第32卷 第21期 计 算 机 工 程 2006年11月
Vol.32 No. 21 Computer Engineering November 2006
·安全技术· 文章编号：1000—3428(2006)21—0152—03 文献标识码：A 中图分类号：TP391

基于 MAC 的 ELF 文件执行安全性的提高

曾勇军，朱俊虎，吴 瀛
（解放军信息工程大学信息工程学院，郑州 450002）

摘 要：随着 Linux 操作系统的应用与普及，其安全性也受到人们的广泛关注。在描述了消息鉴别码(MAC)计算和 ELF 文件格式之后，提出了保证 ELF 文件安全执行的方法，并对存在的问题进行了分析，给出了可行的解决办法。实验结果表明该方法是实际可行的。
关键词：消息鉴别码；ELF；可加载内核模块

Enhancement of ELF Executable Security Based on MAC

ZENG Yongjun, ZHU Junhu, WU Hao
(College of Information Engineering, PLA Information Engineering University, Zhengzhou 450002)

【Abstract】The security of Linux attracts a comprehensive attention with its application. The MAC computation and ELF file format are described, and the method that ensures the secure execution of ELF file is presented, the existing problems are analyzed and some possible solutions are given. Experimental results show that the method is feasible.
【Key words】Message authentication code; Executable and linkable format(ELF); Loadable kernel module

在操作系统中，文件类型多种多样，以满足不同应用的需求。相对于其它文件类型而言，可执行文件是操作系统中最重要的文件类型之一，它包含有指令和数据，是操作的真正执行者。因此，研究可执行文件的格式并构造恶意代码，成为黑客攻击的重要手段之一。Linux 操作系统由于使用方便、功能丰富并且是开放源码等特点，使之在操作系统领域得到了越来越广泛的应用，其安全性也受到越来越多的挑战，而对 Linux 可执行文件的攻击也成为黑客攻击的主要方向。

ELF（Executable and Linkable Format）是 Linux 操作系统中最主要的可执行二进制文件格式。如何保证 ELF 执行的安全性，已经成为研究的热点问题，目前已经出现了一些研究成果。业界著名的 Triewire 就是使用得较多的验证文件完整性的工具，在系统初装时为指定文件生成完整性校验值，运行过程中验证文件的合法性，提供专用安全数据库存放完整性校验值，但该工具只提供了文件内容的定期检查，不能实时监控，在验证时机上存在比较大的问题。文献[1]提出了 ELF 可执行文件签名验证机制，使用 PKCS#7 格式存放签名，X.509 格式存放公钥，由内核验证 ELF 可执行文件，动态库放在用户空间验证，该文提出的方法需要修改动态库的解释器，没有考虑可加载内核模块的安全性，并且签名验证时使用公钥，验证的开销比较大。文献[2,3]改进了文献[1]提出的方法，可执行文件的完整性在内核验证，无需修改解释器，但验证过程同样需要使用公钥，计算的开销也比较大。

本文提出了利用 MAC 提高 ELF 文件安全执行的方法。通过劫持系统调用，所有 ELF 文件的完整性验证均在内核空间执行，能够检测各种类型的 ELF 文件，同时由于 MAC 在计算上的开销比 RSA 等数字签名/验证算法要低，在满足安全性的条件下兼顾了系统的运行效率。

1 MAC 简介

完整性是安全的基本要求之一，在开放的系统中，提供一种途径去检测存储信息的完整性是非常重要的。提供这种完整性检测的机制基于一种通常被称作消息鉴别码（Message Authentication Code, MAC）来实现，如图 1 所示。

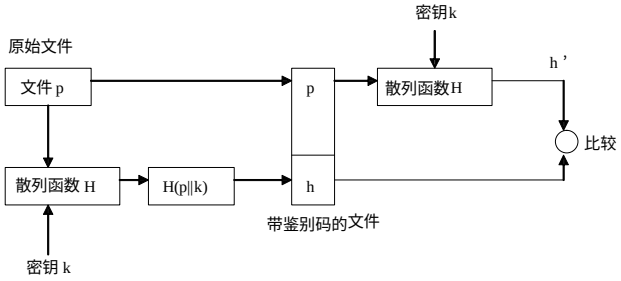

图 1 MAC 完整性计算及验证过程

MAC 算法流程描述如下：

(1)假定用户希望执行程序 p，如果没有完整保护，则攻击方可在 p 中附加恶意代码，系统不能检测。为了提供完整性保护，对程序 p 作如下计算：

$h = H(p||k)$

其中：H 为单向散列函数；||表示两个串首尾相连；k 为秘密密钥；计算的结果 h 为鉴别码，附加在文件末尾。

(2)当带鉴别码的程序 p 加载时，由内核读出文件内容 p 和鉴别码 h。调用散列函数重新计算鉴别码，得到 h’。

(3)比较 h 与 h’，若相等，则说明文件没有被篡改和伪造，可以继续执行；若不相等，则终止执行，报错退出。

在鉴别码计算的过程中引入了密钥，攻击者在没有获得该密钥的情况下不能伪造鉴别码，从而保证程序执行的安全。

2 带鉴别码的 ELF 文件格式

2.1 ELF 文件格式

ELF（Executable and Linkable Format）是由 Unix 系统实

基金项目：国家“863”计划基金资助项目（2003AA146010）
作者简介：曾勇军(1973－)，男，讲师，主研方向：网络信息安全；朱俊虎，讲师、硕士；吴 瀛，教授、硕士
收稿日期：2005-11-28 **E-mail：**zyj216@china.com.cn

验室设计的，现在已成为 Linux 最常用的二进制的可执行文件格式。RedHat7.0 以上版本就已经将可执行的文件格式默认为此格式了。与其它的文件格式（如 ECOFF，a.out）相比，ELF 虽然在性能上有轻微的开销，但它给人的感觉更灵活。感染 Linux 下文件的病毒大多数都是感染这种文件的。ELF 文件格式如图 2 所示。

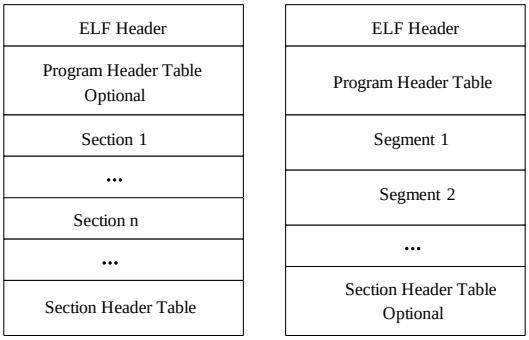

图 2 ELF 文件格式

ELF 文件可从两个视图来描述：链接视图和执行视图。从链接的角度，我们关注节头表，由节头表的指引把各个节连接组装起来；从装载运行角度，我们关注的是程序头表，由程序头表的指引把 ELF 文件加载进内存运行它。

ELF 文件头是一个关于本文件的路线图（road map），从总体上描述文件的组织情况。根据 ELF 文件头，可知 ELF 文件有 3 种主要的类型。

(1)可重定位（relocatable）文件：文件保存着代码和适当的数据，用来和其他的目标文件一起创建一个可执行文件或者是一个共享文件。

(2)可执行文件（executable）：文件保存着一个用来执行的程序，该文件指出 exec 如何来创建进程映像。

(3)共享目标文件：存放代码和数据，用来被以下两个链接器链接：第 1 个是连接编辑器(ld)，可以和其他的可重定位共享文件来创建其它的目标。第 2 个是动态链接器，联合一个可执行文件和其他的共享目标文件来创建一个进程映像。

2.2 加入鉴别码

为了保证 ELF 文件的完整性，在文件的末尾增加了消息鉴别码。系统中没有采用常用的数据库存放鉴别码，因为数据库的实现方式引入了管理开销，要求管理员管理执行文件和签名，同时每次增加新的可执行程序或对其修改时，需要更新库，并且当库表变化后需要重新进行完整性计算。

ELF 文件中加入鉴别码的方法：在文件末尾加入鉴别码，其格式如图 3 所示。

图 3 鉴别码格式

主要字段描述如下：

(1)长度：标识附加数据的长度（即图中阴影部分的长度），按字节算。

(2)ELF 源文件长度：记录计算验证码前 ELF 文件的长度。

(3)鉴别码长度：可采用不同的算法计算，生成的鉴别码长度不一样。

(4)算法标识别：由于 MAC 可采用多种迭代散列函数计算，因此需要算法标识字段来识别。例如 MD5、SHA1 就是这种散列函数。

(5)鉴别码：采用 MAC 和密钥计算 ELF 源文件生成唯一识别码。

鉴别码计算的过程中只涉及 ELF 源文件，不包含扩展部分，并且对源文件内容不做任何修改。这样保证当验证系统没有启动时，ELF 文件仍可正确执行。

3 设计与实现

3.1 设计要求

利用 MAC 保证 ELF 的安全性应满足以下要求：

(1)完整性值的计算完全在内核中执行，通过新增系统调用来实现。在计算完整性值时保证文件是安全的，没有被篡改或伪造。

(2)完整性值的验证也在内核中执行，密钥保存在内核中。

(3)在加载进程映像时验证 ELF 可执行文件的完整性。

(4)在装载 ELF 共享目标文件（即动态库）时验证其完整性。

(5)在加载内核模块时验证其完整性。

(6)合法软件（由管理员识别）可以安装执行。

(7)操作系统的运行不受鉴别码的影响（即增加鉴别码对操作系统性能的影响不大，并且在验证系统没有工作时不影响操作系统的正常运行）。

3.2 实现结构

(1)实现结构

由图 4 可知，系统包含两个部分：运行于内核空间的可加载内核模块和运行于用户空间的管理工具，各部分完成的主要功能如下：

1)可加载内核模块负责劫持系统调用、计算文件的完整性、验证文件的完整性。

2)用户空间的管理工具在安装时负责对干净的操作系统扫描，生成鉴别码；在运行过程中由管理员执行，以收集形成新软件的鉴别码。

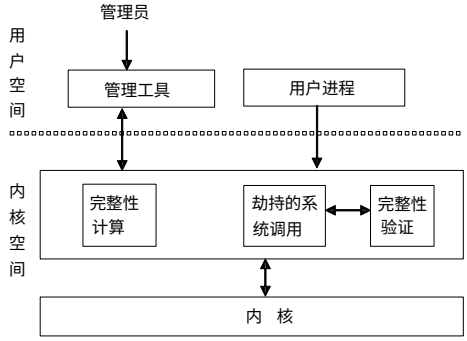

图 4 Linux 安全增强系统体系结构

由于系统采用内核模块的实现方法，存在加载时机的问题。因此要求系统是在安全的状态下启动，确保在内核模块加载之前所有的进程、模块都是安全的。

(2)系统调用劫持

系统调用是用户进程访问内核代码的接口。由于用户进程与内核代码位于不同的地址空间，不能直接访问，只能通过系统调用获得内核提供的功能，这增强了系统的安全性、稳定性与可靠性。用户进程获得内核功能通过软中断 int $0x80 来实现，操作系统经由 IDT（中断描述符表）表找到 0x80 处的中断门，进而找到中断处理程序的入口地址 system_call，并执行该处理程序。根据系统调用号在系统调用表（sys_call_table）中找到系统调用处理函数的入口地址，用 call 指令调用该处理函数。最后将结果返回给用户进程。

系统调用表（sys_call_table）实际上是个数组，保存系统调用服务例程的地址。操作系统利用系统调用号完成对系

统服务处理程序的检索和调用。常用的系统调用劫持就是通过修改系统调用表中系统调用服务程序的地址也来完成的。

在 Linux2.4.18 内核以后，为了增强系统的安全性，sys_call_table 没有导出，不能直接使用。如何定位 sys_call_table 的地址成为实现的一个关键问题。文献[7]提出了一种在没有 LKM 支持和 System.map 文件的情况下定位 sys_call_table 的方法，是通过/dev/kmem 文件（该文件是一个字符设备文件，实际上是虚拟内存的映像，root 用户可读写该文件，用于测试或修改系统）实现的。由于本文提出的方法是基于 LKM 的，因此可以在内核空间中直接查找 call 指令对应的指令串而得到 sys_call_table 的地址。

(3) ELF 文件加载/验证过程

1)可执行文件的加载/验证过程

Linux 系统中 ELF 可执行文件的加载是通过系统调用 sys_execve 来实现的。内核首先确定可执行文件的格式，然后执行对应的加载函数。对于 ELF 格式的文件，其加载函数为 load_elf_binary。该函数执行过程中会首先查询共享库解释器，并在其后进行加载，因此对于共享库的解释器，也需要验证。

我们劫持了该系统调用以验证执行的程序及共享库解释器的完整性。完整性验证的过程中读入源文件的内容，与密钥一起输入 MAC API 函数，将得到的结果与文件中承载的结果比较。若验证成功，可继续加载进程；若验证失败，则终止进程加载，返回错。

2)共享库的加载/验证过程

Linux 系统中共享库的加载是通过函数 dlopen()来实现的，使用 strace 跟踪发现实际上使用函数 open()打开文件，然后通过系统调用 old_mmap 将共享库加载到进程地址空间。因此在内核中我们劫持了系统调用 sys_open，并在其中验证共享库的完整性。验证方法同可执行文件。

3)内核模块的加载/验证过程

内核模块实际上是一种可重定位文件，Linux 支持 LKM（Loadable Kernel Module）技术提供对内核的动态扩充，一旦 LKM 被加载进内核，就成为内核代码的一部分，与其它内核代码地位等同。编写一个 LKM 程序，用编译器将其编译为一个可重定位的 ELF 文件，就可以根据需要动态加载，在不需要其所提供的功能时卸载它。目前 Linux 中大部分的设备驱动程序、文件系统、网络协议等均设计成内核模块。

root 用户可以使用 insmod 和 rmmod 命令明确地加载和卸载一个模块，也可以在需要时由内核自己请求守护进程加载和卸载该模块。

由于 LKM 模块进入内核后，就可修改内核的关键数据结构和变量，完成诸如文件、进程隐藏，预留后门等危险活动，因此确保内核模块的完整性是非常重要的。

LKM 模块的加载经过了一系列活动，在内核中对应的系统调用为 sys_create_module、sys_init_module。通过劫持这些系统调用，可验证加载模块的完整性。这两个系统调用只有模块名，没有模块对应的路径名。由于模块加载时需要打开模块文件，因此可从当前进程的打开文件列表中查询所有的文件描述符，根据文件描述符找到对应的完整路径，若包含系统调用提供的模块名，则为对应的模块路径。验证方法同可执行文件。

3.3 存在的问题及解决办法

(1)密钥的安全性

通过前面的分析可见，整个安全性的基础完全基于 MAC 算法的密钥。若密钥泄露，黑客可随意伪造、篡改 ELF 文件和鉴别码，验证系统不能检测。因此如何保证密钥的安全性，是确保 ELF 文件安全执行的基础。我们要求密钥存放在内核中，在模块加载时根据硬件参数（如硬盘序列号及其它参数）动态生成。在模块加载后，模块文件不可跟踪。

在高安全性的场合，作者建议采用加密卡执行完整性的计算和验证，密钥和算法存放在加密卡中，不会导出到内存，用户无法跟踪获得密钥信息。

(2)性能改进

系统设计的一个要求是在增加了 MAC 计算之后不应该对正常的执行产生过多的影响，这也是没有采用 RSA 签名验证的原因。为了进一步提高性能，在内核中提供了 Cache 表，用于存放最近验证过的文件。当验证请求到达时，将首先查找 Cache 表，若在其中找到对应的条目，则直接返回，可继续加载；若没有找到相应条目，则进行 MAC 计算，若计算的结果与文件存放的记录匹配时，可继续加载，同时记录到 Cache 表中，否则，终止程序的执行。实验结果表明，引入 Cache 可大大提高系统性能。

(3)软件安装

系统中另一个关键问题就是如何安装新软件。由于 Linux 开放源码的特点，新软件自身的安全性只能由管理员从合法的站点下载来保证。新软件安装完毕后由管理员启动管理工具，生成鉴别码。为了保证鉴别码生成的安全性，我们新增了一个系统调用 sys_auth，管理工具调用该系统调用，将在内核空间中计算鉴别码。由于在软件安装的过程中可能需要执行临时文件，如何保证临时文件的执行是存在的一个主要问题，有待于进一步的研究。目前考虑的的解决办法为：安装新软件时开启内核模块的安装开关，此时若被创建进程的父进程为安装进程（如 rpm），则允许执行，不进行完整性检测，安装结束后关闭安装开关。

4 实验结果

表 1 是一组简单的测试数据，这组数据是多次测试后取得的平均值。考虑了 3 种情况：即无验证，验证及带 cache 的验证。测试环境为：CPU Pentium IV 2.4GHz，RAM 512MB，RedHat 9，Kernel linux-2.4.20-8，虚拟机 VMware Workstation，Version:4.0.5 build-6030，配置 RAM 160MB。

表 1 实验结果 （单位：μs）

程序	无验证	验证	带 cache 验证
ls/	9 756	111 342	10 290
gcc-o	213 700	1 092 772	215 725
rpm-ivh	269 124	519 174	278 777

从表中可以看出，在不带 cache 的情况下，由于每次加载的可执行文件及动态库都需要验证，因此验证开销比较大；在带 cache 的情况下，验证开销增加不到 5%，大大降低了对系统性能的影响。

5 结束语

本文提出了一种采用 MAC 保证 ELF 文件安全性的机制。该方法通过在内核中劫持系统调用，验证 ELF 文件的完整性，可防止伪造、篡改、非授权软件的执行，可提升 Linux 操作系统的安全性。通过分析及测试结果表明，该方法安全可行。本文提供的方法是对 ELF 文件静态特征进行验证，对于系统运行过程中产生的各种攻击行为（如缓冲区溢出等）没有防护能力，这是下一步研究的重点。 （下转第 156 页）