内容由 福昕翻译 生成（Content is generated by foxit）

时间（time）: 2024-11-15

15-213, Fall 20xx

20xx 年秋季 15 日至 213 日

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

攻击实验室：了解缓冲区溢出漏洞分配：9 月 29 日星期二

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

到期日：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

最后可能的交车时间：美国东部时间 10 月 11 日星期日晚上 11:59

Introduction

介绍

This assignment involves generating a total of five attacks on two programs having different security vulnerabilities. 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发起总共五次攻击。您将从本实验室获得的成果包括：

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

您将了解当程序不能很好地保护自己免受缓冲区溢出时，攻击者可以利用安全漏洞的不同方式。

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，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.

您将更深入地了解 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.

您将获得更多调试工具的经验，如 GDB 和 OBJDUMP。

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，以便在编写系统代码时避免它们。我们不容忍使用任何其他形式的攻击来未经授权访问任何系统资源。

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

您需要学习 CS:APP3e 书的第 3.10.3 节和第 3.10.4 节，作为本实验室的参考材料。

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。您将为为您自定义的目标程序生成攻击。

Getting Files

获取文件

You can obtain your files by pointing your Web browser at:

您可以通过将 Web 浏览器指向以下位置来获取文件：

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

http://$Attacklab:：服鬠名称：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 Attacklab 服务器的机器。您可以在 attacklab/attacklab.pm 和 attacklab/src/build/driverhdrs 中定义它。h

The server will build your files and return them to your browser in a tar file called targetk.tar, where

服务器将构建您的文件，并在名为 targetk.tar 的 tar 文件中将其返回给浏览器，其中

k is the unique number of your target programs.

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：构建和下载目标需要几秒钟的时间，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工作的（受保护的）Linux 目录中。然后给出命令：tar-xvf targetk.tar。这将提取一个包含以下文件的目录 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。如果出于某种原因您下载了多个目标，请选择一个目标进行处理并删除其余目标。

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 PC 上通过使用 Winzip 等实用程序或让浏览器进行提取来扩展 targetk.tar，则可能会重置可执行文件上的权限位。

The files in targetk include:
targetk 中的文件包括：

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：易受代码注入攻击的可执行程序

rtarget: An executable program vulnerable to return-oriented-programming attacks
rtarget：易受面向返回的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：一个 8 位十六进制代码，您将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.

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

hex2raw: A utility to generate attack strings.

hex2raw：一个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.

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

Important Points

重要事项

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。一旦你开始，它们在这里就作为规则的核心参考。

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类似的机器上执行该任务。

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，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

您注入代码的地址

The address of one of your gadgets from the gadget farm.

小工具场中某个小工具的地址。

You may only construct gadgets from file rtarget with addresses ranging between those for func- tions start_farm and end_farm.

您只能从地址在函数 start_farm 和 end_farm 之间的文件 rtarget 构造小工具。

Target Programs

目标项目

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

CTARGET 和 RTARGET 都从标准输入读取字符串。它们通过函数 getbuf 来实现

defined below:

定义如下：

```
unsigned getbuf()
unsigned getbuf()

{
{

char buf[BUFFER_SIZE];
char buf[缓冲区大小]；
```

Gets(buf);

获取（buf）；

return 1;

返回 1；

6 }

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.

函数 Gets 类似于标准库函数 Gets——它从标准输入（以"\n"或文件末尾结尾）读取字符串，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_SIZE 字节。在生成目标时，BUFFER_SIZE 是特定于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，可能会超出目标分配的存储边界。

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:

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

unix> ./ctarget

unix>/ctarget

Cookie: 0x1a7dd803

Cookie：0x1a7dd803

Type string: Keep it short!

键字符串：保持短！

No exploit. Getbuf returned 0x1 Normal return

没有漏洞。Getbuf 返回 0x1 正常返回

Typically an error occurs if you type a long string:

通常，如果键入长字符串，会出现错误：

unix> ./ctarget

unix>/ctarget

Cookie: 0x1a7dd803

Cookie：0x1a7dd803

Type string: This is not a very interesting string, but it has the property ...

类型字符串：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！：你造成了分段错误！

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，显示的 cookie 值将与您的不同。）RTARGET 程序将具有相同的行为。正如错误消息所示，超过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 输入的字符串，这样它们就能做更有趣的事情。这些被称为漏洞利用字符串。

Both CTARGET and RTARGET take several different command line arguments:

CTARGET 和 RTARGET 都接受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：从文件而不是标准输入提供输入

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.

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

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.

您的漏洞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 Gets 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 需要由一个或多个空格分隔的两位数十六进制值。因此，如果你想创建一个十六进制值为 0 的字节，你需要将其写为 00。要创建单词 0xdeadbeef，您应该将"ef be ad de"传递给 HEX2RAW（注意小端字节序需要反转）。

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

当您正确解决其中一个级别时，您的目标程序将自动向分级服务器发送通知。例如：

unix> ./hex2raw < ctarget.l2.txt | ./ctarget

unix>/hex2raw<ctarget.l2.txt |/ctarget

Cookie: 0x1a7dd803

Cookie：0x1a7dd803

Type string:Touch2!: You called touch2(0x1a7dd803) Valid solution for level 2 with target ctarget

类型字符串：Touch2！：您调用了 touch2（0x1a7dd803）目标 ctarget 的 2 级有效解决方案

PASSED: Sent exploit string to server to be validated. NICE JOB!

PASSED：已将漏洞利用字符串发送到服务器进行验证。干得好！

Phase

阶段

Program

程序

Level

级别

Method

方法

Function

功能

Points

积分

1

1

2

2

3

3

CTARGET

CTARGET

CTARGET CTARGET

CTARGET

1

1

2

2

3

3

CI

CI

CI CI

CI CI

touch1

触摸 1

touch2 touch3

触摸 2 触摸 3

10

10

25

25

25

25

4

4

5

5

RTARGET

RTARGET

RTARGET

RTARGET

2

2

3

3

ROP

ROP

ROP

ROP

touch2

触摸 2

touch3

触摸 3

35

35

5

5

CI:Code injection

CI：代碼入

ROP:Return-oriented programming

ROP：面向返回的程

Figure 1: Summary of attack lab phases

图 1：攻击实验室阶段总结

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.

服务器将测试您的漏洞利用字符串以确保其确实有效，并将更新 Attacklab 记分板页面，指示您的用户 ID（按匿名目标号码列出）已完成此阶段。

You can view the scoreboard by pointing your Web browser at

您可以通过将 Web 浏览器指向以下位置来查看记分牌

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

与炸弹实验室不同，在这个实验室犯错不会受到惩罚。欢迎随时向 CTARGET 开火

and RTARGET with any strings you like.

RTARGET 可以用你喜欢的任何字符串。

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:

重要提示：您可以在任何 Linux 机器上运行解决方案，但要提交解决方案，您需要在以下机器之一上运行：

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，前三个阶段涉及对 CTARGET 的代码注入（CI）攻击，而后两个阶段涉及针对 RTARGET 的面向返回程（ ROP）攻击。

Part I: Code Injection Attacks

第一部分：代碼入攻击

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，您的漏洞利用字符串将攻击 CTARGET。该程序的设置方式是，堆栈位置在每次运行中都是一致的，因此堆栈上的数据可以被视为可执行代码。这些特性使程序容易受到攻击，其中漏洞利用字符串包含可执行代码的字节编码。

Level 1

级别 1

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，您将不会注入新代码。相反，您的漏洞利用字符串将重定向程序以执行现有过程。

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：

```
void test()
无效测试（）

{
{

int val;
int val；

val = getbuf();
val=getbuf（）；

printf("No exploit. Getbuf returned 0x%x\n", val);
printf（“没有漏洞。Getbuf 返回 0x%x\n”，val）；

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 行将）。我们想改变这种行为。在 ctarget 文件中，有一个函数 touch1 的代码，具有以下 C 表示形式：

```
void touch1()
无效触摸 1（）

{
{

vlevel = 1;/* Part of validation protocol */
vlevel=1；/*验证方案的一部分*/

printf("Touch1!: You called touch1()\n");
printf（“Touch1！：您调用了 Touch1（）”）；

validate(1);
验证（1）；

exit(0);
退出（0）；

7 }
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.

你的任务是让 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，您的漏洞利用字符串也可能损坏与此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 exam- ining a disassembled version of CTARGET. Use objdump -d to get this dissembled version.

通过检查 CTARGET 的反汇编版本，可以确定为此级别设计漏洞利用字符串所需的所有信息。使用 objdump-d 获取此已分解版本。

The idea is to position a byte representation of the starting address for touch1 so that the ret

其想法是定位 touch1 的起始地址的字节表示，以便 ret

instruction at the end of the code for getbuf will transfer control to touch1.

getbuf 代码末尾的指令将控制权转移到 touch1。

Be careful about byte ordering.

注意字节顺序。

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，以确保它正在做正确的事情。

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 的堆栈帧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以确定其位置。

Level 2

2 级

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

第 2 阶段涉及注入少量代码作为漏洞利用字符串的一部分。

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

在 ctarget 文件中，有一个函数 touch2 的代码，其 C 表示如下：

```
void touch2(unsigned val)
void touch2（未签名的 val）

{
{

vlevel = 2;/* Part of validation protocol */
vlevel=2；/*验证方案的一部分*/

4
4

if (val == cookie) {
if（val==cookie）{

5
5
```

printf("Touch2!:

printf（“触摸 2！：

You called touch2(0x%.8x)\n", val);

您调用了 touch2（0x%.8x）\n“，val）；

6

6

validate(2);

验证（2）；

7

7

} else {

}其他{

8

8

printf("Misfire:

Printf（“错误：

You called touch2(0x%.8x)\n", val);

您调用了 touch2（0x%.8x）\n“，val）；

9

9

fail(2);

失败（2）；

10

10

}

}

11

11

exit(0);

退出（0）；

12 }

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，而不是返回测试。但是，在这种情况下，您必须使其看起来像是触摸 2，就像您已经将 cookie 作为参数传递一样。

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

您将希望以这样的方式定位注入代码地址的字节表示形式：

ret instruction at the end of the code for getbuf will transfer control to it.

getbuf 代末尾的 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.

您注入的代码应将注册表设置为 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。这些指令的目标地址编码很难公式化。对所有控制权转移使用 ret 说明，即使您没有结束通话。

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

参附　　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。

Level 3

级别 3

Phase 3 also involves a code injection attack, but passing a string as argument.

第 3 阶段还涉及代码注入攻击，但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 ctarget 文件中，函数 hexmatch 和 touch3 的代码具有以下 C 表示形式：

/* Compare string to hex represention of unsigned value */

/*将字符串与无符号值的十六进制表示进行比较*/

int hexmatch(unsigned val, char *sval)

int 十六进制匹配（无符号 val，char*sval）

{

{

char cbuf[110];

Charchubfist；

/* Make position of check string unpredictable */

/*使检查字符串的位置不可预测*/

char *s = cbuf + random() % 100;

char*s=cbuf+随机数（）%100；

sprintf(s, "%.8x", val);

sprintf（s，“%.8x”，val）；

return strncmp(sval, s, 9) == 0;

返回 strncmp（sval，s，9）==0；

9 }

9 }

10

10

void touch3(char *sval)

void touch3（字符*sval）

{

{

vlevel = 3;/* Part of validation protocol */

vlevel=3；/*验证方案的一部分*/

14

14

if (hexmatch(cookie,

如果（hexmatch（cookie，

sval)) {

肌肉）{

15

15

printf("Touch3!:

printf（“触摸 3！：

You called touch3(\"%s\")\n", sval);

您调用了 touch3（“%s\”）\n“，sval）；

16

16

validate(3);

验证（3）；

17

17

} else {

}其他{

18

18

printf("Misfire:

Printf（“错误：

You called touch3(\"%s\")\n", sval);

您调用了 touch3（“%s\”）\n“，sval）；

19

19

fail(3);

失败（3）；

20

20

}

}

21

21

exit(0);

退出（0）；

22 }

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，而不是返回测试。您必须使其看起来像是触摸了 3，就好像您传递了 cookie 的字符串表示作为其参数一样。

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.”

您需要在漏洞利用字符串中包含 cookie 的字符串表示形式。该字符串应由八个十六进制数字组成（从最高有效位到最低有效位排序），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，字符串在 C 中表示为字节序列，后面是值为 0 的字节。在任何 Linux 机器上入“man ascii”以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.

当调用函数 hexmatch 和 strncmp 时，它们会将数据推送到堆栈上，覆盖容纳 getbuf 使用的缓冲区的内存部分。因此，您需要小心放置 cookie 的字符串表示的位置。

Part II: Return-Oriented Programming

第二部分：面向返回的程

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RTARGET 程序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此类攻击：

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，使堆栈位置在每次运行中都不同。这使得无法确定注入代码的位置。

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头，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，聪明的人已经设计出通过执行现有代码而不是注入新代码来在程序中完成有用事情的策略。最常见的形式被称为面向返回的编程（ROP）[1,2]。ROP 的策略是识别现有程序中的字节序列，该序列由一个或多个指令组成，后面是指令 ret。这样的段称为

Stack

堆栈

c3

c3

Gadget 1 code

小工具 1 代码

c3

c3

Gadget 2 code

小工具 2 代码

c3

c3

Gadget n code

小工具 n 代码

%rsp

%rsp

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

图 2：设置执行的小工具序列。字节值 0xc3 对ret 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以执行 n 个小工具的序列。在这个图中，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，用于编码 ret 指令。当程序执行以此配置开始的 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 命令会使程序跳到下一个小工具的开头。

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，特别是函数末尾的代码。在实践中，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。

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:

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

0000000000400f15 <setval_210>:

00000000 400f15<设定值 210>：

400f15:

400f15：

c7

c7

07

07

d4 48 89 c7

d4 48 89 c7

movl

movl

$0xc78948d4,(%rdi)

0xc78948d4 美元，（%rdi）

400f1b:

400f1b：

c3

c3

retq

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 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。）此序列后面是字节值 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，此代码包含一个起始地址为 0x400f18 的小工具，它将把寄存器%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.

RTARGET 的代码包含许多与上面所示的 setval_210 函数类似的函数，我们称之为小工具场。你的工作将是在小工具场中识别有用的小工具，并使用这些小工具执行类似于你在第 2 和第 3 阶段所做的攻击。

Important: The gadget farm is demarcated by functions start_farm and end_farm in your copy of

重要提示：小工具场由您的副本中的函数 start_farm 和 end_farm 划分

rtarget. Do not attempt to construct gadgets from other portions of the program code.

rtarget。不要试图从程序代码的其他部分构造小工具。

Level 2

2 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).

对第 4 阶段，您将重复第 2 阶段的攻击，但要使用小工具场中的小工具对 RTARGET 程序进行攻击。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：此指令由单字节 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：此指令（音 “ no op”，是“no operation”的缩写）由单个字节 0x90 编码。它的唯一作用是使程序计数器增 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函数 start_farm 和 mid_farm 划分的 rtarget 代码区域中找到。

You can do this attack with just two gadgets.

你只需要两个小工具就可以进行这种攻击。

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，您的漏洞利用字符串将包含小工具地址和数据的组合。

Level 3

级别 3

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.

在进入第五阶段之前，停下来考虑一下你迄今为止所取得的成就。在阶段 2 和 3 中，您使程序执行您自己设计的机器代码。如果 CTARGET 是一台网络服务器，您可以将自己的代码注入到远程机器中。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，但您可以注入

一种通过将现有代码序列拼接在一起进行操作的程序。你的实验室也得了 95/100 分。这是一个很好的分数。如果你有其他紧迫的义务，考虑现在停止。

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.

第 5 阶段要求您对 RTARGET 猼 ROP 攻击，以调用函数 touch3，并将指针指向 cookie 的字符串表示。这似乎并不比使用 ROP 攻击来调用 touch2 困难得多，除非我们已经做到了这一点。此外，第 5 阶段只计算 5 分，这并不是衡量它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。

Encodings of movq instructions

movq 指令豞

movq S, D

movq S，D

Source

来源

Destination D

目的地 D

S

S

%rax

%rax

%rcx

%rcx

%rdx

%rdx

%rbx

%rbx

%rsp

%rsp

%rbp

%rbp

%rsi

%si

%rdi

%rdi

%rax

%rax

48

48

89

89

c0

c0

48

48

89

89

c1

c1

48

48

89

89

c2

c2

48

48

89

89

c3

c3

48

48

89

89

c4

补体第四成份

48

48

89

89

c5

c5

48

48

89

89

c6

c6

48

48

89

89

c7

c7

%rcx

%rcx

48

48

89

89

c8

c8

48

48

89

89

c9

c9

48

48

89

89

ca

ca

48

48

89

89

cb

cb

48

48

89

89

cc

复写的副本

48

48

89

89

cd

cd

48

48

89

89

ce

总工程师

48

48

89

89

cf

查阅

%rdx

%rdx

48

48

89

89

d0

d0

48

48

89

89

d1

d1

48

48

89

89

d2

d2

48

48

89

89

d3

d3

48

48

89

89

d4

d4

48

48

89

89

d5

d5

48

48

89

89

d6

d6

48

48

89

89

d7

d7

%rbx

%rbx

48

48

89

89

d8

d8

48

48

89

89

d9

d9

48

48

89

89

da

da

48

48

89

89

db

db

48

48

89

89

dc

dc

48

48

89

89

dd

dd

48

48

89

89

de

的

48

48

89

89

df

df

%rsp

%rsp

48

48

89

89

e0

e0

48

48

89

89

e1

e1

48

48

89

89

e2

e2

48

48

89

89

e3

e3

48

48

89

89

e4

e4

48

48

89

89

e5

e5

48

48

89

89

e6

e6

48

48

89

89

e7

e7

%rbp

%rbp

48

48

89

89

e8

e8

48

48

89

89

e9

e9

48

48

89

89

ea

ea

48

48

89

89

eb

eb

48

48

89

89

ec

ec

48

48

89

89

ed

预计起飞时间

48

48

89

89

ee

ee

48

48

89

89

ef

ef

%rsi

%si

48

48

89

89

f0

f0

48

48

89

89

f1

f1

48

48

89

89

f2

f2

48

48

89

89

f3

f3

48

48

89

89

f4

f4

48

48

89

89

f5

f5

48

48

89

89

f6

f6

48

48

89

89

f7

f7

%rdi

%rdi

48

48

89

89

f8

f8

48

48

89

89

f9

f9

48

48

89

89

fa

fa

48

48

89

89

fb

fb

48

48

89

89

fc

fc

48

48

89

89

fd

fd

48

48

89

89

fe

fe

48

48

89

89

ff

ff

Encodings of popq instructions

popq 指令編

Operation

操作

Register R

注册 R

%rax

%rax

%rcx

%rcx

%rdx

%rdx

%rbx

%rbx

%rsp

%rsp

%rbp

%rbp

%rsi

%si

%rdi

%rdi

popq R

popq R

58

59

59

5a

5a

5b

5b

5c

5c

5d

5d

5e

5e

5f

5f

Encodings of movl instructions

movl 指令編

movl S, D

移动 S、D

Source

来源

S

S

Destination D

目的地 D

%eax

%eax

%ecx

%ecx

%edx

%edx

%ebx

%ebx

%esp

%esp

%ebp

%ebp

%esi

%esi

%edi

%edi

%eax

%eax

%ecx

%ecx

%edx

%edx

%ebx

%ebx

%esp

%esp

%ebp

%ebp

%esi

%esi

%edi

%edi

89 c0

89 摄氏度

89 c8

89 立方厘米

89 d0

89 天

89 d8

89 天 8

89 e0

89 e0

89 e8

89 e8

89 f0

89 英尺

89 f8

89 英尺 8 英寸

89 c1

89 c1

89 c9

89 立方厘米

89 d1

89 d1

89 d9

89 天 9 分

89 e1

89 e1

89 e9

89 e9

89 f1

89 f1

89 f9

89 f9

89 c2

89 c2

89 ca

89 卡

89 d2

89 d2

89 da

89 天

89 e2

89 e2

89 ea

89 个

89 f2

89 f2

89 fa

89 法

89 c3

89 c3

89 cb

89 炭黑

89 d3

89 d3

89 db

89 分贝

89 e3

89 e3

89 eb

89 eb

89 f3

89 f3

89 fb

89 fb

89 c4

89 c4

89 cc

89 毫升

89 d4

89 d4

89 dc

89 直流

89 e4

89 e4

89 ec

89 秒

89 f4

89 f4

89 fc

89 立方英尺

89 c5

89 c5

89 cd

89 cd

89 d5

89 天 5 分

89 dd

89 日

89 e5

89 e5

89 ed

89 ed

89 f5

89 f5

89 fd

89 fd

89 c6

89 c6

89 ce

89 ce

89 d6

89 天 6

89 de

89 德

89 e6

89 e6

89 ee

89 人

89 f6

89 f6

89 fe

89 英尺

89 c7

89 立方英尺

89 cf

89 立方英尺

89 d7

89 天 7

89 df

89 df

89 e7

89 e7

89 ef

89 ef

89 f7

89 英尺 7 英寸

89 ff

89 起

Encodings of 2-byte functional nop instructions

2 字节函数式 nop 指令的编

Operation

操作

Register R

注册 R

%al

%al

%cl

%cl

%dl

%dl

%bl

%bl

andb

和 b

R, R

R、R

20

20

c0

c0

20

20

c9

c9

20

20

d2

d2

20

20

db

db

orb

orb

R, R

R、R

08

08

c0

c0

08

08

c9

c9

08

08

d2

d2

08

08

db

db

cmpb

cmpb

R, R

R、R

38

38

c0

c0

38

38

c9

c9

38

38

d2

d2

38

38

db

db

testb

testb

R, R

R、R

84

84

c0

c0

84

84

c9

c9

84

84

d2

d2

84

84

db

db

Figure 3: Byte encodings of instructions. All values are shown in hexadecimal.

图 3：指令的字节编码。所有值均以十六进制显示。

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.

要解决阶段 5，您可以在 rtarget 中由函数 start_farm 和 end_farm 划分的代码区域中使用小工具。除了阶段 4 中使用的小工具外，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还包含用作函数 nop 的 2 字节指令，即它们不会更改任何寄存器或内存值。其中包括如图 3D 所示的指令，如 andb%al、%al，它们对某些寄存器的低位字节进行操作，但不改变其值。

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.

您需要查看 movl 指令对寄存器上部 4 个字节的影响，如本文第 183 页所述。

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（并非所有小工具都是唯一的）。

Good luck and have fun!

祝你好运，玩得开心！

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 接收一个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 是 0x3x，字符串的末尾由空字节表示。）

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）分隔。我们建议您在处理漏洞字符串时用换行符分隔漏洞字符串的不同部分。HEX2RAW 支持 C 风格的块注释，因此您可以标记漏洞字符串的部分。例如：

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

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

请确保在开始和结束注释字符串（“/*”、“*/”）周围留出空格，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

如果在 exploit.txt 文件中生成十六进制格式的漏洞利用字符串，则可以将原始字符串应用于

CTARGET or RTARGET in several different ways:

CTARGET 或 RTARGET 有几种不同的方式：

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

您可以设置一系列管道，使字符串通过 HEX2RAW。

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

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

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>/hex2raw<exploit.tx>exploit-law.txt

unix> ./ctarget < exploit-raw.txt

unix>/ctarget<exploit-raw.txt

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

从 GDB 内部运行时也可以使用这种方法：

unix> gdb ctarget

unix>gdb-ctarget

(gdb) run < exploit-raw.txt

（gdb）运行<exploit-raw.txt

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

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>/hex2raw<exploit.tx>exploit-law.txt

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

unix>/ctarget-i 开发版-raw.txt

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

从 GDB 内部运行时也可以使用这种方法。

Generating Byte Codes

生成字碼

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

#手工生成的汇编代码示例

pushq$0xabcdef# Push value onto stack addq$17,%rax# Add 17 to %rax

pushq$0xabcdef#将值推送到堆栈 addq$17，%rax#将 17 添加到%rax

movl%eax,%edx# Copy lower 32 bits to %edx

movl%eax，%edx#将低位 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评论。

You can now assemble and disassemble this file:

现在，您可以组装和拆卸此文件：

unix> gcc -c example.s

unix>gcc 示例 。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

example.o：文件格式 elf64-x86-64

Disassembly of section .text:

文本部分的分解：

0000000000000000 <.text>:

0000000000000000<.text>：

0: 68 ef cd ab 00pushq $0xabcdef

0：68 ef cd ab 00pushq$0xabcdef

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

5:48 83 c0 11 加 0x11 美元，%rax

9: 89 c2mov%eax,%edx

9:89 c2mov%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

底部的行显示了由汇编语言指令生成的机器代码。每行左侧都有一个十六进制数字，表示指令的起始地址（从 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:

从该文件中，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，可以通过 HEX2RAW 传递此字符串，为目标程序生成输入字符串。。另外，您可以编辑 example.d 以省略多余的值，并包含 C 风格的注释以提高可读性，从而产生：

68 ef cd ab 00/* pushq $0xabcdef */

68 ef cd ab 00/*pushq$0xabcdef*/

48 83 c0 11/* add$0x11,%rax */

48 83 c0 11/*增加 0x11 美元，%rax*/

89 c2/* mov%eax,%edx */

89 c2/*移动%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，您可以在发送到目标程序之前通过 HEX2RAW。

References

工具书类

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

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。

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

E.J.Schwartz、T.Avgerinos 和 D.Brumley。Q： 利用硬化变得容易。在 2011 年 USENIX 安全研讨会上。