15-213, Fall 20xx

15-213，秋季 20xx

The Attack Lab: Understanding Buffer Overflow Bugs Assigned: Tue, Sept. 29

攻击实验室:了解分配的缓冲区溢出错误:9 月 29 日星期二

Due: Thu, Oct. 8, 11:59PM EDT

到期时间:美国东部时间 10 月 8 日星期四晚上 11:59

Last Possible Time to Turn in: Sun, Oct. 11, 11:59PM EDT

最后可能的上交时间:美国东部时间 10 月 11 日，11:59

1 Introduction

1 导言

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:

这项任务涉及对两个具有不同安全漏洞的程序总共产生五次攻击。您将从本实验中获得的成果包括:

• You will learn different ways that attackers can exploit security vulnerabilities when programs do not safeguard themselves well enough against buffer overflows.

当程序不能很好地防止缓冲区溢出时，您将了解攻击者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.

通过这种方式，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功能，以降低程序的易受攻击性。

• You will gain a deeper understanding of the stack and parameter-passing mechanisms of x86-64 machine code.

您将对 x86-64 机器代码的堆栈和参数传递机制有更深入的了解。

• You will gain a deeper understanding of how x86-64 instructions are encoded.

您将更深入地了解 x86-64 指令的编码方式。

• You will gain more experience with debugging tools such as GDB and OBJDUMP.

您将获得更多调试工具的经验，如 GDB 和奥布迪普。

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:在本实验中，您将获得利用操作系统和网络服务器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，并理解这些安全弱点的本质，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取对任何系统资源的未授权访问。

You will want to study Sections 3.10.3 and 3.10.4 of the CS:APP3e book as reference material for this lab.

您将需要学习计算机科学:应用 3e 手册的第 3.10.3 节和第 3.10.4 节，作为本实验的参考资料。

1

1

2 Logistics

2 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。您将为为您定制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:

您可以通过将网络浏览器指向以下位置来获取文件:

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

http://$ Attacklab::SERVER _ NAME:15513/

INSTRUCTOR: $Attacklab::SERVER_NAME is the machine that runs the

教员:$Attacklab::SERVER_NAME 是运行

attacklab servers.You define it in attacklab/Attacklab.pm and in attacklab/src/build/driverhdrs.h

攻击实验室服务器。您可以在 attacklab/Attacklab.pm 和 attack lab/src/build/driver DRS . 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.

服务器将建立你的文件，并把它们放在一个叫做 targetk.tar 的 tar 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。

Note: It takes a few seconds to build and download your target, so please be patient.

注意:构建和下载目标需要几秒钟，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 文件保存在您计划在其中执行工作的(受保护的)Linux 目录中。然后给出命令:塔尔-xff 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.

你应该只下载一组文件。如果由于某种原因您下载了多个目标，请选择一个要处理的目标并删除其余目标。

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:如果您在电脑上扩展您的 targetk.tar，通过使用一个实用程序，如 Winzip，或让您的浏览器进行提取，您将面临重置可执行文件的权限位的风险。

The files in targetk include:

targetk 中的文件包括:

README.txt: A file describing the contents of the directory

描述目录内容的文件

ctarget: An executable program vulnerable to code-injection attacks

目标:易受代码注入攻击的可执行程序

rtarget: An executable program vulnerable to return-oriented-programming attacks

目标:易受面向返回编程攻击的可执行程序

cookie.txt: An 8-digit hex code that you will use as a unique identifier in your attacks.

cookie.txt:一种 8 位十六进制代码，在攻击中用作唯一标识符。

farm.c: The source code of your target's "gadget farm," which you will use in generating return-oriented programming attacks.

目标“小工具场”的源代码，您将使用它来生成面向返回的编程攻击。

hex2raw: A utility to generate attack strings.

hex2raw:生成攻击字符串的实用程序。

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中，并且您正在该本地目录中执行程序。

2

2

2.2 Important Points

2.2 要点

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.

以下是关于本实验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，这些观点没有多大意义。一旦开始，它们在这里作为规则的中心参考。

• You must do the assignment on a machine that is similar to the one that generated your targets.

您必须在与生成目标的机器相似的机器上完成任务。

• 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。具体而言，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:

The addresses for functions touch1, touch2, or touch3.

函数 touch1、touch2 或 touch3 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.

小工具场中您的一个小工具的地址。

• You may only construct gadgets from file rtarget with addresses ranging between those for functions start_farm and end_farm.

您只能从文件目标构造地址介于函数 start_farm 和 end_farm 之间的小工具。

3 Target Programs

3 个目标计划

Both CTARGET and RTARGET read strings from standard input.They do so with the function getbuf defined below:

CTARGET 和 RTARGET 都从标准输入中读取字符串。它们通过下面定义的函数 getbuf 来实现:

1 unsigned getbuf()

1 个无符号 getbuf()

2 {

2 {

4

4

6 }

6 }

char buf[BUFFER_SIZE];Gets(buf);return 1;

[缓冲器大小]；获取(buf)；返回 1；

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.

函数 gets 与标准库函数 Gets 相似，它从标准输入中读取一个字符串(以“\n”或文件结尾终止)，并将其存储在指定的目标位置(带有空终止符)。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 是特定于您的程序版本的编译时常数。

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 get()和 get()无法确定它们的目标缓冲区是否足够大，可以存储它们读取的字符串。它们只是复制字节序列，可能会超出目的地分配的存储范围。

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:

如果用户键入并由 getbuf 读取的字符串足够短，那么 getbuf 显然将返回 1，如以下执行示例所示:

unix> ./ctarget

unix >。/ctarget

3

3

Cookie: 0x1a7dd803

cookie:0x1a 7d 803

Type string: Keep it short!No exploit.Getbuf returned 0x1 Normal return

字符串类型:保持简短！没有剥削。Getbuf 返回 0x1 正常返回

Typically an error occurs if you type a long string:

如果键入长字符串，通常会出现错误:

unix> ./ctarget Cookie: 0x1a7dd803

unix >。/c target Cookie:0x1a 7d 803

Type string: This is not a very interesting string, but it has the property ...Ouch!: You caused a segmentation fault!Better luck next time

类型字符串:这不是一个非常有趣的字符串，但是它有属性...哎哟！:您造成了分段错误！希望下次运气好一些

(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，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，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:

CTARGET 和 RTARGET 都采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

-问:不要向评分服务器发送结果

-i FILE: Supply input from a file, rather than from standard input

-i FILE:从文件中提供输入，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 值不一致的字节值。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。

Important points:

要点:

• 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.

您的攻击字符串不得在任何中间位置包含字节值 0x0a，因为这是换行符的 ASCII 代码(“\n”)。当 Gets 遇到该字节时，它将假设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 要求两位十六进制值由一个或多个空格隔开。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(注意小端字节排序所需的反转)。

When you have correctly solved one of the levels, your target program will automatically send a notification to the grading server.For example:

正确解决其中一个级别后，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

unix >。/hex2raw < ctarget.l2.txt |。/c target Cookie:0x1a 7d 803

Type string:Touch2!: You called touch2(0x1a7dd803) Valid solution for level 2 with target ctarget PASSED: Sent exploit string to server to be validated.NICE JOB!

类型字符串:Touch2！:您调用了 touch 2(0x1a 7d 803)目标为“已通过”的 2 级有效解决方案:已向服务器发送攻击字符串进行验证。干得好！

4

4

Phase Program Level Method Function Points 1 CTARGET 1 CI touch1 10 2 CTARGET 2 CI touch2 25 3 CTARGET 3 CI touch3 25 4 RTARGET 2 ROP touch2 35 5 RTARGET 3 ROP touch3 5

阶段程序级别方法功能点 1 目标 1 配置项接触 1 10 2 目标 2 配置项接触 2 25 3 目标 3 配置项接触 3 25 4 目标 2 ROP 接触 2 35 5 目标 3 ROP 接触 35

CI: Code injection

CI:代码注入

ROP: Return-oriented programming

ROP:面向返回的编程

Figure 1: Summary of attack lab phases

图 1:攻击实验阶段总结

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.

服务器将测试您的攻击字符串，以确保它确实有效，并且它将更新 Attacklab 计分板页面，指示您的用户标识(由您的匿名目标号码列出)已完成此阶段。

You can view the scoreboard by pointing your Web browser at

您可以通过将网络浏览器指向以下位置来查看记分板

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

http://$ Attacklab::SERVER _ NAME:15513/记分板

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误不会受到惩罚。你可以随意在 CTARGET 和 RTARGET 使用任何你喜欢的字符串。

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:

重要注意事项:您可以在任何一台 Linux 机器上运行您的解决方案，但是为了提交您的解决方案，您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 总结了实验的五个阶段。可以看出，前三个涉及对 CTARGET 的代码注入攻击，后两个涉及对 RTARGET 的面向返回编程攻击。

4 Part I: Code Injection Attacks

4 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，您的利用字符串将攻击 CTARGET。这个程序的设置方式是堆栈位置在每次运行中保持一致，这样堆栈上的数据就可以被视为可执行代码。这些特性使得程序容易受到攻击，其中利用字符串包含可执行代码的字节编码。

4.1 Level 1

4.1 1 级

For Phase 1, you will not inject new code.Instead, your exploit string will redirect the program to execute an existing procedure.

对于阶段 1，您不会注入新代码。相反，您的利用字符串将重定向程序以执行现有过程。

Function getbuf is called within CTARGET by a function test having the following C code:

函数 getbuf 在 CTARGET 中由具有以下 C 代码的函数测试调用:

5

5

1 void test() 2 {

1 无效测试()2 {

int val;

int val

val = getbuf();

val = getbuf();

printf("No exploit.Getbuf returned 0x%x\n", val);

打印(“无漏洞”。Getbuf 返回 0x%x\n”，val)；

4

4

6 }

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 执行它的返回语句时(getbuf 的第 5 行)，程序通常在函数测试中恢复执行(在这个函数的第 5 行)。我们想改变这种行为。在文件目标中，有一个函数 touch1 的代码，它具有以下 C 表示:

1 void touch1()

1 空白触摸 1()

2 {

2 {

4

4

7 }

7 }

vlevel = 1;/Part of validation protocol / printf("Touch1!: You called touch1()\n");validate(1);exit(0);

vlevel = 1。/验证协议/打印的一部分(“Touch1！:您拨打了电话 touch 1()\ n”)；验证(1)；退出(0)；

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，而不是返回测试。请注意，您的利用字符串也可能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。

Some Advice:

一些建议:

• All the information you need to devise your exploit string for this level can be determined by exam-ining a disassembled version of CTARGET.Use objdump -d to get this dissembled version.

您需要为该级别设计利用字符串的所有信息都可以通过检查 CTARGET 的分解版本来确定。使用 objdump -d 获取此已传播版本。

• 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.

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。

• 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 来逐步完成 getbuf 的最后几个指令，以确保它做的是正确的事情。

• 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 堆栈框架中的位置取决于编译时常数 BUFFER_SIZE 的值，以及 GCC 使用的分配策略。您需要检查反汇编代码来确定它的位置。

4.2 Level 2

4.2 第 2 级

Phase 2 involves injecting a small amount of code as part of your exploit string.

阶段 2 涉及注入少量代码作为您的利用字符串的一部分。

Within the file ctarget there is code for a function touch2 having the following C representation:

在文件目标中，有一个函数 touch2 的代码，它具有以下 C 表示:

1 void touch2(unsigned val)

1 空白触点 2(无符号值)

6

6

2 {

2 {

4

4

89

89

10 11

10 11

12 }

12 }

vlevel = 2;/Part of validation protocol / if (val == cookie) {

v level = 2；/验证协议的一部分/ if (val == cookie) {

printf("Touch2!: You called touch2(0x%.8x)\n", val);validate(2);} else {

printf(“Touch 2！:您调用了 touch2(0x%.8x)\n "，val)；验证(2)；}其他{

printf("Misfire: You called touch2(0x%.8x)\n", val);fail(2);

printf(“失火:您调用了 touch2(0x%.8x)\n”，val)；失败(2)；

}

}

exit(0);

退出(0)；

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，而不是返回测试。然而，在这种情况下，您必须让它看起来像是已经传递了 cookie 作为它的参数。

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.

您将希望以这样一种方式定位注入代码地址的字节表示，即 getbuf 代码末尾的 ret 指令将控制权转移给它。

• 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.

您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.

不要试图在漏洞利用代码中使用 jmp 或调用指令。这些指令的目的地址编码很难表述。对所有控制权转移使用 ret 指令，即使您没有从呼叫中返回。

• See the discussion in Appendix B on how to use tools to generate the byte-level representations of instruction sequences.

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 Level 3

4.3 3 级

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，函数 hexmatch 和 touch3 的代码具有以下 C 表示:

1 /Compare string to hex represention of unsigned value / 2 int hexmatch(unsigned val, char sval)

1/将字符串与无符号值的十六进制表示进行比较/ 2 int hexmatch(无符号值，字符值)

3 {

3 {

4

4

7

7

9 }

9 }

char cbuf[110];

char cbuf[110];

/Make position of check string unpredictable / char *s = cbuf + random() % 100;sprintf(s, "%.8x", val);return strncmp(sval, s, 9) == 0;

/使检查字符串的位置不可预测/char * s = cbuf+random()% 100；sprintf(s，" %.8x "，val)；返回 strncmp(sval，s，9)= 0；

7

7

10

10

11 void touch3(char *sval)

11 空白触点 3(char *sval)

12 {

12 {

13 14

13 14

15 16 17

15 16 17

18 19 20 21

18 19 20 21

22 }

22 }

vlevel = 3;/Part of validation protocol / if (hexmatch(cookie, sval)) {

v level = 3；/验证协议的一部分/ if (hexmatch(cookie，sval)) {

printf("Touch3!: You called touch3(\"%s\")\n", sval);validate(3);} else {

printf(“Touch 3！:您调用了 touch3(\"%s\")\n "，sval)；验证(3)；}其他{

printf("Misfire: You called touch3(\"%s\")\n", sval);fail(3);

printf(“失火:您调用了 touch3(\"%s\")\n”，sval)；失败(3)；

}

}

exit(0);

退出(0)；

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，而不是返回测试。您必须让它看起来像是已经传递了 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(从最高到最低排列)，不带前导“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，一个字符串用 C 表示为一系列字节，后跟一个值为 0 的字节。在任何一台 Linux 机器上键入“man ascii”，查看所需字符的字节表示。

• Your injected code should set register %rdi to the address of this string.

您注入的代码应该将寄存器%rdi 设置为该字符串的地址。

• 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.

调用函数 hexmatch 和 strncmp 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，您需要小心放置 cookie 的字符串表示。

5 Part II: Return-Oriented Programming

第二部分:面向返回的编程

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 执行代码注入攻击比 CTARGET 要困难得多，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.

它采用随机分组，因此每次运行的堆栈位置不同。这使得确定您注入的代码将位于何处变得不可能。

• 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始，程序也会因分段错误而失败。

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，聪明的人已经设计了通过执行现有代码而不是注入新代码来完成程序中有用事情的策略。最一般的形式被称为面向返回编程(ROP) [1，2]。ROP 的策略是识别现有程序中的字节序列，该程序由一条或多条指令组成，后面跟着指令 ret。这种片段被称为

8

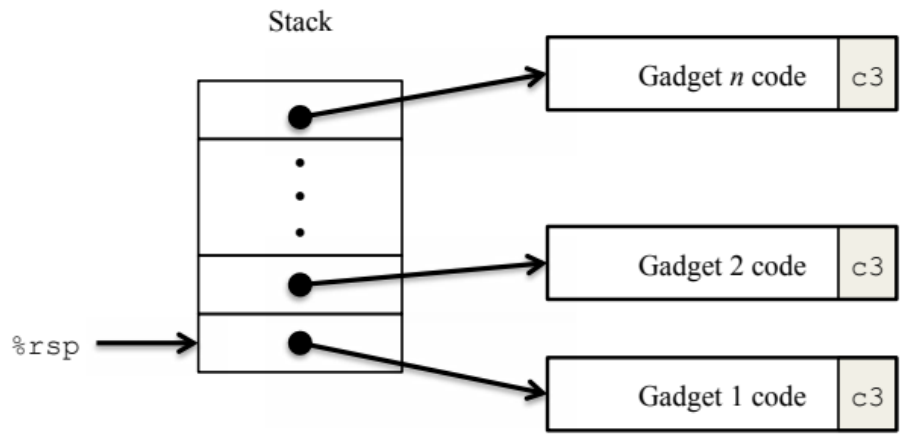

Figure 2: Setting up sequence of gadgets for execution.Byte value 0xc3 encodes the ret instruction.

图 2:设置小工具的执行顺序。字节值 0xc3 编码 ret 指令。

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.

小玩意。图 2 展示了如何设置堆栈来执行一系列 n 个小工具。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，最后一个是 0xc3，编码 ret 指令。当程序从这个配置开始执行 ret 指令时，它将启动一系列小工具执行，ret 指令位于每个小工具的末尾，导致程序跳到下一个小工具的开始。

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.

小工具可以利用与编译器生成的汇编语言语句相对应的代码，尤其是函数末尾的语句。实际上，这种形式可能有一些有用的小工具，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，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。

For example, one version of rtarget contains code generated for the following C function:

例如，rtarget 的一个版本包含为以下 C 函数生成的代码:

void setval_210(unsigned p)

void setval_210(无符号 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>:

0000000000400f15 <setval_210 >:

400f15: c7 07 d4 48 89 c7 movl $0xc78948d4,(%rdi)

400 f15:C7 07 D4 48 89 C7 movl $ 0xc 78948 D4，(%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 48 89 c7 编码指令 movq %rax，%rdi。(有用 movq 指令的编码见图 3A。)这个序列后面是字节值 c3，它编码 ret 指令。该函数从地址 0x400f15 开始，序列从函数的第四个字节开始。因此，该代码包含一个小工具，起始地址为 0x400f18，将寄存器%rax 中的 64 位值复制到寄存器%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.

您的 RTARGET 代码包含许多类似于上面所示的 setval_210 函数的函数，我们称之为小工具场。您的工作将是识别小工具场中有用的小工具，并使用这些小工具来执行类似于您在阶段 2 和阶段 3 中所做的攻击。

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:小工具场由目标副本中的开始场和结束场功能来划分。不要试图从程序代码的其他部分构造小工具。

5.1 Level 2

5.1 二级

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，但是要使用您的小工具场的小工具对 RTARGET 进行编程。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 : 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:该指令由单字节 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:该指令(发音为“no op”，是“no operation”的缩写)由单字节 0x90 编码。它唯一的作用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.

您需要的所有小工具都可以在由 start_farm 和 mid_farm 函数划分的目标代码区域中找到。

• You can do this attack with just two gadgets.

您只需两个小工具就可以完成此攻击。

• 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，您的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 3 级

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.

在你进入第五阶段之前，停下来考虑一下你迄今为止已经取得的成就。在阶段 2 和 3 中，您让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，你可以把你自己的代码注入到一个远程机器中。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。虽然您没有注入自己的代码，但是您能够注入一种通过缝合现有代码序列来操作的程序。你在实验室也得到了 95/100 分。这是个好成绩。如果你有其他紧迫的义务，考虑马上停止。

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 进行 ROP 攻击，用指向 cookie 字符串表示的指针调用函数 touch3。这看起来并不比使用 ROP 攻击调用 touch2 困难得多，只是我们已经做到了。此外，第 5 阶段只计 5 分，这并不是衡量其所需努力的真正标准。对于那些想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。

10

A.Encodings of movq instructions

A.movq 指令的编码

Source S	Destination 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 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 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
%rbp	48 89 e8	48 89 e9	48 89 ea	48 89 eb	48 89 ec	48 89 ed	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 fe	48 89 ff

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 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 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
RBP %	48 89 E8	48 89 e9	48 89 ea	48 89 EB	48 89 EC	48 89 ed	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 Fe	48 89 ff

B.Encodings of popq instructions

B.popq 指令的编码

Operation Register R

操作寄存器

%rax %rcx %rdx %rbx %rsp %rbp %rsi %rdi

% rax % rcx % RDX % rbx % RSP % RBP % RSI % rdi

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

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

C.Encodings of movl instructions

C.movl 指令的编码

Source S	Destination D							
	%eax	%ecx	%edx	%ebx	%esp	%ebp	%esi	%edi

%eax %ecx %edx %ebx %esp %ebp %esi %edi	89 c0 89 c8 89 d0 89 d8 89 e0 89 e8 89 f0 89 f8	89 c1 89 c9 89 d1 89 d9 89 e1 89 e9 89 f1 89 f9	89 c2 89 ca 89 d2 89 da 89 e2 89 ea 89 f2 89 fa	89 c3 89 cb 89 d3 89 db 89 e3 89 eb 89 f3 89 fb	89 c4 89 cc 89 d4 89 dc 89 e4 89 ec 89 f4 89 fc	89 c5 89 cd 89 d5 89 dd 89 e5 89 ed 89 f5 89 fd	89 c6 89 ce 89 d6 89 de 89 e6 89 ee 89 f6 89 fe	89 c7 89 cf 89 d7 89 df 89 e7 89 ef 89 f7 89 ff

来源 S	目的地 D							
	%eax	%ecx	%edx	%ebx	%esp	%ebp	%esi	%edi
% eax % ecx % EDX % ebx % esp % ebp % ESI % EDI	89 c0 89 c8 89 d0 89 D8 89 E0 89 E8 89 F0 89 F8	89 C1 89 C9 89 D1 89 d9 89 E1 89 e9 89 f1 89 F9	89 C2 89 ca 89 D2 89 da 89 E2 89 ea 89 F2 89 fa	89 C3 89 CB 89 D3 89 db 89 E3 89 EB 89 F3 89 FB	89 C4 89 cc 89 D4 89 DC 89 E4 89 EC 89 F4 89 fc	89 C5 89 CD 89 D5 89 DD 89 E5 89 ed 89 F5 89 FD	89 C6 89 ce 89 D6 89 de 89 E6 89 ee 89 F6 89 Fe	89 C7 89 cf 89 D7 89 df 89 E7 89 ef 89 F7 89 ff

D.Encodings of 2-byte functional nop instructions

D.2 字节功能 nop 指令的编码

Operation	Register R			
	%al	%cl	%dl	%bl
andb R, R orb R, R cmpb R, R testb R, R	20 c0 08 c0 38 c0 84 c0	20 c9 08 c9 38 c9 84 c9	20 d2 08 d2 38 d2 84 d2	20 db 08 db 38 db 84 db

操作	寄存器 R			
	%al	%cl	%dl	%bl
和 R，R orb R，R cmpb R，R 测试 b R，R	c0 08 c0 38 c0 84 c0	20 c9 08 c9 38 c9 84 c9	20 d2 08 d2 38 d2 84 d2	20 db 08 db 38 db 84 db

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。

11

11

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.

要解决阶段 5，您可以在由函数 start_farm 和 end_farm 划分的目标中的代码区域使用小工具。除了第 4 阶段中使用的小工具，这个扩展的场还包括不同 movl 指令的编码，如图 3C 所示。场的这一部分中的字节序列也包含 2 字节指令，用作功能性 nop，即它们不改变任何寄存器或存储器值。这些指令包括如图 3D 所示的指令，例如和% a1、% a1，它们对一些寄存器的低位字节进行操作，但不改变它们的值。

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.

如本文第 183 页所述，您需要回顾一条 movl 指令对寄存器高位 4 字节的影响。

• The official solution requires eight gadgets (not all of which are unique).

官方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

使用

HEX2RAW

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 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 码是 0x3x，字符串的结尾用空字节表示。)

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)隔开。我们建议您在开发过程中用换行符分隔开发字符串的不同部分。HEX2RAW 支持 C 风格的块注释，因此您可以标记出您的利用字符串的部分。例如:

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

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

* /

* /

Be sure to leave space around both the starting and ending comment strings ("/", "/"), so that the comments will be properly ignored.

请务必在开始和结束注释字符串("/"，/")周围留出空间，以便正确忽略注释。

If you generate a hex-formatted exploit string in the file exploit.txt, you can apply the raw string to CTARGET or RTARGET in several different ways:

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

1.You can set up a series of pipes to pass the string through HEX2RAW.

1.您可以设置一系列管道，使管柱通过 HEX2RAW。

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

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

2.You can store the raw string in a file and use I/O redirection:

2.您可以将原始字符串存储在文件中，并使用输入/输出重定向:

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

unix >。/hex 2 raw < exploit . txt > exploit-raw . txt UNIX >。/ctarget < exploit-raw.txt

This approach can also be used when running from within GDB:

当从 GDB 内部运行时，也可以使用这种方法:

12

12

unix> gdb ctarget

unix> gdb ctarget

(gdb) run < exploit-raw.txt

(gdb)运行< exploit-raw.txt

3.You can store the raw string in a file and provide the file name as a command-line argument:

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

unix >。/hex 2 raw < exploit . txt > exploit-raw . txt UNIX >。/ctarget -i exploit-raw.txt

This approach also can be used when running from within GDB.

在 GDB 境内跑步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 作为汇编器，OBJDUMP 作为反汇编器，可以方便地为指令序列生成字节码。例如，假设您编写了一个包含以下汇编代码的文件示例:

Example of hand-generated assembly code pushq $0xabcdef # Push value onto stack addq $17,%rax # Add 17 to %rax movl %eax,%edx # Copy lower 32 bits to %edx

#手动生成的汇编代码示例将值推送到堆栈地址推$17，%rax #将 17 添加到%rax movl %eax，%edx #将低 32 位复制到%edx

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。 “#”字符右侧的任何内容都是注释。

You can now assemble and disassemble this file:

您现在可以组装和拆卸该文件:

unix> gcc -c example.s

unix> gcc -c 示例

unix> objdump -d example.o > example.d

unix> objdump -d 示例. o >示例. d

The generated file example.d contains the following:

生成的文件示例. d 包含以下内容:

example.o: file format elf64-x86-64

示例. o:文件格式 elf64-x86-64

Disassembly of section .text:

拆卸部件。文本:

0000000000000000 <.text>:

000000000000000000 <。文本>:

0: 68 ef cd ab 00 pushq $0xabcdef

0: 68 ef cd ab 00 pushq $0xabcdef

5: 48 83 c0 11 add $0x11,%rax

5: 48 83 c0 11 增加$0x11，%rax

9: 89 c2 mov %eax,%edx

9: 89 c2 mov %eax，%edx

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

13

13

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.From this file, you can get the byte sequence for the code:

“:”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代码 68 ef cd ab 00。从这个文件中，您可以获得代码的字节序列:

68 ef cd ab 00 48 83 c0 11 89 c2

68 ef cd ab 00 48 83 c0 11 89 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，这个字符串可以通过 HEX2RAW 来生成目标程序的输入字符串..从本质上来说，您可以编辑示例.d 以忽略无关的值，并包含 C 风格的注释以提高可读性，从而产生:

68 ef cd ab 00 /

68 ef cd ab 00 /

48 83 c0 11 /

c0 11 /

89 c2 /

89 c2 /

pushq $0xabcdef * add $0x11,%rax * mov %eax,%edx

pushq $0xabcdef *添加$0x11，%rax * mov %eax，%edx

* / * / * /

* / * / * /

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，您可以通过 HEX2RAW，然后发送到其中一个目标程序。

References

参考

[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.

[1]罗默、布坎南、沙哈姆和萨维奇。面向返回的编程:系统、语言和应用。《信息系统安全交易》，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.

[2] E. J .施瓦茨，T. Avgerinos 和 d .布鲁姆利。问:利用硬化变得容易。2011 年 USENIX 安全研讨会。

14