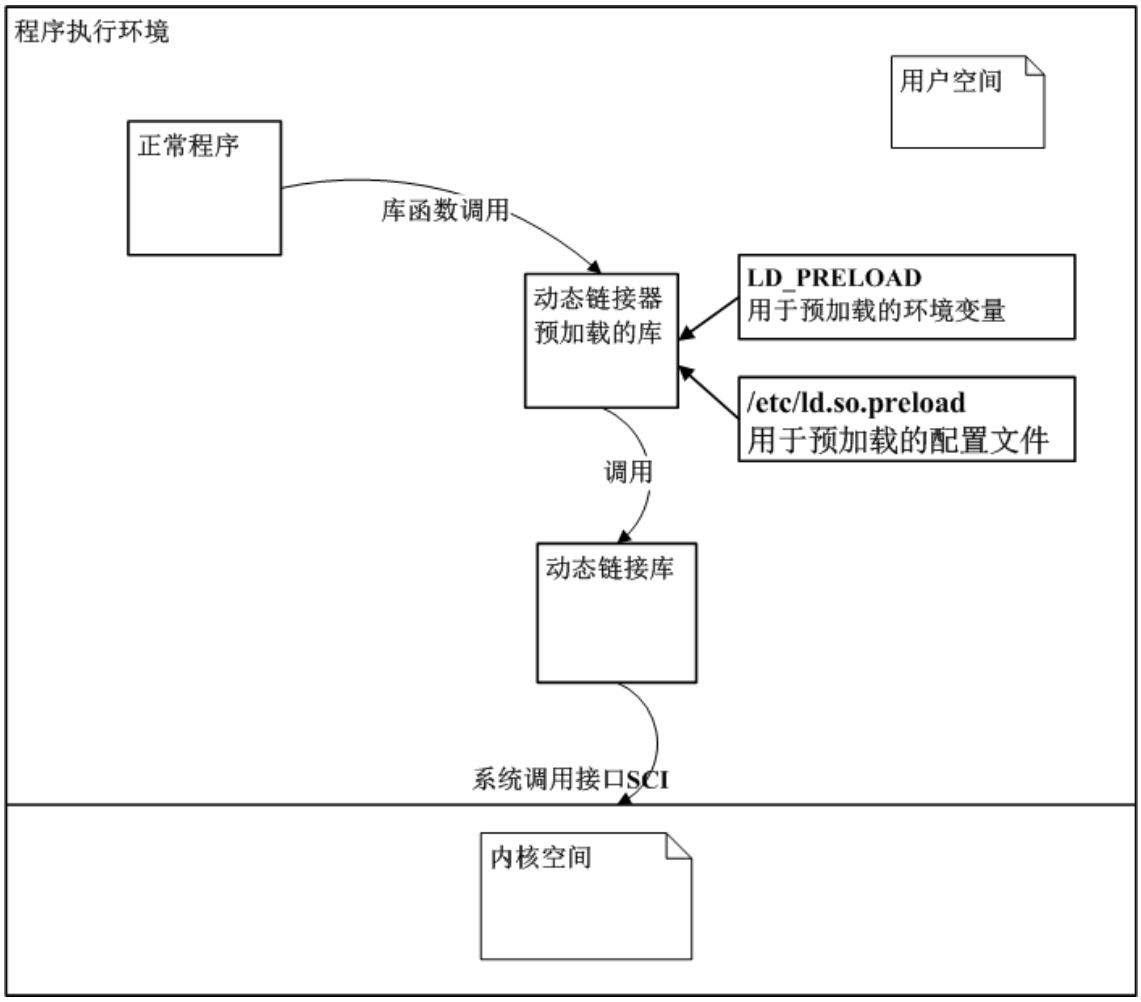
警惕利用 LINUX 预加载型恶意动态链接库的后门

文/太白遗风

0x00 动态链接库预加载型 rootkit 概述

动态链接库预加载机制是系统提供给用户运行自定义动态链接库的一种方式，在可执行程序运行之前就会预先加载用户定义的动态链接库的一种技术，这种技术可以重写系统的库函数，只需要在预加载的链接库中重新定义相同名称的库函数，程序调用库函数时，重新定义的函数即会短路正常的库函数，这种技术可以用来重写系统中有漏洞的库函数，达到修复漏洞的目的，如 get_host_byname 导致 ghost 漏洞的这类函数。这种技术也可以被不怀好意的攻击者用来写 rootkit，通过重写 mkdir, mkdirat, chdir, fchdir, opendir, opendir64, fdopendir, readdir, readdir64 等和系统文件，网络，进程相关的库函数来达到隐藏文件，进程的目的。相对于普通的用户空间 rootkit 而言，手段更加隐蔽，更加难以被发现，相对于内核模块 rootkit 来说，兼容性更好，编写难度更低，综合这两种优点，使得这类型 rootkit 逐年增多，难以查杀。

0x01 动态链接库预加载型 rootkit 所用技术



1. linux 动态链接库预加载机制

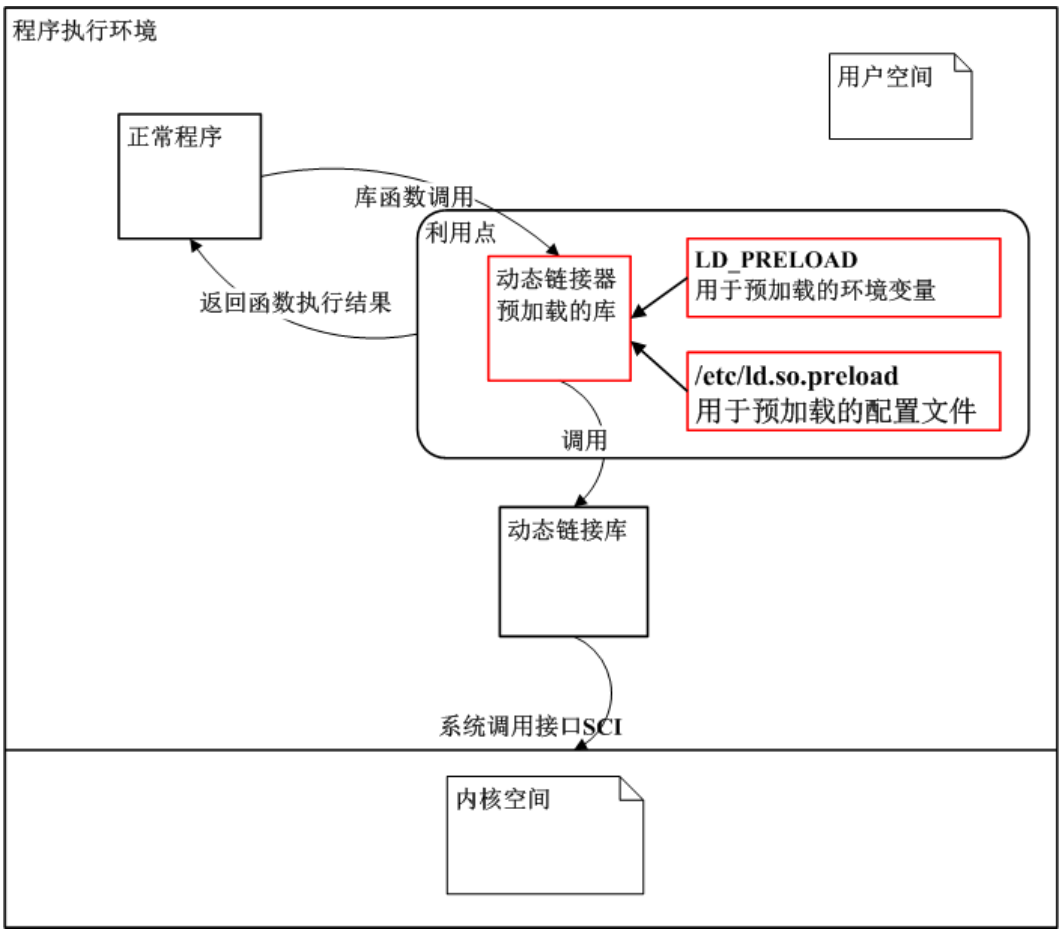
在 linux 操作系统的动态链接库加载过程中，动态链接器会读取 LD_PRELOAD 环境变量的值和默认配置文件/etc/ld.so.preload 的文件内容，并将读取到的动态链接库进行预加载，即使程序不依赖这些动态链接库，LD_PRELOAD 环境变量和/etc/ld.so.preload 配置文件中指定的动态链接库依然会被装载,它们的优先级比 LD_LIBRARY_PATH 环境变量所定义的链接库查找路径的文件优先级要高，所以能够提前于用户调用的动态库载入。

2. 全局符号介入

全局符号介入指的是应用程序调用库函数时，调用的库函数如果在多个动态链接库中都存在，即存在同名函数，那么链接器只会保留第一个链接的函数，而忽略后面链接进来的函数，所以只要预加载的全局符号中有和后加载的普通共享库中全局符号重名，那么就会覆盖后装载的共享库以及目标文件里的全局符号。

3. rootkit 利用的技术点

因为动态链接库预加载机制和全局符号介入这两种系统机制,可以控制程序运行时的链接（Runtime linker），允许用户在程序运行前优先加载自定义的动态链接库，使得恶意动态链接库优先于正常动态链接库加载，根据全局符号介入的顺序原理来”短路”正常函数,执行攻击者定义的恶意函数。



从上图中我们可以看到 3 种利用方式：

（1）将恶意动态链接库通过 LD_PRELOAD 环境变量进行加载。

（2）将恶意动态链接库通过/etc/ld.so.preload 配置文件进行加载。

（3）修改动态链接器来实现恶意功能，例如修改动态链接器中默认的用于预加载的配置文件路径/etc/ld.so.preload 为攻击者自定义路径，然后在里面写入要加载的恶意动态链接库，当然修改的姿势还有很多，如修改默认环境变量，直接将要 hook 的动态链接库写入到动态链接器当中。

0x02 动态链接库预加载型 rootkit

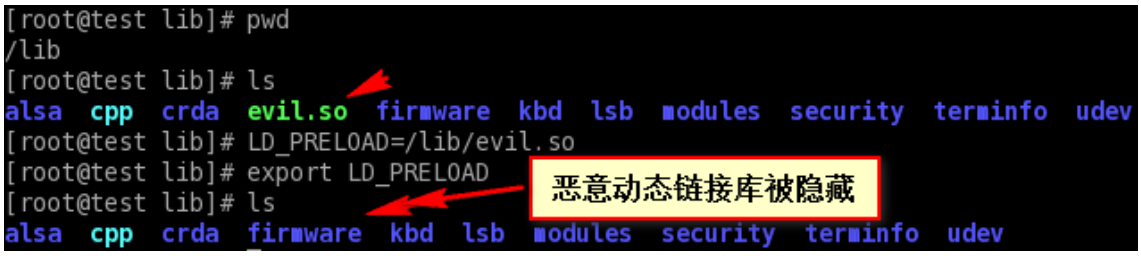
利用 LD_PRELOAD 加载恶意动态链接库

Step1. 安装

LD_PRELOAD 环境变量是会及时生效的，使用 LD_PRELOAD 加载恶意动态链接库方法如下：

```
LD_PRELOAD=/lib/evil.so          //LD_PRELOAD 的值设置为要预加载的动态链接库
export LD_PRELOAD                //导出环境变量使该环境变量生效
unset LD_PRELOAD                 //解除设置的 LD_PRELOAD 环境变量
```

测试使用的 rootkit 下载地址 https://github.com/mempodippy/cub3

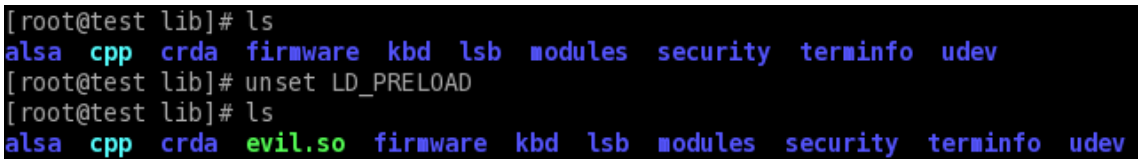


Step2. 检测

直接打印出 LD_PRELOAD 的值(默认 LD_PRELOAD 环境变量无值)，如果 LD_PRELOAD 中有值，则将该文件上传到 virustotal 或微步在线等恶意软件检测平台检测该文件是否正常，或者用自制的特征进行匹配或者人工 strings 或者用 ida 看一下，即可判断出是否是恶意程序。

Step3. 清除

使用命令 unset LD_PRELOAD 即可实现卸载使用 LD_PRELOAD 环境变量安装的恶意动态链接库。如下图，可以看到被隐藏的文件 evil.so 显示出来了。



利用 /etc/ld.so.preload 加载恶意动态链接库

Step1. 安装

将恶意动态链接库路径写入/etc/ld.so.preload(没有则创建)配置文件中，如下图所示将恶意动态链接库路径写入/etc/ld.so.preload 文件中即生效，对应的恶意动态链接库文件被隐藏。

```
[root@test lib]# ls
alsa  cpp  crda  evil.so  firmware  kbd  lsb  modules  security  terminfo  udev
[root@test lib]# cat /etc/ld.so.preload

[root@test lib]# echo "/lib/evil.so" > /etc/ld.so.preload
[root@test lib]# ls
alsa  cpp  crda  firmware  kbd  lsb  modules  security  terminfo  udev
```

Step2. 检测

因为恶意动态链接库一般都有隐藏 /etc/ld.so.preload 文件的功能，我们使用普通的 ls,cat 等命令无法读取对应配置文件的内容，此时我们可以使用静态编译的 ls 命令,cat 命令（推荐使用 busybox 自带命令)来绕过预加载的恶意动态链接库，如果没有 ls 命令 cat 命令，有时候将 ls 命令和 cat 命令改成其他任意字符也可以绕过恶意动态链接库的隐藏，主要得看恶意动态链接库具体实现方式。

如下图，通过使用普通的 cat 命令和 busybox 中的 cat 命令查看/etc/ld.so.preload 文件内容对比，即可判断出是否有通过/etc/ld.so.preload 配置文件加载的恶意动态链接库。

```
[root@test lib]# cat /etc/ld.so.preload
cat: /etc/ld.so.preload: No such file or directory
[root@test lib]# busybox cat /etc/ld.so.preload
/lib/evil.so
```

Step3. 清除

因为恶意动态链接库具有隐藏文件的功能，所以清除时需要使用静态编译的基础命令来进行对应操作，清除过程如下所示。

首先清除上方/etc/ld.so.preload 文件中查看到的/lib/evil.so 文件，使其无法正常预加载，然后清除/etc/ld.so.preload 中的恶意文件内容，有的恶意动态链接库会修改该文件的隐藏权限，以及普通的读写权限，所以需要看一下，然后再清除，到此为止即清除成功。(因为我这里以前就没有配置预加载的库，所以直接清空，如果你们有业务配置了预加载的库则需要清除特定行，而不是直接清空)

```
[root@test lib]# mv evil.so test.so
[root@test lib]# ls
ERROR: ld.so: object '/lib/evil.so' from /etc/ld.so.preload cannot be preloaded:
 ignored.
alsa  cpp  crda  firmware  kbd  lsb  modules  security  terminfo  test.so  udev
[root@test lib]# busybox lsattr /etc/ld.so.preload
------------ /etc/ld.so.preload
[root@test lib]# echo ""> /etc/ld.so.preload
```

修改动态链接器来实现恶意功能

Step1. 安装

修改动态链接器实现恶意功能目的有多种方法，这里使用替换动态链接器中的默认预加载配置文件/etc/ld.so.preload 路径的 rootkit，来实现更加隐蔽的恶意动态链接库预加载,安装

的方法是修改动态链接器中配置文件路径/etc/ld.so.preload 为自定义的路径，然后再在该路径中写入要预加载的恶意动态链接库的绝对路径。测试使用的恶意 rootkit 名称为 Vlany，下载地址为 https://github.com/mempodippy/vlany 。

Step2. 检测

修改默认动态链接器来达到实现恶意功能的目的会破坏原有动态链接器的完整性，我们可以使用文件完整性检查来检查该动态链接器是否被修改。

首先获取系统中的动态链接器的文件路径，然后判断该动态链接器文件的完整性。这里测试系统是 centos，自带 rpm 校验功能，下图中的 5 指的是文件的 md5 发生了改变，T 指的是修改时间发生了改变。

```
[root@test home]# readelf -a /bin/ls |grep interpreter
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
[root@test home]# ls -la /lib64/ld-linux-x86-64.so.2
lrwxrwxrwx 1 root root 10 Jan 12 00:48 /lib64/ld-linux-x86-64.so.2 -> ld-2.12.so
[root@test home]# rpm -Vf /lib64/ld-2.12.so
..5....T.    /lib64/ld-2.12.so
```

如果知道了动态链接器被修改过，(排除系统升级导致)那么可以判断动态链接器存在较高的安全风险，我们需要对该修改进行进一步确认,如果攻击者修改动态链接器但是实现恶意功能的方式不是修改了预加载配置文件，而是修改了默认的环境变量，或者直接根据开源代码将恶意功能植入动态链接器然后重新编译生成的恶意动态链接器，那么下面的检测方法可能是无效的,需要视情况分析。

使用 strace 命令来查看预加载的配置文件是不是 /etc/ld.so.preload 文件，如下图，动态链接库预加载的配置文件是 /sbin/.XsknPn3F 而不是原有的配置文件，我们即可确认系统中存在修改动态链接器的 rootkit。

```
[root@test ~]# strace -f -e trace=file /bin/ls
execve("/bin/ls", ["/bin/ls"], [/* 41 vars */]) = 0
access("/sbin/.XsknPn3F", R_OK)         = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
open("/lib64/libselinux.so.1", O_RDONLY) = 3
open("/lib64/librt.so.1", O_RDONLY)     = 3
open("/lib64/libcap.so.2", O_RDONLY)    = 3
open("/lib64/libacl.so.1", O_RDONLY)    = 3
open("/lib64/libc.so.6", O_RDONLY)      = 3
open("/lib64/libdl.so.2", O_RDONLY)     = 3
open("/lib64/libpthread.so.0", O_RDONLY) = 3
open("/lib64/libattr.so.1", O_RDONLY)   = 3
```

/sbin/.XsknPn3F为攻击者修改动态链接器指定的链接库预加载配置文件，使得运行程序时不再去加载/etc/ld.so.preload文件中的动态库

使用 busybox 自带的 cat 命令查看该文件，因为使用正常 cat 命令无法查看该文件，被预加载的库函数隐藏了。

```
[root@test ~]# busybox cat /sbin/.XsknPn3F
/lib/libc.so.root.11/t4dGiRfcOsb9.so.$PLATFORM[root@test ~]#
[root@test ~]# cat /sbin/.XsknPn3F
cat: /sbin/.XsknPn3F: No such file or directory
[root@test ~]#
```

```
[root@test ~]# busybox cat /sbin/.XsknPn3F
/lib/libc.so.root.11/t4dGiRfcOsb9.so.$PLATFORM[root@test ~]#
```

Step3. 清除
清除修改动态链接器的 rootkit，需要使用相同系统的相同版本动态链接器替换被修改了的动态链接器，才能达到彻底清除的目的,暂时缓解的方式则是将上方检测过程中看到的恶意动态链接库删除，以及将对应的动态链接库配置文件中的内容清除。

0x03 通用检测方法总结

根据动态链接库预加载机制可知，预加载型恶意动态链接库只对需要使用动态库函数的程序有效，恶意动态链接库基本功能隐藏文件，根据隐藏文件功能的特点，检测的思路是应用程序尽量不使用动态库，即可绕过这个文件隐藏的功能。

交叉试图：
使用普通 ls 命令对特定目录下文件进行查看，使用静态编译的基础命令来对特定目录文件进行查看，必看文件/etc/ld.so.preload 经典目录/lib/判断是否有隐藏文件，应急响应时推荐使用 busybox，或者自己静态编译的 ls 等命令。
动态链接库预加载机制会读取预加载配置文件内容，然后加载配置文件中对应的动态链接库，根据这一特性，可以通过跟踪常用命令执行过程中加载的文件，来判断是否存在恶意动态链接库。

strace 文件跟踪：

可执行程序运行时，首先会访问动态链接库预加载配置文件，然后读取对应配置文件中的动态库，并预先加载，然后再去加载正常所需链接库，通过跟踪系统/bin/ls 打开的相关文件，即可找到预加载配置文件以及被预加载的动态链接库，如果恶意动态链接库有反 strace 措施，可以修改 strace 名称或使用 LD_PRELOAD 环境变量预先加载一个无关动态链接库，然后再 strace 进行跟踪，可以绕过恶意动态链接库的根据可执行程序名称检测的反 strace 措施。

文件完整性检查：
部分攻击者通过修改动态链接器的方式实现恶意功能的目的，但这种做法会破坏动态链接器的完整性，通过检测动态链接器的完整性即可检测出修改动态链接器型 rootkit。

0x04 参考链接

https://github.com/mempodippy/vlany

https://github.com/mempodippy/cub3