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—0162—02 文献标识码：A 中图分类号：TP309

一种基于用户的Capabilities安全模型及其实现

许 平[1]，陆松年[2]，杨树堂[2]
(1. 上海交通大学信息安全工程学院，上海 200240；2. 上海交通大学电子工程系，上海 200030)

摘 要：分析了Linux下Capabilities安全模型的不完善性，并提出了一种基于用户的Capabilities安全模型，给出了在Linux下的实现方式。同时，还分析了如何利用改进后的安全模型来增强Linux的安全性。

关键词：Capabilities；Linux；访问控制

A User-based Security Model of POSIX Capabilities and Its Realization

XU Ping[1], LU Songnian[2], YANG Shutang[2]
(1. School of Information Security Engineering, Shanghai Jiaotong University, Shanghai 200240;
2. Department of Electronic Engineering, Shanghai Jiaotong University, Shanghai 200030)

【Abstract】 This paper analyzes the disadvantage of POSIX capabilities on Linux, which also puts forward a user-based model of POSIX capabilities and its realization on Linux. And the enhancement of Linux security with the new model is analyzed as well.

【Key words】 Capabilities; Linux; Access control

Linux凭借着开放免费的代码、稳定可靠的性能以及适应多种硬件平台的特性正获得越来越多的应用。与此同时，Linux的安全性能也面临着越来越大的挑战。

在Linux中，一个重要的安全隐患来自于属主为root的setuid程序。由于Linux中root用户拥有系统所有的特权而普通用户没有任何特权，因此经常采用setuid机制来帮助普通用户完成某些特权操作。通过setuid机制，用户在执行程序时能暂时获得与属主相同的权限。例如，一个普通用户需要使用ping命令。这是一个SUID命令，会以其属主也就是root的权限运行。如果程序存在安全漏洞，就可能被攻击者利用，获得系统的控制权。而实际上这个程序只是需要RAW套接字建立必要ICMP数据包，除此之外的root权限对这个程序都是没有必要的。setuid机制安全问题的根本原因在于Linux对特权的分配，root用户拥有所有特权，而普通用户则没有任何特权，这样的做法对系统安全是不利的。为了解决以上的安全问题，POSIX1e中提出了Capabilities（本文中译为权能）安全模型。

1 Capabilities安全模型简介

Capabilities安全模型的基本思想是分割root用户的特权，从而避免权限过分集中导致的安全隐患。该模型将root用户的特权分割成多种权能，只有拥有特定权能的进程，才能完成相对应的特权操作。

在该模型中，只有进程和可执行文件拥有权能。进程拥有3个权能参数，分别是cap_effective(简称pE)、cap_inheritable(简称pI)和cap_permited(简称pP)。pE表示进程当前拥有的权能，pI表示进程可以传递给子进程的权能，pP表示进程能拥有的最大权能。其中，pE决定了该进程有权完成哪些操作。

同样，可执行文件也有3个权能参数，分别是cap_effective（简称fE）、cap_allowed（简称fA）和cap_forced（简称fF）。fE表示可执行文件运行时可以使用的权能，fA表示可执行文件运行时可以从pI中继承的权能，fF表示运行文件时必须拥有才能完成的权能。

能否进行特权操作由对应进程的权能决定，所以进程的权能是该模型的关键。每个进程的权能由其父进程和与其相关的可执行文件的权能共同决定，具体的计算公式如下：

pI’ = pI
pP’ = fF | (fA& pI)
pE’ = pP’ & fE

其中，pI’、pP’和pE’是新进程的权能参数，fA、fF和fE是其对应可执行文件的权能参数，pI、pP和pE是父进程的权能参数。

2 Capabilities模型在Linux中的实现

Linux内核从2.2版开始支持Capabilities模型，并随着版本的提高不断完善。发展到2.4以及最新的2.6版中，Linux内核将root用户的特权分割为29种权能。内核用一个32位无符号长整数的低29位表示这29种权能。

进程的3个权能参数就以这种形式存储在内核内存的task_struct结构中，分别名为cap_effective、cap_inheritable和cap_permited。此外Linux中还引入了控制位keep_capabilities，用于控制进程的用户身份由root变为非root时，是否保留其权能参数不变。

除此之外，Linux中还引入了权能边界集（capability

基金项目：国家“863”计划基金资助重大项目（2005AA145110，2002AA145090）
作者简介：许 平(1981－)，男，硕士生，主研方向：嵌入式Linux；陆松年，教授；杨树堂，副教授
收稿日期：2005-12-06 **E-mail：**sixxu@sjtu.edu.cn

bound set）的概念，用来表示所有进程可能拥有的最大权能。Linux 用全局变量 cap_bset 来表示权能边界集，将它的某一位置 0，则所有进程都无法获得对应的权能。具体的实现方法是：对于 init 进程，其权能参数的设置如下：

```
pI = 0;
pE = pP = cap_bset;
keep_capabilities = 0;
```

同时修改 pP 的计算公式如下：

```
pP' = (cap_bset & fF) | (fA & pI)
```

由于所有进程都是 init 进程和其子进程，经过上述的修改后，所有进程的 pE 都不可能大于 cap_bset，从而提高了系统的安全性。比如，Linux 默认屏蔽权能 CAP-SETPCAP，即所有进程都不能修改进程的权能。

系统调用 fork()或 clone()生成子进程时，子进程完全复制父进程的权能参数，只有当进程调用 exec()时，子进程的权能参数才根据父进程和其调用的可执行文件重新确定。然而在 Linux 中，可执行文件的权能模型并没有真正实现。系统仅仅是通过进程用户身份来对可执行文件的 3 个权能参数临时赋值：对于 root 用户进程，可执行文件的权能参数赋最大值；对于非 root 用户进程，可执行文件的权能参数赋 0 值。这就造成了 Linux 下的 Capabilities 模型并没有从根本上达到分割 root 用户权限的目的，root 用户仍然拥有了 cap_bset 所规定的所有特权。

3 对 Linux 下 Capabilities 模型的改进

从上述的分析可以看出，Linux 下的 Capabilities 模型并没有成功分割 root 用户特权，只能从系统中去除一些不安全的特权。造成这种情况的根本原因是没有基于用户的权能模型。在 Capabilities 模型下，只有 root 用户和非 root 用户的区别，分割 root 用户特权当然无从谈起。为了解决这个问题，下文将提出一种基于用户及用户组的 Capabilities 模型及其在 Linux 下的实现方案。

3.1 基于用户及用户组的 Capabilities 模型

为了实现基于用户及用户组的 Capabilities 模型，需要为每个用户引入 2 个权能参数 user_permited(简称 uP)和 group_permited(简称 gP)，分别代表该用户及所在用户组所拥有的权能。显然，用户所有的权能不能大于其所在用户组的权能，所以当用户权能参数发生变化时，计算公式如下：

```
if（!gp'）      //没有新的参数 gP'
  gP' = gP;
if（!uP'）      //没有新的参数 uP'
  uP' = uP;
uP' = uP' & gP';
```

上述公式中，gP'、uP'代表新的用户权能参数，gP、uP 代表旧的用户权能参数，计算公式就保证了 uP 始终不大于 gP。这样，可以通过修改 gP 来影响组内所有用户的 uP，方便管理。

引入权能参数 uP 和 gP 之后，还需要修改原来的进程权能计算公式：

```
pP' = (fF | (fA & pI)) & uP
```

这样就保证了某个用户的进程不会拥有超过用户本身的权能。通过引入新定义的权能参数 uP 和 gP，root 用户的特权要受到 uP 的约束，特定的非 root 用户也可以通过 uP 获得更多的特权，从而达到了分割 root 用户特权的目的。

3.2 在 Linux 下的实现方案

为了在 Linux 下实现基于用户及用户组的 Capabilities 模型，需要实现用户权能信息的静态存储。系统在完成初始化之后、运行第 1 个用户程序之前，将用户权能信息从文件读入到内核内存之中。这样的做法会占用一部分内核内存，降低系统性能，但是由于创建进程需要用户的权能信息，每次从磁盘上读取的速度会更慢。由于系统中可能存在大量用户权能参数相同的用户，可以将这些相同的参数设定为一个缺省值写入权能信息文件。使用缺省值的用户权能信息不必存入权能信息文件，也不会被读入内存，当需要该信息但查找不到时，使用缺省值即可。这样就大大减少对内存的占用。

3.2.1 存储结构

当进程建立完毕，调用 exec()时，系统在内核内存中查找出对应的用户的权能信息，并根据父进程、可执行文件以及用户的权能信息综合计算出新进程所拥有的权能。其中，用于存放用户权能信息的数据结构如下：

```
struct list_head{
  struct list_head *next, *prev;    };
struct user_cap{
  struct list_head user_cap_list;
  uid_t uid;
  gid_t gid;
  kernel_cap_t user_permited, group_permited;    };
```

user_cap_list 是一个双向链表结构，通过遍历这个链表可以查找出相应的权能信息。uid、gid 分别是用户 id 和用户组 id，user_permited、group_permited 分别是 uP 和 gP。

3.2.2 调整权能计算公式

由于 Linux 引入权能边界集的概念，改进后的权能计算公式相应地变为

```
pP' = ((cap_bset & fF) | (fA & pI)) & uP
```

对于 root 用户进程来说，上述公式完全能正常工作；然而对于非 root 用户进程，由于 fF 和 fA 被赋 0 值，pP'=0，仍然需要诸如 setuid 的机制来获得特权。因此需要做出修改，将 fF 和 fA 始终赋值 cap_bset，方能保证正常工作。

可以看到，经过修改后，进程的权能实际上与可执行文件的权能参数无关。这点并不奇怪，因为基于可执行文件的权能模型在 Linux 下并没有真正实现，它本质上是个粗略的基于用户的机制。现在有了较完善的基于用户的机制，原来粗略的机制也就没必要起作用了。当然，在公式中仍然保留了接口，如果 Linux 真正实现了基于可执行文件的权能模型，上述公式仍然可以正常使用。

3.2.3 权能信息文件的保护

由于权能信息文件包含了所有用户的权能信息，因此如何防止该文件被非法用户篡改是加强 Linux 系统安全性的关键。Linux 系统中的文件都有 S-IMMUTABLE（不可修改）属性标志，用来禁止对该文件的任何修改操作。如果用户具有权能 CAP-LINUX-IMMUTABLE，就能够使用系统调用 ioctl()来修改文件的该属性标志。所以，可以建立拥有权能 CAP-LINUX-IMMUTABLE 的用户，同时取消其他所有用户的该权限。该用户通过有口令设置的专门程序来管理权能信息文件，每次完成修改后即置 S-IMMUTABLE 标志，从而保证权能信息文件的安全。

3.3 利用改进的 Capabilities 模型增强系统安全

setuid 机制安全问题的根本原因在于 root 用户拥有所有特权，而普通用户又没有任何特权。改进后的 Capabilities 模型可以从这两方面解决这个问题。一方面，新模型可以分割

（下转第 166 页）