**Linux so文件加壳程序**

1. **项目目的**

本项目主要是实现了对Linux中so文件的加壳功能，使市面上一些分析工具例如readelf、IDA等没法分析出其内在结构。这项技术适用于Linux系统，包括Ubuntu、Android等操作系统，能在一定程度上的保护软件的安全。

1. **技术实现基本思想**

Linux中so文件是ELF文件格式的，通过对ELF文件解析，对其中某个section进行加密。因为在程序加载so文件运行时内存中不是按section进行读取运行的而是以segment进行，所以在ELF头文件中关于section的信息基本没有什么作用，因此只需要完整的保存segment段的头信息。然后把加密的位置和大小存入到ELF头文件中关于section的字段中即可实现。

实现so文件解密就是通过之前记录在ELF头文件的字段得到加密section的位置和大小，从而进行解密。其中有两种方式，一是可以在外部对so文件进行解密然后加载到内存中使用，二是可以在内存中对so进行解密，这样就不会更改外部so的文件，这种方式更加安全和实用。

加密算法使用的是密钥与原始字符串相加的算法，把相加后的数字与255求余得到最终的加密后的字符串。

1. **技术分析细节**

3.1、ELF文件格式

ELF文件格式具有三种类型：可重定位文件、可执行文件、共享目标文件。这里我们分析的是共享目标文件，也就是so的文件格式。

下图是so的文件格式：



开头的ELF Header描述了体系结构和操作系统等基本信息,并指出Section Header Table和Program Header Table在文件中处于什么位置。左边是从链接角度来看，Program Header Table在汇编和链接过程中没有用到, 所以是可有可无的, Section Header Table中保存了所有Section的描述信息。右边是从加载器的视角来看这个文件, 开头是ELF Header, Program Header Table中保存了所有Segment的描述信息, Section Header Table在加载过程中没有用到, 所以是可有可无的。

 我们在汇编程序中用.section 声明的Section会成为目标文件中的Section, 此外汇编器还会自动添加一些Section(比如符号表)。Segment是指在程序运行时加载到内存的具有相同属性的区域, 由一个或多个Section组成, 例如有两个Section都要求加载到内存后可读可写, 就属于同一个Segment。有些Section只对汇编器和链接器有意义, 在运行时用不到, 也不需要加载到内存, 那么就不属于任何Segment。

3.2、重要文件头格式

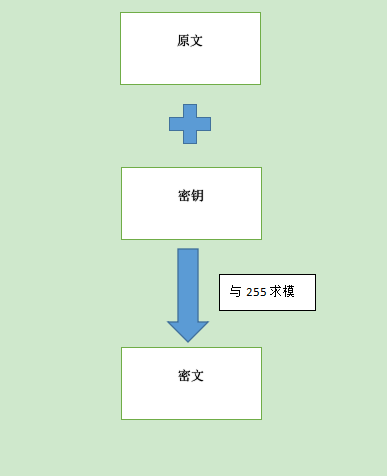
3.2.1、 ELF文件头格式:

|  |
| --- |
| typedef struct  {  unsigned char e\_ident[EI\_NIDENT]; /\* Magic number and other info \*/  Elf64\_Half e\_type; /\* Object file type \*/  Elf64\_Half e\_machine; /\* Architecture \*/  Elf64\_Word e\_version; /\* Object file version \*/  Elf64\_Addr e\_entry; /\* Entry point virtual address \*/  Elf64\_Off e\_phoff; /\* Program header table file offset \*/  Elf64\_Off e\_shoff; /\* Section header table file offset \*/  Elf64\_Word e\_flags; /\* Processor-specific flags \*/  Elf64\_Half e\_ehsize; /\* ELF header size in bytes \*/  Elf64\_Half e\_phentsize; /\* Program header table entry size \*/  Elf64\_Half e\_phnum; /\* Program header table entry count \*/  Elf64\_Half e\_shentsize; /\* Section header table entry size \*/  Elf64\_Half e\_shnum; /\* Section header table entry count \*/  Elf64\_Half e\_shstrndx; /\* Section header string table index \*/  } Elf64\_Ehdr; |

3.2.2、Section Header Table文件头格式

|  |
| --- |
| typedef struct  {  Elf64\_Word sh\_name; /\* Section name (string tbl index) \*/  Elf64\_Word sh\_type; /\* Section type \*/  Elf64\_Xword sh\_flags; /\* Section flags \*/  Elf64\_Addr sh\_addr; /\* Section virtual addr at execution \*/  Elf64\_Off sh\_offset; /\* Section file offset \*/  Elf64\_Xword sh\_size; /\* Section size in bytes \*/  Elf64\_Word sh\_link; /\* Link to another section \*/  Elf64\_Word sh\_info; /\* Additional section information \*/  Elf64\_Xword sh\_addralign; /\* Section alignment \*/  Elf64\_Xword sh\_entsize; /\* Entry size if section holds table \*/  } Elf64\_Shdr; |

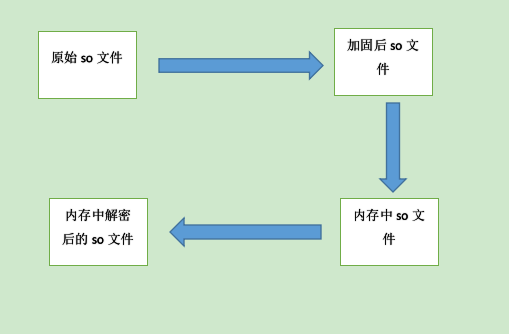
3.3、加密算法



用户输入一定长度的密钥，明文与密钥一一对应相加与255取模得到最终的密文，密钥整体不断的循环使用直至明文全部加密成密文，解密算法就是反向而行就可以了。

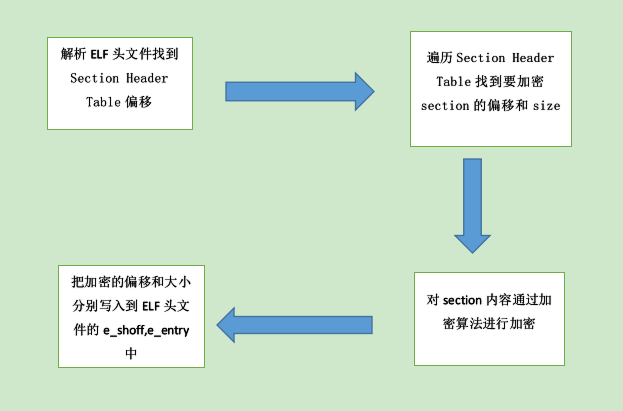
1. 技术实现

4.1、so文件加固与使用总体流程



4.2、加密

4.2.1、加密流程



ELF头文件中的e\_shoff，e\_entry都是与section有关的，但是在内存载入中这些已经没有用了，所以替换掉他们不会对so文件的载入使用造成影响也非常好寻找和定位。

4.2.1、so文件生成要求

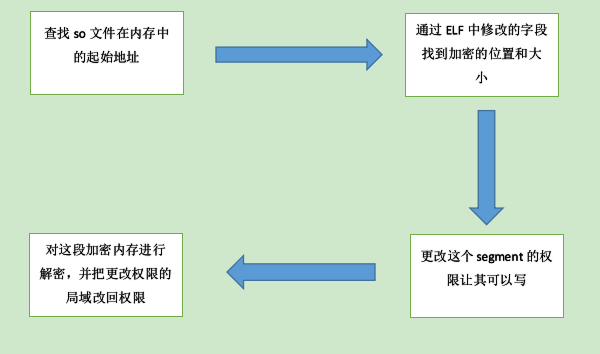
虽然这个加密可以实现对so文件进行加密并且在内存中对其进行解密运行，但是在内存中对其解密运行时由于自带的section都与加载有着密切的关系，所以在生成so文件时就要对特定重要的函数写入到一个特定的section中，然后对其进行加密。

例如：

|  |
| --- |
| \_\_attribute\_\_((section(“hackme”)) int myadd(int a, int b);  Int myadd(int a, int b)  {  Static int c = 10;  Return a + b + c  } |

在加密时候就对”hackme”这个section进行加密，然后在内存中对这个hackme的section段进行解密。

4.3、解密



4.3.1如何查找内存中so文件的起始地址：

通过/proc/pid/maps查找。/proc/pid/maps列出了某进程虚拟地址空间的“线性区”，只要遍历这个文件的每一行，查找是否出现so名，出现的话则读取该行第一串十六进制地址，这个地址即是so在进程虚拟空间的其实地址

4.3.2.如何对segment权限进行改变

进程地址空间被分为不同的线性区（segment），不同segment的权限不同，要被修改的地址必须要有写权限，才能进行section的解密。Linux中可以使用mprotect来修改内存权限，只要添加上写权限即可。有一点要注意：mprotect以页（4k）为单位，修改的起始地址和长度必须为页的整数倍

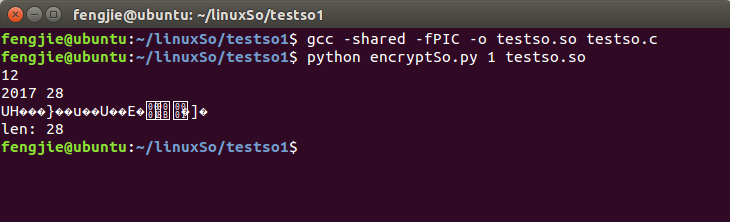
4.3.3. 内存中何时进行解密：

通过so文件中的init函数，把解密函数写入到这个初始函数中，这个函数的段会放在so文件的.init中，在so文件加载入内存之后就会运行这个解密函数进行解密，使函数在调用它时不会出错。

1. **作品展示**

这是一个简易的demo,主要就是主程序中通过调用so文件中的加法程序对三个数进行相加，然后so文件中的程序返回相加之后的数据并显示出来。

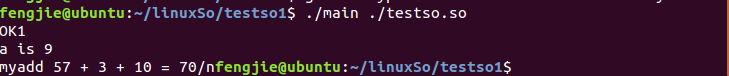
1. 生成so文件，并对其加密



2、生成主程序执行程序



1. 主程序调用so文件中的函数进行运算并输出



Demo文件介绍：

So文件生成：testso.c是用于so文件生成的。

主文件：mian.c 和decryptSo.c用于主程序的运行和内存中so文件的解密使用，由于 是demo所以解密函数单独拿出来，并且so通过注册函数对decryptSo.c文件 中的解密算法进行调用。

加密：encryptso.py文件用于so文件的加固，其中也包括了解密函数，用于测试。