Linux/Unix环境中ELF格式病毒的分析

谢瑶[(1)] 潘剑峰[(2)] 朱明[(2)]

（中国科学技术大学 [(1)] 电子工程与信息科学系，[(2)] 自动控制系 安徽合肥 230027）

摘要：本文结合 Linux/Unix 病毒实例，分析了 Linux/Unix 系统 ELF 文件格式病毒的原理，并从五个方面讲述实现技术的关键：病毒体寄生 ELF 文件技术，寄主程序入口重定位，病毒体地址无关代码，LKM 可重载内核技术，以及网络功能的实现。最后，提出对这一类病毒检测与防范的方法及策略。

关键词：Linux/Unix 系统，ELF 文件病毒，病毒寄生，地址无关代码（PIC），可重载内核 LKM

Analysis of ELF Virus in Linux/Unix Systems

Yao Xie, Jianfeng Pan, Ming Zhu

(Department of Electronic Engineering and Information Science, Department of Automatic control, University of Science and Technology of China, Anhui, 230027)

Abstract: The technical details about the infection mechanism of ELF virus are studies by analyzing a specific case of ELF virus. Five critical technical aspects of ELF virus are analyzed which include: parasite in host file, redirect executive entry of host to realize virus self-execution, PIC (Position Independent Code), LKM (Loadable Kernel Modules) and Internet propagation of virus. Finally, we presented several practical strategies for virus detection and protection.

Key Words: Linux/Unix System, ELF virus, Virus Parasite, PIC (Position Independent Code), LKM (Loadable Kernel Mode)

1 引言

Internet的飞速发展，使得电脑病毒也呈现出传播更快、破坏性更大的发展趋势，由此造成巨大的经济损失并直接危害社会的各个方面。目前Linux服务器环境越来越流行，因此分析研究Linux环境下病毒实现机制，探索相应的检测方法，已变得十分迫切。

攻击Linux的最常用技术之一就是使用内核代码，即使用可加载内核模块（LKM）。另外，Linux系统通用的ELF可执行文件格式【1】在提高系统程序可移植性的同时，也带来了易被病毒感染的不安全因素。本文结合一个Linux/Unix环境下ELF格式病毒的实例【2】，分析此类病毒的实现原理，给出了相应的检测与防范对策。

2 Linux/ Unix 病毒特点

Linu/Unix病毒一般具有三种功能：直接寄生宿主文件，感染系统内核，网络升级。本文实例具有此类病毒的典型特点：寄生性，传染性以及自启动性。因此，首先病毒须以某种方式寄生于宿主，这就意味着病毒需要修改宿主插入自身代码。当被感染宿主执行时，宿主源代码与病毒体共同映射到程序进程空间，病毒抢先于宿主执行，之后将控制权交还宿主。其次，病毒本身具有感染性，被感染寄主程序也能进一步感染同类文件，使病毒在本地目录迅速传播。其三，病毒体使用可加载LKM重载系统调用，很容易被多次触发执行；病毒还可通过修改系统参数隐藏自身不被ls, #lsmod等工具检测到，具有很大的隐蔽性【3】。除此而外，病毒具有网络升级功能【4】。

本文将围绕分析实现以上功能的关键技术展开详细叙述。

3 ELF文件格式

3.1 ELF 文件分类

ELF（Executable Linking Format）即可执行链接文件格式，在Unix/Linux 系统，以及SVR4和Solaris 2.x系统上，作为默认的可执行文件的格式。目前标准接口委员会TIS已将ELF 标准化为一种可移植的目标文件格式 (portable object file format，运行于32位Intel体系微机上，可与多种操作系统兼容。

ELF文件格式优点在于提供了统一的编程接口，便于程序设计者使用统一的二进制文件接口，在多操作系统的环境下，流水线化程序开发过程。减少了在不同的微机体系下重新编码，编译带来的工作量。

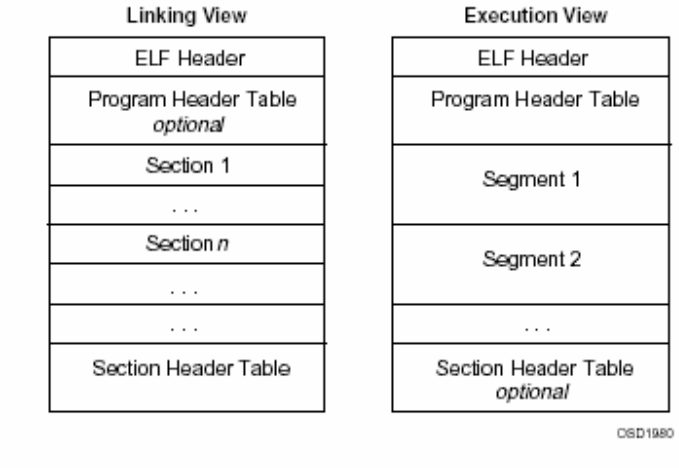
ELF文件格式分为以下的三种，它们具有ELF 的一般结构，但在程序段（segment），节（section）的组织，头表（header）的结构上存在细微差别。如表一所示：

表一、ELF 文件的分类及实例

ELF文件类型	内容	实例
可执行文件（executable file）	可执行的代码和相关数据。	# file dltest dltest: ELF 32-bit LSB executable, Intel 80386, version 1,dynamically linked (uses shared libs), not stripped
可重定位文件(relocatable file)	可执行代码以及和其他重定位文件和共享的object文件一起连接时使用的数据	# file libfoo.o libfoo.o:ELF 32-bit LSB relocatable, Intel 80386, version 1, not stripped
共享object文件（共享库）(shared object file)	可执行代码以及被连接器ld和动态连接器使用的数据。 注：动态连接器可能称为ld.so.1,libc.so.1 或者ld-linux.so.1。	# file libfoo.so libfoo.so:ELF 32-bit LSB shared object, Intel 80386, version1, not stripped

3.2 ELF文件一般结构

为了便于讨论，我们按ELF文件结构中各个部分的不同功能，从链接和执行两个不同角度分别描述。如图一示：



图一. ELF 文件一般组成结构

表二. ELF 文件各部分功能表

ELF 头	在程序的开始部位，作为“引路表”描述整个 ELF 的文件结构

程序头表	为加载器（Loader）提供如何将程序加载到内存中的信息
节头表	描述程序节，为编译器（compiler）和链接器（linker）提供信息

其中，程序段（segment）是由程序头表（program header）描述的单位；程序节（section）是由节头表（section header）描述的单位。程序节(section)保存着链接器所用信息，包括：程序指令，数据，符号表，重定位信息，字符表等等。而程序段(segment)主要是从文件（file）如何映射（mapping）到内存的角度来划分的。相应部分的功能如表二所示：

以上的三个部分决定目标文件如何与其他目标文件连接，载入内存，以及如何执行。因此病毒体寄生宿主，驻留内存的关键在于修改寄主的ELF 头，程序头表，节头表信息，然后插入自身代码于寄主文件体中。如，ELF头的数据结构见图二：

```
#define EI_NIDENT       16

typedef struct {
          unsigned char  e_ident[EI_NIDENT];
          Elf32_Half     e_type;
          Elf32_Half     e_machine;
          Elf32_Word     e_version;
          Elf32_Addr     e_entry;
          Elf32_Off      e_phoff;
          Elf32_Off      e_shoff;
          Elf32_Word     e_flags;
          Elf32_Half     e_ehsize;
          Elf32_Half     e_phentsize;
          Elf32_Half     e_phnum;
          Elf32_Half     e_shentsize;
          Elf32_Half     e_shnum;
          Elf32_Half     e_shstrndx;
} Elf32_Ehdr;
```

图二：ELF 头的数据结构

其中病毒将使用的域为：

[e_ident] 标示该文件为一个 ELF 文件，并提供一个机器无关数据，解释文件内容。

[e_entry] 系统执行程序代码的起始内存地址

[e_phoff] 程序头表（program header table）在文件中的偏移量(以字节计数)

[e_shoff] 节头表（section header table）在文件中的偏移量(以字节计数)

3.3 两个重要的 section

图一ELF 文件体一般结构中还有两个重要的部分：代码节、数据节，只有这两个节存在于静态文件映像中。

- 代码节 .code

代码节只包含只读数据. 除了程序代码外，在运行过程中保持不变的数据也位于代码段，如ASCII字符串；ELF头部信息也位于代码段首部；不同进程可共享代码段。

- 数据段 .data

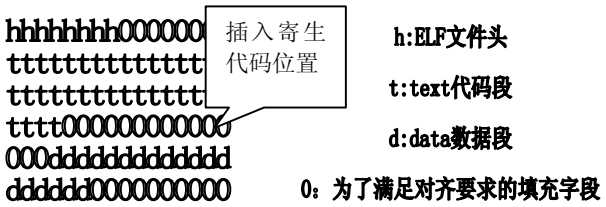
数据段包括初始化过的数据、未初始化过的数据以及动态内存分配所使用的堆区(Heap)。它们各自占用了数据段的不同部分，并不重叠，但是访问权限相同，具有读写属性。

3.4 对齐（Align）原则 — 可利用的特点

ELF 规范中有如下要求：p_vaddr (mod PAGE_SIZE)＝p_offset (mod PAGE_SIZE)，因此，寄生部分的大小为页整数倍，不足时辅以填充字段。例如：一个 ELF 的程序代码段（.code）

就可能有这样的结构，如图三所示。

在填充 0 的部分可以写入病毒体而不改变文件结构。有一类 ELF 病毒是这样实现的，但病毒大小受填充（padding）的大小限制。



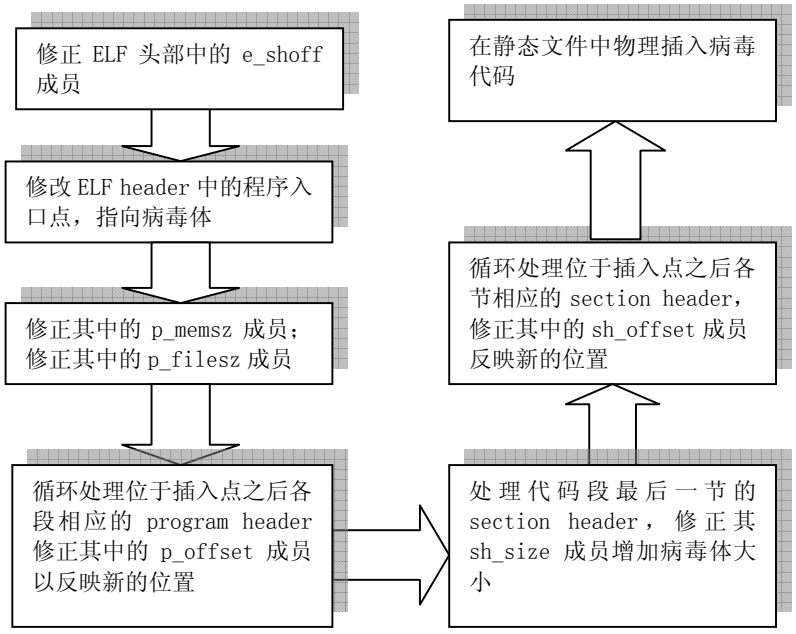
图三、病毒寄生在 ELF 填充字段

4 病毒的寄生方式

一般来说，病毒寄生四种主要方式：

- 病毒直接覆盖宿主，受感染后的host被破坏，即覆盖可执行文件，破坏原始数据。这种传染方式将导致下次宿主程序执行失败，病毒很易被发现。如果被破坏的宿主是系统赖以生存的重要文件，将导致整个系统崩溃。
- 向宿主文件中插入病毒体，不破坏宿主主体，修改程序流程，以便病毒跟随宿主一起执行。寄生病毒不改变宿主内部对象格式，在自身得到执行后将控制交还给宿主。病毒体可以通过修改寄主t某个段的大小，寄生在寄主的代码段前、数据段后、代码段和数据段中间的填充区。但这样会改变原有文件的格式，比较容易检测到。
- 病毒合入宿主进程映像中，用病毒替换宿主进程映像，之后恢复，在宿主进程映像首部或尾部植入病毒代码，覆盖进程映像各段之间的填充区域等等。
- 利用ELF中本身有的填充（padding）字段，将virus植入，不改变原有程序段的大小，不过这样有明显缺点：填充字段一般小于64byte，只能植入较小的病毒体，如时间炸弹。

本文中的病毒实例【5】采取的方式是在数据段尾部插入病毒体，并相应后移静态文件插入点之后的部分。这将改变了 ELF 文件布局，必须修改 ELF 头部及各表中相关信息。病毒代码实现寄生原理的流程如图四所示：

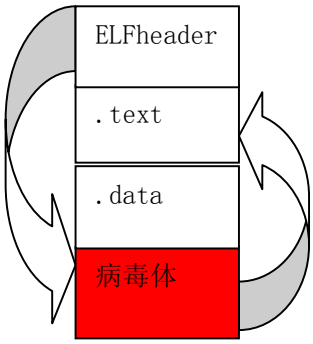


图四. 病毒寄生流程图

做了如上修改后，病毒寄生，以及自启动功能已经完备。但一个问题在于，植入病毒体后的程序入口点位于数据段尾部，病毒检测程序就可以利用利用这点(程序入口点不在.init 节)检测出病毒。

5 病毒自启动实现

正如已经提及的，为了实现自启动病毒需要修改寄主 ELF 头入口点(entry point)为病毒的代码执行起始地址，被感染寄主文件将按照病毒－寄主的顺序执行。修改后的寄主程序运行顺序如图五所示：

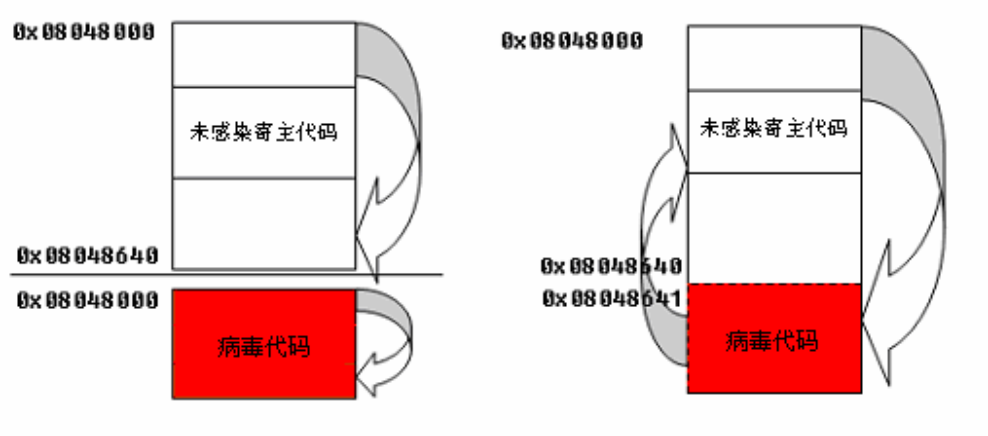


图五 受感染寄主程序运行顺序

6 地址无关代码 (PIC)

所谓地址无关代码 PIC（Place Independent Code）【4】,就是其变量，标号，以及调用函数不使用编译时生成内存地址，因此能够加载入内存中的任何位置执行的代码。由于病毒体附着在寄主代码后，其在内存中的加载位置就要决定于寄主的加载位置以及寄主 ELF 文件的大小。因此，一般病毒代码实现常使用以下两种手段实现地址无关代码。

其一，由于寄生后病毒时在内存中加载位置不定，病毒代码中需要根据这一偏移（offset）对引用的变量地址作相应的修改。我们的系统中默认为将运行的进程加载到以 0x08048000 开始的一段内存地址。产生这一问题的说明如图六所示：



图六、病毒加载位置不定问题的产生

为了解决加载位置不定的问题，病毒体代码需要找出偏移值（offset）的大小，并将引用的变量的地址都加上这个偏移。

问题二在于，由于病毒体事先不可能得知系统中 C 运行库的位置，不能像普通程序那样直接调用 C 运行库。Linux 操作系统有一些嵌在内核中可被系统的任意操作使用的函数，称为系统调用，其位置位置是固定的，对应于操作系统用户执行态（User Mode）和内核执行态（Kernel Mode）之间的转换。内核中存在结构数组 Sys_call_table[]，系统调用号作为数组的索引。调用时系统调用号和其他参数放入相应寄存器中，int 0x80 可得到所需服务函数。用户系统完全的系统调用列表可参见 /usr/include/sys/syscall.h 文件。

7 LKM 以及网络功能

最后值得一提的是病毒实例的两个特点。其一是，病毒通过 LKM 内核重载技术，重载系统调用会具有更强大的功能：首先感染模块，然后通过系统调用我们重载的 LKM 模块，进一步用文件感染方法。其二，病毒可以通过使用 Linux 中的网络功能系统调用，实现从与指定 IP 服务器建立连接，取得新的病毒体的功能。这就相当于为病毒的释放者开了后门，可以随时为病毒提供“升级”。

8 Linux/Unix病毒的防范

本文中的病毒实例结合了 Linux 系统中几种可能的病毒感染原理。由于病毒体在寄生在文件内部，不易发现；感染的寄主具有传染性，被感染的内核等都会加速病毒的传播；网络后门功能这样病毒一旦进入系统内部长期隐藏，后果不堪设想。

通过以上的分析，可得以下的一些有效得安全检测与防范措施：

1）由于病毒体的寄生增加了寄主数据段大小，可用 ls –l 等命令人工检测到。此外，也可用 strip 命令优化程序，去掉容易寄生病毒的.note .debug 段.也可使用 ls 对应的系统调用 sys_getdents 编程实现检测文件的大小。

2）病毒的寄生会使程序的入口有所变化，病毒检测程序可以利用程序入口点不在.init 节检测出。

3）由于 Linux/Unix 系统中的权限限制，可以使病毒不能感染写权限以外的程序。所以加强系统管理，限制用户之间文件的非法拷贝，防止病毒获得 root 权限，可限制病毒的传染规模。

4）通过 Netlog 记录，lsof 等，严格监视到到未知 web server 的连接，防止病毒通过网络更新，或引入新的病毒体。

5）清除病毒：清除掉检测出的被感染的寄主程序，以及病毒体文件。被感染的模块需要用解除重载。

9．结束语

Linux/Unix 系统安全性是目前系统安全研究的一大重点。由于 Linux 系统本身源码开放，并且允许用户重载内核，造成了很大的不安全因素。虽然当前 Linux 病毒的种类还不是很多，随着 Linux/Unix 系统日益广泛的使用，系统管理员急需加强警觉，防患于未然。本文解析了一个典型的 Linix/Unix 系统 ELF 病毒实现的机理，探讨了相应的检测与防范策略。具有新型功能的 Linux/Unix 病毒，以及相应的新的防范技术，仍将是今后研究的重点。

参考文献：

【1】Tool Interface Standard (TIS), Executable and Linkable Format (ELF) Specification，version 1.2, 1995，Charpter 1

【2】Martin Karresand, Separating Trojan Horses, Viruses, and Worms – A Proposed Taxonomy of Software Weapons, IEEE, 2003, charpter 1

【3】Pragmatic/THC, Complete Linux Loadable Kernel Modules, version 1.0, http://www.vxheavens.htm , 1999, charpter 3

【4】Unix ELF parasites and virus，Silvio Cesare, http://www.vxheavens.htm , 1998 AT&T

【5】Cesare,SILLOV 病毒源码, http://www.big.net.au/~silvio , 1999

第一作者简介：

谢瑶，女，1982 年出生，中国科学技术大学电子工程与信息科学系四年级本科生，主要研究兴趣： 通信中的信号处理，网络通信。

其他作者：

潘剑峰，男，1981 年出生，中国科学技术大学自动化系五年级本科生。

朱明，男，1963 年出生，中国科学技术大学自动化系副教授。