第 34 卷 第 3 期 计 算 机 工 程 2008 年 2 月
Vol.34 No.3 Computer Engineering February 2008
·软件技术与数据库· 文章编号：1000—3428(2008)03—0113—03 文献标识码：A 中图分类号：TP393.03

Linux 寄生程序加载动态库的研究与实现

夏 宏，刘立宇
(华北电力大学计算机系，北京 102206)

摘 要：寄生程序是指注入到可执行文件中的程序代码，被广泛地应用在二进制文件加解密、版权保护等领域。病毒也是寄生程序的一种。Linux 下的寄生程序很难利用宿主没有加载的动态连接库，使其功能受到很大限制。该文通过对 ELF 动态连接机制的研究，采用了一种寄生程序通过 proc 文件系统进行加载和利用动态库的方法，并对这种方法进行了实现。

关键词：寄生程序；可执行可连接文件格式；动态连接

Research and Implementation of Linux Parasite Access to Dynamic Library

XIA Hong, LIU Li-yu
(Department of Computer, North China Electric Power University, Beijing 102206)

【Abstract】 A parasite is defined as code that is injected into a host executable. There are many potential uses for parasite code: binary decryption, unpacking and copyright protection, to name a few. And the most common form of parasite is the virus. Development of feature rich Linux parasites is severely limited by the inability to reliably access functions external to the host file. This paper explores the dynamic linking mechanisms of the Executable and Linkable Format(ELF), and brings up a methodology that allows parasite code access to shared objects. The implementation of this methodology is presented.

【Key words】 parasite; Executable and Linkable Format(ELF); dynamic linking

Linux 动态连接的实现需要动态连接器和 ELF 文件的紧密合作。ELF 文件提供了一系列的数据结构，动态连接器利用这些数据结构进行动态加载库和解析符号的工作。寄生程序现有的加载和利用动态库的方法通常采用 2 种方法：一种是在开发寄生程序的环境中解析好一个符号的地址并把它保存在程序中。这就要求该符号在运行环境中和开发环境中的地址一致，只要运行环境稍微有所不同就会导致寄生程序失败。同时，这种方法也不能利用宿主程序没有加载的动态库。另一种是在寄生程序中保存动态库函数的一份拷贝。但这样做使得寄生程序的代码长度变大，而寄生程序的代码长度是非常重要的一个方面。一个大的寄生程序很容易被发现，从而失去它的作用。

1 ELF 文件格式及 ELF 动态连接机制

1.1 ELF 文件格式

ELF 文件格式为二进制文件提供了 2 种接口：可执行接口(图 1)和可连接接口。程序在执行时不需要可连接接口，因此本文不讨论。ELF 的程序头详细描述了 ELF 的可执行接口。

程序头保存在程序头表中，包含了产生进程所必需的信息。每个程序头描述一个段，包括该段在文件中的偏移和段的大小。程序头中的类型域决定该段如何被处理，如可载入的、可执行的以及可修改的等。段中包含了程序的代码、数据，以及其他运行时需要的信息[1]。

ELF 运行时的环境参数保存在动态描述表(图 1 中的段 *n*)的段中。

下面是动态描述表的结构体定义。

```
typedef struct
{
Elf32_Sword d_tag;
union
{
Elf32_Word d_val;
Elf32_Addr d_ptr;
}d_un;
} Elf32_Dyn;
```

图 1 ELF 可执行接口

动态描述表中的 d_tag 域决定 d_un 域如何解释。d_un 域可以被解释为指针或整数。对动态连接最重要的几种动态描述表都被解释为指针，包括指向动态符号表的指针和指向动态字符串表的指针。

可执行的 ELF 文件最终要在系统中运行，产生进程。下面简单介绍一下由 ELF 文件到进程的产生以及运行时 ELF 的布局。

程序执行时首先由载入器把 ELF 文件中可载入的段(包括动态库中的可载入段)通过系统调用 mmap()映射到内存地址中，接着载入器把控制转交给 ELF 的入口程序。

作者简介：夏 宏(1965－)，男，副教授，主研方向：计算机体系结构，网络安全；刘立宇，硕士研究生

收稿日期：2007-03-23 **E-mail：**Liuliyuqd@163.com

可执行的ELF文件被载入到静态连接时就设定好的固定地址中。这样，静态连接器就可以对局部代码和数据进行重定位。而 ELF 动态库因为共享的需要，必须能够被映射到进程的不同地址空间中。动态库中包含动态连接器需要的重定位表，以便动态连接器能对它做运行时的动态重定位。动态库中的代码都是位置无关代码。位置无关代码要利用其他的一些结构体来对外部数据和函数进行引用。

程序头中除了要给出一个可载入的段在文件中的大小，还要给出它在内存中的大小。段在内存中的大小要求向上舍入到内存页大小的整数倍。大部分可载入段的大小不是页的整数倍，它们在内存中会被填入文件中的其他内容。特别的，内存中的第一个代码段被填入的是 ELF 头和程序头。寄生程序在加载动态库时需对内存中的 ELF 头和程序头进行分析。

1.2 ELF 文件的动态连接技术

动态连接器负责动态连接工作，包括加载动态库和解析符号。

ELF文件通过符号表引用另一个ELF文件中的数据对象和函数对象。符号表中包含符号名和符号值。符号名是符号在动态字符串表中的索引。符号值是符号在ELF文件中的偏移，在运行时该值需要加上该运行模块被载入的基地址。可执行文件的载入地址在静态连接时就能确定，所以可执行文件内部的符号引用在静态连接时就能进行重定位处理[2]。

ELF 文件的数据段中有一个名为全局偏移表(GOT)的数组，表中每一项都是本运行模块要引用的一个全局变量或函数的地址。可以用 GOT 表来间接引用全局变量、函数，也可以把 GOT 表的基地址作为一个基准，用相对于该基准的偏移量来引用静态变量、静态函数。

过程连接表(PLT)是一个可以把程序控制传递到外部函数的结构体。i386 体系上的过程连接表的结构如下：

```
PLT0: push GOT[1] ;
      jmp GOT[2];
      nop
      ...
PLTn: jmp GOT[x + n]
      push n
      jmp PLT0
PLTn+1: jmp   GOT[x +n +1]
        push   n+1
        jmp   PLT0
```

当可执行模块调用一个外部函数时，控制首先被传递到静态连接器为该函数设置好的过程连接表项中。过程连接表中的第一条指令是一个跳转指令，如果这是本运行模块第 1 次调用该函数，此处的跳转相当于一个空指令，继续往下执行，重定位表中的偏移入栈，接着跳转到过程连接表中第 1 项。该过程连接表项会把程序流程引入到动态连接器中去。

动态连接器利用重定位项、符号表以及字符串表计算出函数的实际地址，并填入全局偏移表中。第一次调用以后，GOT 表项已指向函数的入口。以后再有对该函数的调用，跳到 PLT 表后，不再进入动态连接器，而是直接进入函数入口执行。

除了符号表、全局偏移表、过程连接表以及字符串表，ELF 文件还包含一个哈希表(hash table)。哈希表帮助迅速判定符号表中的哪个符号是要找的符号。

通常 ELF 文件的动态连接是以透明方式进行的。然而，也可以显式地访问动态连接函数来加载动态库。动态连接函数包含在动态连接器本身的代码段中。为了访问这些函数，静态连接时需要指明函数库 libdl。该库包含了一些包装函数，从而能够在静态连接时解析函数的引用。但包装函数只是简单地返回 0。真正的函数驻留在动态连接器中。

有了哈希表、动态字符串表以及动态符号表这些信息。就可利用下面的算法计算任何符号的地址：

```
hn = elf_hash(sym_name) % nbuckets;
for (ndx = hash[ hn ]; ndx; ndx = chain[ ndx ]) {
symbol = sym_tab + ndx;
if ( strcmp(sym_name, str_tab + symbol->st_name) == 0)
return (load_addr + symbol->st_value);
}
```

对符号名进行哈希运算，结果再对哈希表中元素个数取模，求得该符号在哈希表中的索引。然后把对应该索引的符号和要找的符号进行比较，找出匹配的符号。最后把符号的地址值加上模块的基地址就得到符号的地址[3]。

除上面的 ELF 知识外，还需要检查进程的方法才能实现寄生程序加载动态库的功能。Linux 提供了 2 种检查进程的方法。系统调用 ptrace()是一种功能有限的检查进程映象的方法，更为高级的方法利用了 proc 文件系统。

可以通过 proc 文件系统检查任意一个进程的状态。系统中的每个进程在/proc 目录下都有自己的目录。每个目录的名字就是该进程的 pid。此外，还有一个名字为 self 的目录，这是当前活动进程目录的一个符号连接。进程可以通过/proc/self 目录检查自己的状态。

proc 文件系统中包括描述进程状态的文件和描述进程映象的文件。寄生程序加载和利用动态库的方法主要用到描述进程映象的文件。

2 寄生程序加载和利用动态库的原理和实现

2.1 原理

动态连接器中包含了动态库加载函数。寄生程序可以通过 proc 文件系统，找到这些函数地址。然后显式调用函数，加载和利用自己需要的动态库。实现步骤如下：

(1)在进程中定位提供动态连接功能的运行模块；

(2)在该模块内定位加载和卸载动态库的函数；在该模块内定位符号解析的函数(可选)；

(3)调用动态库函数，加载、卸载动态库、解析符号。

步骤(1)最难实现。寄生程序必须确定进程映象的哪个模块提供了动态连接功能。通过 C 语言的文件操作函数，对 proc 文件系统进行读取并进行分析，利用 ELF 文件所提供的动态连接的机制，就可以找到要找的模块。

步骤(2)确定 dlopen()函数和 dlclose()函数的绝对地址。有了哈希表、符号表和字符串表的帮助，在一个 ELF 文件中确定一个符号的地址非常简单。上文已经给出计算任意符号地址的算法。

负责加载动态库的模块通常也负责符号的解析。如果这个模块中不包含符号解析的函数，有 2 个选择，一是在别的模块中找到 dlsym()函数；二是在寄生程序中实现符号解析的功能。

步骤(3)对先前获得的数据进行管理利用。有很多方法管理这些数据。因为堆中的数据在寄生程序的生存周期内一直可用，而且堆中的数据可以动态管理。本文选择把数据保存在堆中的一个链表中。

有了 dlopen()和 dlclose()函数的地址，寄生程序就可以任

意地加载和利用所需的动态库。这些函数地址和堆内存管理函数 malloc()和 free()的地址保存在一个结构中。寄生程序把这个结构体的指针传递给那些需要进行加载利用动态库的函数。dlopen()函数和 dlsym()函数返回的指针和上面的结构体一起保存在堆中的链表中。垃圾回收时只要简单地遍历链表，对每个加载的动态库调用 dlcose()函数，然后调用 free()函数释放掉每个链表结点即可。这样宿主映象就可以很容易地还原到原始状态。

2.2 实现

Linux下，由glibc进行动态连接。libdl中的dlopen()函数只是glibc中_dl_open()函数的包装[4]。因此，第一步需要定位的模块是glibc。

文件/proc/self/maps 描述了进程映象。通过分析该文件来对 glibc 模块定位。该文件是 ASCII 文件，下面是它的格式：

4001b000-400ff00 r-xp 00000000 03:01 390597 /lib/libc-2.1.3.so

第 1 个域是模块的起始地址是和最高地址，第 2 个域是模块的权限描述。如可读、可写、可执行和私有，分别用 r、w、x 和 p 表示。接下来的 3 个域和本文的讨论无关。最后的域是本文最关心的，它是该模块的完整路径。

下面是搜索 glibc 模块的算法：

```
for (i = 0; i < nread; i++)
{
start = end = buffer + i;
while ((*end++ != '\n') && (*end) && (i++ < nread)) ;
*end = 0;
for (ptr = end; (ptr > start) && (*ptr != ' '); ptr--)
if ((*ptr == *lib_name)) && (strncmp(ptr, lib_name, strlen(lib_name)) == 0))
return ((void *)strtol(start, NULL, 16));
}
```

buffer 是一个字符数组，里面保存了 read()函数从文件/proc/self/maps 中读入的内容。nread 是读入的字节数。buffer 被迭代处理，一直到所有的字节处理结束。start 指向正在处理的字符串的开始，指针 end 用来寻找字符串的结束。

这个算法首先提取出每一个字符串。这里的字符串指的是以回车换行符或 NULL 结束的字节序列。为了使用 strncmp()函数时比较方便，每个字符串被强行加上 NULL 结束符。

接着把提取出的字符串和要找的文件名进行比较。为了优化程序的性能，在调用 strncmp()函数前，先判断一下字符串的第一个字节是不是和要找的文件名的第一个字节相同。如果判断失败，则 strncmp()不被调用。否则调用 strncmp()来确定是不是要找的文件。当找到要找的文件时，调用 strtol()函数把文件加载地址转化为整数类型。至此算法结束。

Linux 下 dlsym()函数存在于文件 libdl 中。用处理 glibc 模块的方法同样可以找到 libdl 模块。并分析出 dlsym()函数的地址。

3 结束语

动态连接对寄生程序非常重要，可靠地利用动态库的机制可以显著地提高寄生程序的性能。本文采用了一种寄生程序通过 proc 文件系统进行加载和利用动态库的方法。通过本文的例子，显示了这种方法完全可行。利用这种方法，寄生程序就可以突破现有的很多限制。

参考文献

[1] Unix System Laboratories. Executable and Linkable Format(ELF) Pecification(Version 1.2)[S]. 1995.

[2] 毛 伟, 韩 冰, 席裕庚. Linux 下的动态连接库及其实现机制[J]. 计算机工程, 2000, 26(8): 112-114.

[3] The grugq. Cheating the ELF[Z]. (2001-01-01). http://madchat.org/coding/Cheating_elf.pdf.

[4] 何先波, 唐宁九, 吕 方, 等. ELF 文件格式及应用[J]. 计算机应用研究, 2001, 18(11): 144-145.

~~~~~~

（上接第 112 页）

(3)回到第(2)步，直到 $E' = E, E'' = \phi$，即所有边都已着色。

(4)安排教室。

由于第(2)步已经保证一定有教室可安排，因此这是一个从课程到教室的完全匹配问题。

根据实际情况，需要把教室分成不同种类，因此，在开课计划中对教室的使用必须指定教室的种类。在这种更细致的划分下，教室只与同色边集中那些只使用这种教室类型的边进行匹配，这时的三分图是原三分图的子图，即去掉非同色边、同色边中的非指定教室类型的边以及非指定类型的教室顶点的图的匹配。

为了更合理地使用教室，可以在实际应用中提出一些更优化的目标，在具体实践中加以完善，如：班级人数与教室容量越接近越好，前一个上课教室与后一个教室距离较近等。

4 结束语

本算法的主要时间花费是边着色(集中在冲突查找上)，理论时间复杂度是阶乘 $n!$。但是排课问题是一个面向实际的问题，除了考虑时间复杂度外还要考虑实际问题的解决程度。一般，各个学校的教师、班级、教室的数量都不算大，因此，仍可以在可容忍的限度内计算出结果，具有较强的实践意义。在目前还没有找到时间复杂度更低并且解决方案更完善的情况下，本算法不失为一个实用的算法。

参考文献

[1] 钟 声, 云 敏, 焦安全. 求解单圈多部图的匹配算法[J]. 广西师范大学学报: 自然科学版, 2007, 25(2): 202-205.

[2] Buckley F, Lewinter M. 图论简明教程[M]. 李慧霸, 王凤芹, 译. 北京: 清华大学出版社, 2005.

[3] 方剑英. 排课问题的理论与算法[D]. 乌鲁木齐: 新疆师范大学, 2004.

[4] 陶华亭, 张桃改. 基于图论方法的自动优化排课模型研究[J]. 微计算机信息, 2005, 21(7): 3-9.

[5] 张 健. 基于图论的高校排课系统实现[J]. 重庆师范大学学报: 自然科学版, 2005, 22(1): 35-38.
~~~~~~