1. Linux环境和相关工具

在本章，我们将集中介绍Linux环境，因为这将贯穿我们整本书的始终。本书重点是对Linux的二进制分析，那么利用好Linux自带的一些通用的本地环境工具将会对Linux二进制分析非常有帮助。Linux自带了应用普遍的binutils工具，也可以在网站http://www.gnu.org/software/binutils/ 找到，里面包含了一些用于二进制分析和破解的工具。这不是另一本介绍IDA Pro用法的书。IDA是一款唾手可得的进行二进制反编译的通用软件，在需要使用的时候我会鼓励大家使用它。不过在本书中我们不会使用IDA。相反，通过本书的学习，面对任何的Linux系统你都能够有思路利用现有的环境进行二进制破解。籍由此，便可以欣赏到作为一个真正的黑客可以利用许多免费工具的Linux环境之美。在本书中，我们将会展示各种工具的使用，随着每个章节的推进，也会不断回顾这些工具的使用方法。那么我们将本章节作为参考章节，来介绍Linux环境下的相关工具和技巧。如果你已经对Linux环境以及反编译、调试、转换ELF文件的工具非常熟悉，可以跳过本章节。

**Linux工具**

在本书中，我们将会使用许多免费公用的工具。这部分内容将会对其中的某些工具进行概要阐述。

**GDB**

GNU调试器（GDB）不止可以用来调试有bug的应用程序。也可以用来研究甚至改变一个程序的控制流，可以用来修改代码、寄存器还有数据结构。这对于一个致力于寻找软件漏洞或者是破解一个内部非常复杂的病毒的黑客来讲，都是非常常见的工作。GDB主要用于分析ELF二进制文件和Linux进程，是Linux黑客的必用工具，在本书中我们也会在各种不同的例子中使用到GDB。

**GNU binutils中的Objdump**

Object dump（objdump）是一种对代码进行快速反编译的简洁方案，在反编译简单、未受干扰的二进制文件时非常有用，但是要进行任何真正有挑战性的反编译任务，特别是针对恶意软件时，objdump就很快显示出了它的局限性。其最主要的一个缺陷就是需要依赖ELF不同节（section）的头部，不会做任何控制流的分析，这都极大降低了objdump的鲁棒性。如果要反编译的文件没有节头，那么使用objdump的后果就是不能够正确的反编译二进制文件中的代码，甚至都不能打开二进制文件。不过，对于一些比较传统的任务，比如反编译普通的未经强化、拆解或者以任何方式混合的二进制文件，objdump已经足够了。objdump可以读取所有的通用的ELF类型。下面是使用objdump的一些常用的例子：

* 查看ELF文件中每个段的所有 data/code

objdump –D <elf\_object>

* 只查看ELF文件中的程序代码

objdump –d <elf\_object>

* 查看所有符号：

objdump –tT <elf\_object>

在第二章，介绍ELF二进制规范时，我们将更深入的介绍objdump和其他相关工具。

**GNU binutils 中的objcopy**

Object copy（Objcopy）是一款不能够一言概之的非常强大的小工具。我推荐读者读一下描述详细的使用手册。Objcopy可以用来分析和修改各种ELF对象，虽然objcopy的某些特征是针对特定的ELF对象的。Objcopy通常用来修改ELF二进制文件或者将ELF段从一个文件拷贝到另一个文件中。

比如，将.data段从一个ELF对象中拷贝到一个文件中，可以使用下面的命令：

objcopy –only-section=.data <infile> <outfile>

Objcopy工具会在本书的后续内容中用到。现在只要记住这个工具的存在，并且知道这是对Linux二进制黑客来说非常有用的一个工具就可以了。

**strace**

System call trace（strace）[系统调用追踪]是基于ptrace(2) 系统调用的一款工具，strace通过在一个循环中使用PTRACE\_SYSCALL请求来显示运行中的程序所进行的系统调用活动以及程序执行中捕捉到的信号量。Strace在调试过程中非常有用，也可以用来收集运行时的系统调用相关信息。

使用strace命令来跟踪一个基本的程序：

strace /bin/ls –o ls.out

使用strace命令来对一个已有进程进行追加：

strace –p <pid> -o daemon.out

初始的输出将会显示每个系统调用的文件描述序列号，系统调用会将文件描述作为参数，如下所示：

SYS\_read(3, buf, sizeof(buf));

如果想查看读入到描述文件3中的所有数据，可以运行下面的命令：

strace –e read=3 /bin/ls

也可以使用 –e write=fd命令查看写数据。Strace是一个非常有用的小工具，将会有很多地方需要使用它。

**ltrace**

Library trace(ltrace)[库追踪]是另一个简洁的小工具，跟strace类似。ltrace跟strace的工作原理相似，ltrace会去解析一个程序的共享库链接信息，并打印出用到的库函数。

**基本的ltrace命令**

除了可以查看库函数调用之外，还可以使用-S标志查看系统调用。ltrace命令通过解析可执行文件的动态段，并打印出共享静态库的实际符号和函数，来提供更细粒度的信息：

ltrace <program> -o program.out

**ftrace**

Function trace(ftrace)[函数追踪]是我自己设计的一个工具。功能跟ltrace类似，同时也可以显示出二进制文件本身的函数调用。我没有找到现成的实现这个功能的Linux工具，于是就决定自己编码写了一个。这个工具可以在网站<https://github.com/elfmaster/ftrace> 找到。在下一个章节中会对这个工具的使用进行介绍。

**readelf**

readelf命令是解析ELF二进制文件的一个非常有用的工具。在进行反编译之前，该命令能够提供收集对象相关信息所需要的针对ELF文件的所有数据。在本书中我们将会使用readelf命令收集符号、段、节、重定向入口、数据动态链接等相关的信息。readelf命令是分析ELF二进制文件的瑞士军刀。在第二章ELF二进制规范中我们会对该命令进行更深入的介绍，下面是几个常用的参数：

* 查询节头表：

readelf –S <object>

* 查询程序头表：

readelf –l <object>

* 查询符号表：

readelf -s <object>

* 查询ELF文件头数据：

readelf –e <object>

* 查询重定向入口：

readelf –r <object>

* 查询动态段：

readelf –d <object>

**ERESI——ELF反编译系统接口**

ERESI工程（<http://www.eresi-project.org> ）中包含着许多Linux二进制黑客梦寐以求的工具。坏消息是，其中的有些工具没有持续更新，有的与64位的Linux不适配。ERESI工程支持许多的体系结构，无疑是迄今为止最有创新性的破解ELF二进制文件的工具集合。由于我个人不太熟悉ERESI工程中工具的用法，并且其中有些不再更新了，在本书中就不再对该工程进行更深入的探讨了。不过，有两篇Phrack的文章能够说明ERESI工具的创新和强大的特性：

• Cerberus ELF interface (http://www.phrack.org/archives/issues/61/8. txt)

• Embedded ELF debugging (http://www.phrack.org/archives/ issues/63/9.txt)

**有用的设备和文件**

Linux有许多的文件、设备，还有/proc 条目，对狂热的黑客还有反编译工程师来说非常有用。在本书中，我们将会展示其中的许多文件。下面介绍下我们本书中常用到的一些文件。

**/proc/<pid>/maps**

/proc/<pid>/map文件包含了一个进程镜像的布局，通过内存映射来进行展现。包括可执行文件、共享库、栈、堆、VDSO等。这个文件对于快速解析一个进程的地址空间分布是至关重要的，在本书中会多次用到该文件。

**/proc/kcore**

/proc/kcore是proc文件系统的一项，是Linux内核的动态核心文件。也就是说，它是以ELF核心文件的形式所展现出来的原生内存转储，GDB可以使用/proc/kcore来对内核进行调试和分析。我们在第九章，Linux /proc/kcore分析，里面会更深入的介绍 /proc/kcore。

**/boot/System.map**

这个文件在几乎所有的Linux发行版上都有，对内核黑客来说是非常有用的一个文件，包含了整个内核的所有符号。

**/proc/kallsyms**

kallsyms跟System.map类似，区别就是kallsyms是内核所属的/proc的一个条目并且可以动态更新。如果安装了新的LKMs（Linux Kernel Modules），符号会自动添加到/proc/kallsyms中。/proc/kallsyms包含了内核中绝大部分的符号，如果在CONFIG\_KALLSYMS\_ALL内核配置中指明，则可以包含内核中全部的符号。

**/proc/iomem**

iomem是一个非常有用的proc条目，跟/proc/<pid>/maps类似，不过它是跟系统内存相关的。比如，想知道内核的text段所映射的物理内存，可以搜索“Kernel”字符串，就可以查看code/text段，data段和bss段的相关内容：

$ grep Kernel /proc/iomem

01000000-016d9b27 : Kernel code

016d9b28-01ceeebf : Kernel data

01df0000-01f26fff : Kernel bss

**/ECFS**

Extended core file snapshot(ECFS)[扩展核心文件快照]是一项特殊的核心转储技术，专门为高级取证分析进程镜像所设计的。这个软件的代码可以在https://github.com/elfmaster.ecfs看到。本书的第八章，ECFS——扩展核心文件快照技术，将会单独介绍ECFS及其使用方法。如果你已经进入到了高级内存取证分析阶段，你会非常想关注这一部分内容。

**链接器相关环境指针**

动态加载器/链接器还有链接的概念，在程序链接、执行的过程中都是避不开的基本组成部分。在本书中，你还会学到更多相关的概念。在Linux中，有许多方法可以代替动态链接器的行为可供二进制黑客使用。随着本书的深入，你会开始理解链接、重定向和动态加载的过程。下面是几个很有用处的链接器相关的属性，在本书中将会用到。

**LD\_PRELOAD环境变量**

LD\_PRELOAD环境变量可以设置成一个指定的库的路径，动态链接时可以比其他库有更高的优先级。这就允许预加载库中的函数和符号能够覆盖掉后续链接的库中的函数和符号。这在本质上允许你通过重定向共享库函数来进行运行时修复。在后续的章节中，这项技术可以用来绕过反调试代码，也可以用作外围rootkit。

**LD\_SHOW\_AUXV环境变量**

该环境变量能够通知程序加载器来展示程序运行时的辅助向量。辅助向量是放在程序栈（通过内核的ELF常规加载方式）上的信息，附带了传递给动态链接器的程序相关的特定信息。在第三章，Linux进程跟踪，我们将会对此进行进一步验证，不过这些信息对与反编译和调试来说非常有用。比如，要想获取进程镜像VDSO页的内存地址（也可以使用maps文件获取，之前介绍过），就需要查询AT\_SYSINFO。

下面是一个带有LD\_SHOW\_AUXV的辅助向量的例子：

$ LD\_SHOW\_AUXV=1 whoami

AT\_SYSINFO: 0xb7779414

AT\_SYSINFO\_EHDR: 0xb7779000

AT\_HWCAP: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov

pat pse36 clflush mmx fxsr sse sse2

AT\_PAGESZ: 4096

AT\_CLKTCK: 100

AT\_PHDR: 0x8048034

AT\_PHENT: 32

AT\_PHNUM: 9

AT\_BASE: 0xb777a000

AT\_FLAGS: 0x0

AT\_ENTRY: 0x8048eb8

AT\_UID: 1000

AT\_EUID: 1000

AT\_GID: 1000

AT\_EGID: 1000

AT\_SECURE: 0

AT\_RANDOM: 0xbfb4ca2b

AT\_EXECFN: /usr/bin/whoami

AT\_PLATFORM: i686

elfmaster

在第二章，ELF二进制规范中将会对辅助向量更进一步介绍。

**链接脚本**

链接脚本是我们的一个兴趣点，因为链接脚本是链接器翻译的，把程序划分成相应的节，内存，符号。默认的链接脚本可以使用“ld -verbose”查看。

ld链接程序有其自己翻译的一套语言，当有文件输入时，ld链接程序会用自己的语言来决定输出文件(比如可执行程序)的组织方式。比如，如果输出的是一个ELF可执行文件，链接脚本能够决定该输出文件的布局，每个段里面包含哪些节。另举一个例子：.bss节总是放在data段的末尾，这就是链接脚本决定的。你可能很好奇，这为什么就成了我们的一个兴趣点呢？一方面，对编译时链接过程有一定深入的了解是很重要的。gcc依赖于链接器和其他程序来完成编译的任务，在某些情况下，能够控制可执行文件的布局是很重要的。ld命令语言是一门相当深入的语言，不过超出了本书的范围，也是非常值得探究的。另一方面，在对可执行文件进行反编译时，普通段地址或者文件的其他部分有时候会被修改，这就表明引入了一个自定义的链接脚本。gcc通过使用 –T 标志来指定链接脚本。在第五章，Linux二进制保护中我们会介绍一个使用链接脚本的例子。

**总结**

我们仅接触了Linux环境和工具相关的一些基本概念，在后续的每个章节中都会经常用来进行展示。二进制分析主要是了解一些可用的工具和资源并进行相关的整合。目前我们只简要覆盖到了这部分工具，在接下来的章节里，随着对Linux二进制破解的这个大世界进行更进一步的探索，我们会有机会对每一个工具进行更进一步的介绍。在下一章，我们将会对ELF二进制规范进行更深入的探索，也会涉及到一些其他有趣的概念，比如动态链接，重定向，符号，节（section）等。