 传递该字符串，以为目标程序生成输入字符串。.可以编辑 example.d 以省略无关的值，并包含 c 风格的注释以提高可读性，结果是:

```
    ef cd
    英孚    ab        /* pushq    $0xabcdef     */
68 cd       Ab   00   /* pushq    $0xdef        */
       c0             /* add      $0x11,%rax    */
48 83  C0  11         /* add      $0x11,% rax   */
                      /* mov                    */
    c2                           %eax,%edx
89 C2                 /* mov      X，x          */
```

This is also a valid input you can pass through HEX2RAW before sending to one of the target programs.
这也是一个有效的输入，您可以在发送到目标程序之前通过 HEX2RAW。

References
参考资料

[1] R. Roemer, E. Buchanan, H. Shacham, and S. Savage. Return-oriented programming: Systems, lan-guages, and applications. ACM Transactions on Information System Security, 15(1):2:1–2:34, March 2012.

罗默尔，e. 布坎南，h. 沙查姆和 s. 萨维奇。Return-to-libc 攻击: 系统、语言和应用。资讯系统安全交易》，15(1) : 2:1-2:34,2012 年 3 月。

[2] E. J. Schwartz, T. Avgerinos, and D. Brumley. Q: Exploit hardening made easy. In USENIX Security Symposium, 2011.

《利用硬化变得容易》，《 USENIX 安全研讨会，2011》。

14
图 14