x86_64逆向工程简介——其他练习

lzhtony / 2018-07-27 17:54:28 / 浏览数 2682 技术文章 技术文章 顶(0) 踩(0)

本篇为译作，原文：Additional Exercises in Reverse Engineering

本系列上一篇：x86_64逆向工程简介

这是我（相当长时间以前的）x86_64逆向工程简介（即本系列上一篇文章）的续集。这篇文章快速过了一遍那些没有在该教程中提到的实用的逆向工程方法，并提供了许多

CrackMe程序

您可以在GitHub上找到文中讨论的CrackMe程序。克隆这个存储库，并且在不查看源代码的情况下，使用`make crackme01`,`make crackme02`,.....构建所有CrackMe。

工具和软件

这些CrackMe仅适用于Unix系统，我使用Linux编写本教程。您需要安装开发环境的基本知识——C编译器（`gcc`）、对象检查工具（`objdump`，`objcopy`，`xxd`）等等。本

```
sudo apt install build-essential gcc xxd binutils
```

您可以在这里安装Radare2。

对于其他系统，通过对应系统的包管理器安装相应的包即可。

练习解答

crackme05.c

这个CrackMe非常类似于前一个教程中提供的那些，但更加模块化。它把成功和失败的情况封装成函数（用`aaa`分析然后`afl`列出函数），这两个函数打印出相应的字符串然

`main`函数中有一些对失败函数`sym.fail`的调用，每个次调用都源于不同的条件。只有通过所有的检查，执行流程才会达到0x880，其中RDI（函数的第一个参数）加载了输

其中一些检查是很明显的的。例如，在0x7d7的代码处检查字符串长度必须正好是16：

```
call sym.imp.strnlen
cmp eax, 0x10
jne 0x850
```

但其他检查调用了函数`check_with_mod`，每次带有三个参数。例如在0x81a处：

```
lea rdi, [rbx + 8]
mov edx, 5
mov esi, 4
call sym.check_with_mod
```

这里RBX是`argv[1]`，所以这实际上调用`check_with_mod(argv[1] + 8, 5, 4)`。第三个参数，这里是4，在所有调用中看起来是一样的，但第二个参数是会变化的。那么`check_with_mod`是在做什么的呢？

像之前一样，`s sym.check_with_mod`后跟`pdf`会给我们答案。它实际上是一个非常简单的函数，只有20个指令。它的核心是一个循环，它将输入字符串（参数1）中某些字节的值相加，

接下来函数执行整数除法`idiv r8d`。这个指令把RDX除以R8（第二个参数），商保存在RAX中，余数保存在EDX中。然后代码检查RDX为是否零，并把商丢弃,这是模运算。

所以这段代码的迷雾被揭开，它是检查给定地址处的4个字节的和，是否可被给定值整除。回到`main`函数，我们可以看到有四个地方调用这个函数，需要分别满足那对应四个

这还不够。程序还要求第2个字节为'B'和第0xd（13）字节为'Q'。

所以目前字符串为：`..B.........Q..`

现在我们需要计算那些空白填什么。通过做一些数学计算我们可以找到`EEBD,,,,2222QQOO`，正确！

crackme06.c

使用Radare2静态分析（即仅查看静态的代码）可以非常容易解决这个问题，但我想演示一下逆向工程师工具箱中的另一个工具——`strace`。

strace打印出程序在运行时的每个系统调用。这对于了解程序的基本行为非常有用，同时可以让您轻松地分离出特定类型的行为，比如网络连接和文件I/O。

这里我们运行strace ./crackme06.64 test：

```
execve("./crackme06.64", ["./crackme06.64", "test"], [/* 56 vars */]) = 0
brk(NULL)                               = 0x5645ad7a0000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=249222, ...}) = 0
mmap(NULL, 249222, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f8dded0d000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\340\22\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1960656, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8dded0b000
mmap(NULL, 4061792, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f8dde743000
mprotect(0x7f8dde919000, 2097152, PROT_NONE) = 0
mmap(0x7f8ddeb19000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1d6000) = 0x7f8ddeb19000
mmap(0x7f8ddeb1f000, 14944, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f8ddeb1f000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7f8dded0c4c0) = 0
mprotect(0x7f8ddeb19000, 16384, PROT_READ) = 0
mprotect(0x5645ab906000, 4096, PROT_READ) = 0
mprotect(0x7f8dded4a000, 4096, PROT_READ) = 0
munmap(0x7f8dded0d000, 249222)          = 0
brk(NULL)                               = 0x5645ad7a0000
brk(0x5645ad7c1000)                     = 0x5645ad7c1000
openat(AT_FDCWD, "test", O_RDONLY)      = -1 ENOENT (No such file or directory)
dup(2)                                  = 3
fcntl(3, F_GETFL)                       = 0x8402 (flags O_RDWR|O_APPEND|O_LARGEFILE)
fstat(3, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 1), ...}) = 0
write(3, "PANIC! Aborting due to: No such "..., 50PANIC! Aborting due to: No such file or directory
) = 50
close(3)                                = 0
exit_group(-1)                          = ?
+++ exited with 255 +++
```

大部分输出对我们无用——加载未设置的共享库预加载，并映射内存，这些全部由shell完成。但后还是调用了openat：

```
openat(AT_FDCWD, "test", O_RDONLY)      = -1 ENOENT (No such file or directory)
...
write(3, "PANIC! Aborting due to: No such "..., 50PANIC! Aborting due to: No such file or directory
) = 50
```

程序用文件名test调用openat，尝试以只读模式打开它，得到一个错误，然后输出这个错误。

显然，它想要一个存在的文件。创建test文件并在里面放入一些内容，我们看到：

```
openat(AT_FDCWD, "test", O_RDONLY)  = 3
fstat(3, {st_mode=S_IFREG|0664, st_size=13, ...}) = 0
read(3, "some content\n", 4096)         = 13
read(3, "", 4096)                       = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 1), ...}) = 0
write(1, "Access denied.\n", 15Access denied.
)        = 15
```

这里代码成功打开文件并返回文件描述符（3）。它第一次读取文件返回“some content\n”，然后再次读取它得不到任何内容。最后打印“Access denied.”

所以我们知道程序正在从文件中读取数据。进入Radare，很容易发现那些被读入的字节后来与字符串“scrambled egg
42”进行比较。如果您加载含有这个字符串的文件，就能通过这个CrackMe。

花时间静态地跟踪程序会得到一样的结果，但会花更多的时间。动态分析有时候很有效，特别是用于找出目标可执行文件关键部分的时候。

crackme07.c

这个CrackMe非常简单，但它采用了一个稍有些奇怪的输入机制。它将第一个参数与一个静态字符串进行比较，但只有当前时间在某个范围内时才会判定成功。

有一个函数sym.cur_hour，它调用了libc函数localtime。它取回了函数生成的localtime结构体，并将这个结构体偏移8个字节处的一个四字（译者注：此处应该是双字

根据localtime文档，这个结构体长这样：

```
struct tm {
   int tm_sec;         /* seconds */
   int tm_min;         /* minutes */
   int tm_hour;        /* hours */
   int tm_mday;        /* day of the month */
   int tm_mon;         /* month */
   int tm_year;        /* year */
   int tm_wday;        /* day of the week */
   int tm_yday;        /* day in the year */
   int tm_isdst;       /* daylight saving time */
};
```

所以8个字节偏移处是`tm_hour`，当前的小时。在`main`函数中，这个值被这样使用：

```
call sym.cur_hour
mov ebx, eax
...
sub ebx, 5
cmp ebx, 1
jbe 0x985
```

换句话说，小时数必须在5到6之间（即时间必须在0500到0659之间）。所以要么熬夜，要么暂时修改你的系统时间，才能通过这个CrackMe。

crackme08.c

这是另一个"计算"的CrackMe。也就是说，它在程序运行时即时计算正确的密码。

程序的一个重要特点是它在0x869处进行了内存分配：

```
mov edi, 0xf
call sym.imp.malloc
```

这是在C程序中非常常见的调用——这是程序在堆上获取内存的方法——之前的CrackMe并不需要这样做。这个调用分配了0xf（15）个字节。

紧接着有一个通常情况看不到的指令：`cpuid`。这个指令是CPU向软件标识自身的一种方式，它按顺序在EBX，EDX和ECX中放置"特征字节"。虽然它们可以是任何字节，但
SOC"、搞笑的"CyrixInstead"。

总之，这些字节后来被移入其他寄存器，并与刚才调用malloc返回的指针一起传递给函数`sym.shift_int_to_char`。看看这个函数，它很简单：它将给定双字的第一，第

回到main，我们看到其他一些字节被设置为：'3'，'Q'和空字节。可以肯定的是，密码由CPUID指令的三个双字加上"3Q"构成，并且可以看出：三个双字是12个字符，加上

果然，程序马上就使用`strcmp`对这个缓冲区进行检查，然后通过`free`函数把空间释放，还给系统。

这是许多CrackMe中第一个在不同的计算机上有不同解的。对我的计算机来说，答案是"GenuineIntel3Q"。

附录

感谢您阅读第二篇教程！我希望它带来帮助和指引。我下一步是要创造一些更接近实际的例子，例如逆向一些关键的C应用程序和精简版JavaScript，以及用GDB进行更深入

如果您喜欢这些CrackMe，请支持我的Patreon！对于那些已经在赞助我的人，非常感谢您支持我的工作。

目录