一个受控特权框架的设计与实现*

梁 彬+, 孙玉芳, 石文昌, 孙 波

(中国科学院 软件研究所,北京 100080)
(信息安全国家重点实验室,北京 100080)

Design and Implementation of a Controlled Privilege Framework

LIANG Bin+, SUN Yu-Fang, SHI Wen-Chang, SUN Bo

(Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)
(State Key Laboratory of Information Security, Beijing 100080, China)
+ Corresponding author: Phn: +86-10-62555043 ext 19, E-mail: liangbin01@ios.cn, http://www.sonata.iscas.ac.cn

Received 2003-07-25; Accepted 2004-05-08

Liang B, Sun YF, Shi WC, Sun B. Design and implementation of a controlled privilege framework. *Journal of Software*, 2004,15(Suppl.):74~82.

Abstract: In order to support to the principle of least privilege effectively, considering the limitations of traditional privilege mechanisms, a new Linux privilege mechanism called controlled privilege framework (CPF) is proposed. CPF provides a fine-granularity partition of system privileges; improves the privilege computing mechanism of privileged process; and introduces the notation of privilege state for privilege control, refines the unit of privilege control farther. Based on CPF, fine-granularity and automatic privilege control can be performed totally transparent to all applications. The experimental results show that the threats of intrusion are reduced and effective support to the principle of least privilege can be achieved.

Key words: privilege; privileged process; capability; principle of least privilege; Linux

摘 要: 为了有效地支持最小特权原则,针对传统特权机制存在的局限性,提出了一个新的 Linux 特权机制受控特权框架(CPF).该框架提供了细粒度的特权划分;完善了特权进程的特权计算机制;引入了特权状态的概念来进行特权控制,进一步细化了特权控制的单位.基于此框架,可以对特权进程进行细粒度的,自动的特权控制,并对应用完全透明.试验表明其能大大降低入侵事件的危害性,有效地支持了最小特权原则.

关键词: 特权;特权进程;权能;最小特权原则;Linux

---

* Supported by the National High-Tech Research and Development Plan of China under Grant No.2002AA141080 (国家高技术研究发展计划(863)); the National Natural Science Foundation of China under Grant Nos.60073022, 60373054 (国家自然科学基金); the Knowledge Innovation Engineering Program of the Chinese Academy of Sciences under Grant No.KGCX1-09 (中国科学院知识创新工程)

作者简介: 梁彬(1973－),男,云南昆明人,博士生,主要研究领域为系统软件与信息安全;孙玉芳(1947－),男,研究员,博士生导师,主要研究领域为系统软件和中文信息处理;石文昌(1964－),男,博士,研究员,主要研究领域为系统软件与信息安全;孙波(1975－),男,博士研究生,主要研究领域为系统软件与信息安全.

© 中国科学院软件研究所 http://www.jos.org.cn

在操作系统中,特权(privilege)是进程执行一些安全相关操作所必须具备的权限.这些操作涉及到系统安全的各个方面,例如在Linux操作系统中,mount/umount文件系统,绑定1024以下的端口等许多操作都需要进程具有相应特权.对特权的滥用将导致非常严重的后果,特权的获取往往也是入侵者或恶意代码的首要目标.特权的控制和管理关系到整个系统的安全.通常,这些具有特权的进程在安全操作系统中属于可信进程(trusted process)的范围,特权机制为可信进程机制的最重要的组成部分,而可信进程又是可信计算基(trusted computing base, TCB)的一部分.特权的控制和管理是研发安全操作系统不可回避的主要问题之一.

已有系统的特权机制的实现方式差异很大,体现在特权的界定,特权的属主,进程特权的计算方法,特权变化和对最小特权原则的支持等多个方面.这些实现机制都存在着局限性,主要有:特权的属主局限于单一类型的实体,特权划分的粒度较粗,缺乏对最小特权原则的充分支持,缺乏对特权在进程生命周期中变化的完善的控制和与主流操作系统兼容性较差造成的应用障碍等.作者在研发一个基于Linux的满足GB 17859-1999第4级要求的安全操作系统的项目过程中,针对已有系统特权机制的局限性,提出了一个新的Linux特权机制框架——受控特权框架(controlled privilege framework, CPF).在对系统调用进行安全性分析的基础上,CPF给出了一个细粒度的特权划分,并且与原有特权划分向下兼容;将Linux传统特权机制的支持进行了扩展,使得程序成为了特权的属主,改变了传统机制中特权的计算完全依赖于对进程用户ID是否为超级用户root进行检查的状况;引入了进程特权状态(privilege state)这一新的概念,大大细化了最小特权原则的支持粒度,并且进程特权的变化完全按照安全策略的规定由TCB自动加以配置.作为整个安全操作系统项目的一部分,作者选用了已经被Linux新核心采用的LSM(linux security modules)框架作为CPF实施基础,以内核可加载模块的形式在内核2.5.72版和2.6.4版上实现了CPF.

本文第1节对特权机制和已有实现进行了分析,指出了现有技术的局限性.第2节提出了CPF框架,介绍了其设计思想和总体结构.第3节对CPF的实现情况进行了描述.第4节介绍分析了相关工作.最后是全文的总结.

1 问题分析

特权的管理和控制最重要的目标之一是支持最小特权原则(principle of least privilege),最小特权原则是系统安全中最基本的原则之一,所谓最小特权,指的是"系统不赋予主体(用户或进程)完成当前工作所必不可少的特权之外的特权."这使得由可能的事故,错误和实体被篡改等原因造成的损失降到最低.实施最小特权原则也是降低缓冲区溢出等入侵方法的危害的有效手段.以下我们对Linux特权机制进行分析,指出其存在的局限性,并探讨相应的解决思路.

Linux继承了传统Unix的特权管理机制,即基于超级用户的特权管理机制.设立一个root超级用户,用户ID为root的进程不受系统访问控制规则的约束.在特权划分方面,Linux参照POSIX.1e标准[1]将特权划分为29个权能(capability),其中7个为POSIX.1e标准所规定的权能,其余22个为Linux特有的权能(其中权能CAP_SETPCAP实际上被废置不用).用户有效UID(euid)和文件系统UID(fsuid)为0(root)的进程拥有除CAP_SETPCAP外的所有权能,而普通用户进程不具备任何权能,特权机制实际上完全依赖于进程的用户ID属性,与进程的逻辑完全无关,任何用户UID为0的进程都潜在地能够进行任何特权操作.

Linux这种简单的特权机制虽然效率较高,但给系统带来了较大的安全隐患,存在的主要问题有:程序文件并不是特权的属主,进程的特权与程序的逻辑无关,导致无法进行任何与程序逻辑相关的特权最小化处理,所有root进程所具有的特权都超出了其实际需要;其次,所有特权完全集中于root用户,一旦入侵者取得root身份就能获得对系统的完全控制,其危害性是不言而喻的,而且,普通用户进程若要使用特权或者root进程要放弃特权只有通过改变进程的用户ID属性进行,而Linux(包括Unix和其它类Unix操作系统)相关的set*uid/set*gid系统调用微妙的逻辑常常造成对进程用户ID属性不适当的设置,带来了潜在的安全隐患[2];此外,Linux特权的划分也较粗,有的权能涵盖范围过大,如权能CAP_ADMIN,其涵盖了设置安全注意键(security attention key, SAK),网络文件系统管理,设备管理等数十种特权操作,不利于实施最小特权原则.

针对Linux特权机制存在的问题,在设计实现一个完善的Linux特权机制时,应保证目标系统满足以下要求:

(1) 细粒度的特权划分,并且与原有特权划分向下兼容;

© 中国科学院软件研究所 http://www.jos.org.cn

(2) 用户与程序文件都应成为特权的属主,进程的特权应与对应程序的逻辑相关;

(3) 支持进程特权的动态变化,并且提供由 TCB 强制实施的特权变化机制,以支持最小特权原则;

(4) 具有方便高效的,表达能力较强的特权规则(策略)说明机制;

(5) 特权机制对原有应用程序透明,确保与原系统的兼容.

2 受控特权框架 CPF

在本节中,作者提出一个新的 Linux 特权框架——CPF.所采用的基本技术是截获 Linux 系统调用,由一个引用监控机制进行检查

2.1 基本原理

CPF 中引入进程的特权状态这一概念来对进程的特权行为进行分块控制,即将进程整个生命周期根据其完成操作所需的特权划分为几个部分,分别用特权状态标识.在各个特权状态中,进程具有不同的特权,当进程特权状态发生转换时,其特权自动随之变化.这样作既提供了适当的控制粒度又不致以引入过分复杂的控制和配置结构.特权状态的引入大大提高了系统对最小特权原则的支持. 可用状态机模型来描述进程的特权变化.进程的整个运行过程可以看作为一个状态机,每个状态由进程特权相关属性组成,对导致进程特权相关属性变化的系统调用的调用构成了状态转换.再不同的状态下,进程可能具有不同的特权.系统管理员可根据程序逻辑指定程序的在各个特权状态下完成操作所需要的特权,使得程序也成为特权的属主.当程序所对应的进程在运行过程中发生特权状态转换时,系统根据进程的特权配置,进程用户和新的进程特权状态为进程重新计算特权,把对最小特权原则的支持粒度细化到了进程状态一级.

2.2 LSM简介

CPF 选用了 LSM 框架[3]作为实施的基础. LSM 是一个为 Linux 内核提供的轻量级的,面向通用的访问控制框架,使得不同的访问控制模型能实现为可动态装载的内核模块.LSM 在内核中资源访问控制点上插入了一百多个钩子(hook),并规定了安全模块和钩子函数的接口标准.各个安全项目按照统一的标准通过实现其自己的钩子函数来完成对内核资源访问操作的控制.LSM 已经被 Linux 官方内核所采用,最新版的 Linux 标准内核(2.6)也已加入了 LSM. SELinux, DTE, OWL 和 LIDS 等安全项目已经被移植到了 LSM 上,内核中原有的 Capability 特权机制也基于 LSM 实现为可动态装载的内核模块.

2.3 特权划分

设计一个完善的特权机制的首要任务是进行细粒度的特权划分.特权划分的基础是特权的界定,因为系统调用是应用程序访问资源的唯一途径,在对 Linux 系统调用安全相关性分析的基础上进行特权的界定和划分是十分自然的.作者所采用的系统调用安全性分析方法与 REMUS[4]类似,根据与系统安全的关系将系统调用分为影响整个系统安全的(通过其能获得系统的完全控制或者使得系统出现拒绝服务 DoS(denial of service)错误),影响当前调用进程安全的和无害的 3 类.在 CPF 中,对第 1 类系统调用(见表 1)进行控制,即只有相应 CPF 特权才能进行调用或者使用特别的调用参数进行调用.

此外,在为 Linux 这样的主流操作系统设计特权机制,必须保证新设计的特权机制与原有机制和应用的兼容,这样才能保证不修改内核代码及机制对应用程序的透明.CPF 根据以上分析结果在 Linux 权能设置的基础上进行了特权划分.在 CPF 中设置了 70 个特权,大致可以分为以下 9 类: Linux 原有权能的细化、等价于 Linux 原有权能、调用特权系统调用(例如_sysctl)、安全策略配置文件访问特权、系统文件访问特权、CPF 机制自身控制特权、审计特权、强制访问控制特权(下写特权和安全标签设置特权等)及危险环境下调用敏感系统调用的特权等.

© 中国科学院软件研究所 http://www.jos.org.cn

表 1 受控系统调用

System calls	Threat
open link unlink mknod chmod lchown16 mount rename chroot symlink fchmod fchown16 chown16 lchown fchown chown init_module execve setuid16 setgid16 setreuid16 setregid16 setgroups16 setfsuid16 setfsgid16 setresuid16 setresgid16 setreuid setregid setgroups setresuid setresgid setuid setgid setfsuid setfsgid capset	Full control
oldumount umount truncate ftruncate swapon swapoff quotactl flock nfsservctl truncate64 ftruncate64 ipc mlock mlockall vm86 delete_module fork ptrace setrlimit setpriority ioperm iopl vhangup vm86old clone modify_ldt sched_setparam sched_setscheduler vfork kill tkill socketcall reboot _sysctl stime sethostname settimeofday setdomainname adjtimex syslog	DoS

CPF 的特权设置具有以下特点:

(1) 细粒度的特权划分.例如,将 Capability 机制中 CAP_SYS_ADMIN 等权能拆分为多个 CPF 特权;

(2) 引入特权参数这一概念.进程要进行特权操作不仅要具有相应的特权,其操作参数还必须满足特权策略所规定的参数限制.原有的特权机制中特权的含义是固定的,即使是与特权操作的参数相关,其关系也是隐式的和固定的.CPF 在部分 CPF 特权引入了特权参数,特权与特权操作的参数之间的关系是显式的,并可根据具体的特权进程进行配置,大大扩展了特权机制控制的灵活性和准确度.这样作还在一定程度上保证了特权操作的正确性(限定了特权操作的相关参数).目前带有参数的 CPF 特权操作有安装文件系统,卸载文件系统,安全策略文件操作,系统文件操作,强制访问控制超越操作等等.以安装文件系统操作(CPF_SYS_ADMIN_MOUNT)为例,其参数为待安装的设备和目标目录等;

(3) 将在危险环境中进行敏感系统调用也作为特权统一处理.有的进程在运行过程中必须和外部环境不断地进行交互,例如 http 和 ftp 服务器进程等,它们往往是入侵者进入系统的渠道.当入侵者通过缓冲区溢出等手段获得了系统的控制权之后,也必须通过一些系统调用才能达到其非法的目的.在这些进程中对一些敏感系统调用往往是安全问题的源头.在 CPF 中,可以通过标识这类程序应该控制的敏感系统调用,并通过特权机制进行调用检查,即使其被入侵所控制,其行为也仍然被控制在合法的范围.目前进行控制的敏感系统调用有 execve, set*uid, set*gid, setgroups, kill, open 等,对应 CPF 特权的编号从 96 开始,都为带参特权.需要特权强调的是此类特权是局部性,即调用同一系统调用对大多数程序而言是平常的操作,而对于有的程序必须作为特权操作加以控制,例如在 http 服务器进程中调用 execve 系统调用就必须严加控制.

(4) CPF 特权设置完全包含了 Capability 权能,为其超集.

2.4 特权状态

对于特权状态的划分准则的选择,考虑到进程特权属性依赖于用户特权属性和程序特权属性,进程的用户或组标识的改变往往意味着进程访问权限的改变.此外,在已有进程属性标识中,进程用户标识(*uid,*gid 等)一直被用来标识用户当前操作所代表的用户,操作系统原有的访问控制也主要以此为基础.由于我们面对的应用程序数量十分巨大,在保证向下兼容的需求下完全抛弃用户标识而完全由其它进程属性标识进程特权状态并不适当.故在 CPF 的原型系统开发中考虑首先使用进程的*uid,*gid 来标识进程的特权状态,即进程不同的*uid,*gid 标识描述了进程处于不同的特权状态下,进而可能具有不同的特权.进程对 set*id(setuid, seteuid, setreuid, setresuid, setgid, setegid, setregid, setresgid, setfsuid 和 setfsgid 等)的调用事件导致了特权状态的转换.

由于 Linux 进程用户和组 ID 属性在运行过程中的变化可能会出现反复,即相同用户和组 ID 属性的状态会多次出现.虽然这些状态的用户和组 ID 属性相同,但所需特权可能由于进程的逻辑不同而不同.为此,CPF 系统对每个特权状态进行了编号,对于用户和组 ID 集相同的特权状态可用状态编号进一步划分.由于编号的引入,多个不同特权状态的用户和组 ID 集可能相同.为了保证特权状态转换的正确,每个特权状态还具有一个可转换目标状态编号集.在进行特权状态转换时,除了匹配用户和组 ID,还要根据状态编号检查目标特权状态是否在当前状态可转换到的状态集中.

需要特别指出的是,这种特权变化由控制机制自动进行.维护进程的特权是一个非常危险的操作,这种特权

© 中国科学院软件研究所 http://www.jos.org.cn

的自动调整将大大降低这种危险.而且这种机制对已有应用程序透明,不必依赖对已有应用程序的修改就可以在较细的粒度支持最小特权原则.

此外,在这种机制下的进程特权计算完全基于进程当前的 ID 属性与特权策略的匹配结果,所有 ID 属性都参与了匹配,消除了对进程用户 ID 属性的可能的不适当的设置所造成的安全隐患.这些不适当的设置往往是由于 Linux set*uid/set*gid 系统调用微妙的逻辑所造成的,所造成的安全案例并不罕见[2].

2.5 特权属性

用户,程序文件和进程三种实体拥有特权属性,但它们的特权属性并不完全相同:

(1) 用户的特权属性不包括调用敏感系统调用的特权(96 以后),并且其包括的特权不带有参数;

(2) 程序的特权属性描述了与此程序对应的进程在各种特权状态下所具有的特权(包括全部特权),若有必要还包括特权参数.此外,程序的特权属性还包括对其的调用应该加以控制的敏感系统调用,即标识在此程序的运行环境中对于这些敏感系统调用来说可能为危险操作,因为这些性质与进程的逻辑和其运行环境密切相关,不能在用户的特权属性中得到准确的描述;

(3) 系统中还设置了一个全局特权属性,提供对系统特权机制的全局控制;

(4) 进程的特权由进程当前特权状态和程序及进程用户的特权属性决定.

2.6 框架体系结构

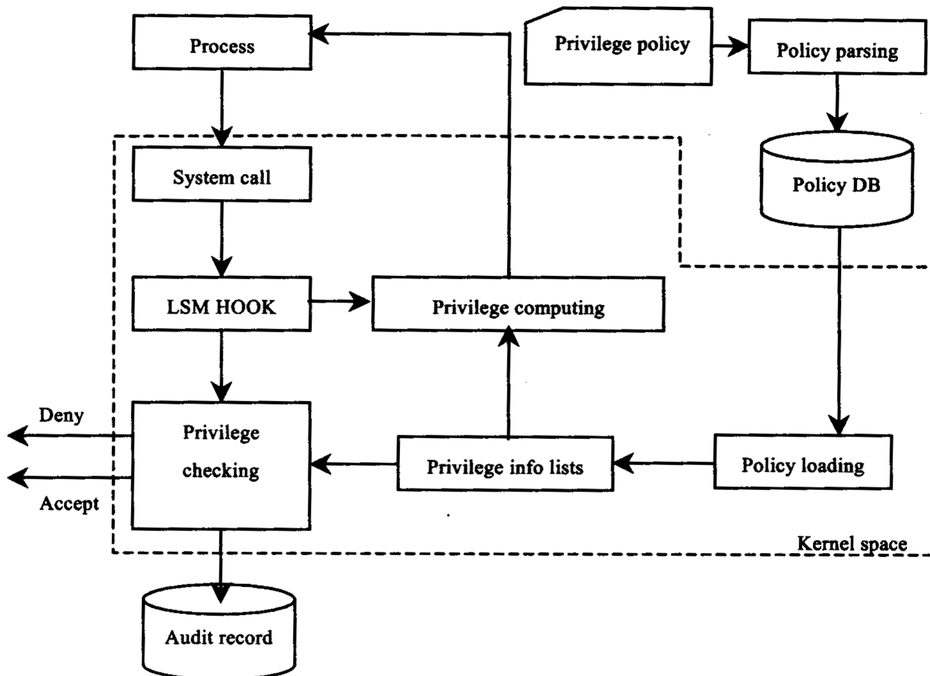

图 1 CPF 体系结构

CPF 框架体系结构如图 1 所示,其包括了以下几个主要部分:

(1) 特权策略说明和分析

对用户,程序特权属性的规定形成了特权策略.特权策略采用一种可读性很强的语言进行说明,存放在特权策略文件中,通过策略分析器分析后存放进一个二进制的特权数据库中.

(2) 策略装载

内核空间的策略装载器读取特权数据库中的特权信息,将其装入内核中的特权信息列表中,供特权检查机制进行特权判断和特权计算使用.

(3) 特权计算

© 中国科学院软件研究所 http://www.jos.org.cn

当有新的进程产生或进程的特权状态发生变化时,系统进行特权计算,设置进程的特权属性.

(4) 特权检查

当进程要进行特权操作时,通过LSM钩子截获相应的系统调用,由特权检查机制进行判断其是否具有进行此操作的特权.特权检查也是审计信息的重要来源之一,每次对特权操作请求的拒绝或接受都由特权检查机制写入审计记录中,供审计子系统使用.

3 CPF的实现

作者在已加入LSM机制的2.5.72版和2.6.4版Linux内核上以内核可装载模块的形式实现了CPF.

3.1 特权策略说明和分析

用户,程序文件和进程三种实体拥有特权属性,其中用户和程序的特权属性需要永久保存,在系统启动时读入内核.这些永久特权属性信息保存在磁盘上的配置文件中.由于CPF特权策略的配置较为复杂,作者设计了一门特权策略配置语言PSL(Privilege Specification Language)用于描述CPF特权策略.

特权策略包括对用户,程序的特权属性的说明,还包括对系统中的安全策略文件与系统文件的标识.其中用户的特权属性存放在文件/etc/cpf/user.conf中,描述说明了各个用户对特权编号为0~95的CPF特权的拥有情况.安全策略文件与系统文件的标识文件分别为/etc/cpf/secpolicy.conf 和/etc/cpf/sysfile.conf,其包含有相应文件或目录的路径,特权检查机制使用这些信息识别当前进程访问对象是否为一安全策略文件或系统文件.程序的特权属性存放在文件/etc/cpf/prog.conf中,针对各个特权程序描述说明了其在各个可能的特权状态下具有的特权,如果必要,还要说明特权参数和标识此程序的安全敏感系统调用.

文本格式的特权策略文件并不方便内核特权机制的读取,在策略配置完成后,可通过策略分析器(parser)将其转换为二进制的格式,存放在特权数据库中.这种策略说明和存放机制与系统所用文件系统无关,可适用于Linux支持的各类文件系统.

3.2 策略装载

当CPF模块被装载进内核时,通过一个内核态的策略装载器将特权数据库中的信息读入内核空间.由于必须在内核中直接读取位于磁盘文件上的特权数据库,作者编制了专门的内核态的文件读写函数,简化了读写过程,能高效安全地读写特权数据库文件.内核特权信息列表以有序链表的形式存放,访问频率较高的部分(例如安全策略文件和系统文件标识)采用了Hash表存放,以提高效率.

3.3 特权计算

在CPF中,使用4个无符号长整型数(共128位)来存放进程当前特权,每位标识一个CPF特权(为1标识具有此特权,为0标识不具有此特权).进程特权如何确定进程的特权是设计特权机制的关键.在CPF中,进程特权由进程当前特权状态和程序及进程用户的特权属性决定.当执行程序(execve)或调用set*id设置完用户标识和组标识后,都需要重新计算进程的特权.首先使用将进程当前状态与进程对应的程序特权属性项中的特权状态标识进行匹配.若匹配失败,则进程当前特权全置空.若匹配成功,则按以下公式计算进程当前特权:

进程特权 i=用户特权 i & 状态特权 i & 全局特权 i ($i=0\sim95$); 进程特权 i=状态特权 i ($i=96\sim127$)

进程特权信息的存储使用到了LSM在进程数据结构(task_struct)新增的成员security.此成员为一指向void的指针,用于指向存放进程安全属性的数据结构,进程的特权信息也被存放于这个数据结构中.此外,系统还维护一个指向当前特权状态特权参数列表的指针,供需要进行参数检查的特权检查时引用.

3.4 特权检查

3.4.1 检查点

虽然linux的权能机制实现的并不完善,但为保证向下兼容和利用遍及内核各处的权能的检查函数capable这一宝贵的资源,要求实现CPF时充分考虑与Linux Capability之间的关系.

CPF特权与权能的关系可分为3种情况:完全等同,细化和新增等.在实现CPF的检查机制时,前两种的情

© 中国科学院软件研究所 http://www.jos.org.cn

形要利用到原有Linux内核特权检查函数capable这一资源.LSM框架中为capable函数提供了钩子,相应的CPF检查机制可以在这个钩子中实现.但对于第 2 种情况,还需要获得当前进程是由于调用哪个系统调用陷入内核这一信息,必要的时候还需要知道调用参数,而LSM的capable钩子函数中未提供足够的相应信息.为此,作者研究分析了 Linux 系统调用机制,利用进程的系统堆栈(系统调用号与参数的存放区域)与进程的数据结构task_struct 处于连续的两个页面这一特性,通过进程数据结构指针(指向此二页面的起始位置)获得系统堆栈的地址直接访问系统堆栈区,获得系统调用号及其调用参数.这样作无须对 LSM 和内核代码作任何修改,在确保兼容性的基础上实现了系统功能. 对于新增的CPF特权,将在相应的其它LSM钩子函数中实现其检查机制.在有的情况下,LSM的钩子函数的参数也未能提供足够的判断信息,也需从内核系统堆栈中获得必要的信息.

需要特别注意的是,CPF特权机制仅仅利用了原Linux权能机制的capable函数,舍弃了其余的部分(权能计算,权能检查等),且CPF特权为Linux Capability的超集,包含了所有原有权能,原有的内核代码无需改动即可正常运行.

3.4.2 与其他访问控制间的关系

包含CPF的安全操作系统项目种还包括其它一些访问控制(MAC、DAC等),它们与CPF特权机制之间的关系可分为两大类:

(1) 大部分 CPF 特权是其它访问控制机制的补充.在这种情况下,访问请求必须同时满足所有的访问控制策略(MAC、DAC、CPF等),才能得到实施,它们之间是"AND"的关系.

(2) 某些CPF特权超越了其它访问控制策略.在这种情况下, 若具备相应的CPF特权,无论是否满足其它的访问控制策略,访问请求都可以实施,它们之间是"OR"的关系,如CPF_MAC_OVERRIDE等.

3.4.3 特权检查机制

在接收到访问请求时,系统首先进行MAC和DAC等访问控制策略的判断.如果访问请求不满足这些策略,则检查进程是否具有相应的超越特权,若有则允许访问进行,否则拒绝此请求.如果访问请求满足这些策略且访问请求为一特权操作,则还需检查进程是否具有相应的CPF特权.

在进行特权检查时,若待检查的特权为一带参特权,还需将访问请求的参数与进程当前特权状态特权参数列表中的相应项进行比较,只有相符才能进行特权操作.

3.5 特权配置管理

由于特权状态和特权参数的引入, CPF特权管理的复杂性与原有Linux特权机制相比有明显的增加.尤其是为了使得进程特权与程序逻辑相关,必须对各个特权程序的进行特权状态划分和特权设置,这是一个较为困难的工作.在CPF中,我们采取跟踪与源代码分析相结合的方法进行进程特权属性的配置.作者专门开发了一个负责跟踪进程特权操作的 LSM 安全模块 CPFMonitor.此模块的工作原理是利用 LSM 钩子截获选定程序（进程）的系统调用,集中跟踪进程的特权状态变化和特权操作(包括敏感系统调用),将相关系统调用号,调用参数,调用次序和调用环境等信息存放在一个跟踪数据库中供分析所用.在运行选定特权程序前将此安全模块装载进内核,CPFMonitor 模块将记录下相应进程的关键系统调用信息.系统管理员可在跟踪记录的辅助下有目的地对程序代码进行分析,从而最终形成系统CPF特权策略.利用CPFMonitor,作者已经完成了基本的Linux系统的CPF特权策略配置,涵盖了 login, passwd, X Server 等常用特权程序.在此配置的控制下,系统运行正常.

系统中的特权进程在一般情况下较为固定,对特权进程的配置并不需要经常进行.CPF 的特权配置虽然较复杂,但仍然在可以接受的范围内.而且为了有效支持最小特权原则,这样的复杂性是不可避免的.

3.6 应用示例

以下我们采用一个例子来说明CPF特权机制的应用.wu-ftpd是目前Linux世界中使用最广的文件传输服务器之一,在安全领域中常被作为研究分析的对象.wu-ftpd的服务过程可划分为4个状态: (1) wu-ftpd后台监控进程的用户ID属性全为0(root),当有连接产生时,为此连接创建一个新的服务进程;(2) 在用户身份鉴别完成后,设置服务进程的有效用户 ID(euid)为连接用户的 ID.在余下的服务过程中,服务进程一般以连接用户的身份运行;(3) 在某些需要root权限的场合下,服务进程将其有效用户 ID 设置为 root,转换到状态 3(状态 2 可转换到的

© 中国科学院软件研究所 http://www.jos.org.cn

状态集只包括状态 3)以 root 身份进行特权操作,在完成特权操作后,又将其有效用户 ID 设置回连接用户 ID 回到状态 2 进行普通操作;(4) 在传输过程中,用户可能需要执行一些外部命令(tar, compress 等),为此系统由状态 2 转换到状态 3,创建一个新进程并将其全部用户 ID 属性全置为连接用户 ID.此新进程调用 execve 系统调用执行相应的外部命令后结束.

以上 4 个状态按照需要被分配了相应的 CPF 特权,其中状态 1 分配了绑定 1024 以下端口和设置进程用户 ID 等 CPF 特权;状态 2 未被分配任何 CPF 特权,但规定了其调用 *set*uid* 的参数并限定其只能转换到状态 3;状态 3 分配了改变根目录(chroot),超越自主访问控制和改变文件属主等 CPF 特权;针对 wu-ftpd, CPF 将对一些与系统安全密切相关的系统调用(execve, set*uid 和 kill)的调用标识为特权操作加以控制.在状态 4 中,进程被分配了调用 execve 的特权,但通过特权参数对其能执行的程序进行了限制.

在配置完毕后,利用 wu-ftpd 中存在的一个缓冲区溢出漏洞对系统进行了拒绝服务攻击试验.在未装载 CPF 机制时,攻击代码在顺利获取了超级用户权限后,调用 reboot 系统调用关闭系统使之停止一切服务,成功地实施了攻击;装载 CPF 机制后,虽然入侵者进入了系统,但受 CPF 特权机制限制,无法执行关闭系统的操作,攻击失败.

4 相关工作

Linux 入侵检测系统 LIDS[5]扩展了基本 Linux 系统所提供的权能机制,能对部分程序进行某种程度上的细粒度特权控制,在关键特权与进程逻辑之间建立了一定程度的联系.对于比较重要的特定权能,系统可进行全局性的禁止,对于确实需要此权能方能正常运行的程序,可单独为其分配已被禁止的权能.这些单独分配的权能可认为是一种对禁止的超越.LIDS 的特权处理方法比较新颖,与原有机制兼容,效率也较高.但 LIDS 这种对少数关键权能进行的“先禁止后超越”的特权控制方式仍然不能彻底支持最小特权原则,且 root 进程仍可能具有不必要的多余特权(多余的未被禁止的权能).在实现上,LIDS 也已移植到了 LSM 框架上,LIDS 安全模块能够直接插入标准 Linux 内核进行安全控制.

TIS 公司研制的 Trusted XENIX[6]是安全操作系统领域的一个重要的产品,是为数甚少的通过了 TCSEC B2 级标准的安全操作系统.作为一个类 Unix 操作系统,其特权机制的设计实现对研究开发 Linux 特权机制具有重要的参考和借鉴价值. Trusted XENIX 的特权机制分为两大部分:(1)特殊用户和组标识类,借助特殊的 UID 或 GID 获得普通用户不具备的对系统管理文件和目录的自主访问控制权限,此类特权机制由自主访问控制(discretionary access control, DAC)和用户标识机制实现.(2)统一特权机制(generalized privilege mechanism, GPM)类,系统定义 36 个 GPM 特权,每个 GPM 特权对应一个特定特权操作,程序文件为 GPM 特权的属主,特权进程根据其程序文件动态获取 GPM 特权,具有 GPM 特权的进程可启动特权系统调用或非特权系统调用的特权选项.GPM 特权可在进程操作过程中动态获取和废弃(进程主动调用一个新增加的系统调用对进程当前特权进行设置),进而,特权操作可明确限定在一个较小的代码区,在一定程度上实施了最小特权原则.总的来说,Trusted XENIX 的特权机制是已有系统中较好的之一,但仍然存在以下不足:GPM 特权完全由程序文件决定,用户不是 GPM 特权的属主,不同的用户只要能执行特权进程都可以获得同样的 GPM 特权;其次,特殊用户和组标识类特权的实现部分依赖于自主访问控制,靠用户所能影响的自主访问控制来实现特权机制容易导致失控的特权操作,这类特权机制仍然是基于 UID 的;此外,进程操作过程中特权的动态变化依靠进程主动调用相应系统调用实现,对已经存在的应用程序,只能通过修改其源代码来实现特权的动态变化.

REMUS[4]是 Bernaschi 等人实现的一个安全增强 Linux 操作系统,本质上为一沙盒(Sandboxing)系统.在 REMUS 中,将 Linux 系统调用按与系统安全的关系进行了分析,通过截获进程的关键系统调用进行判断将其行为限制在一合法的范围内.从特权控制的角度看,REMUS 未能提供一个完整的特权机制解决方案,很多 Linux 特权操作超出了 REMUS 的控制范围.REMUS 中对 Linux 系统调用进行安全相关分析所采用的方法较为科学,CPF 采用了类似的方法进行系统调用安全相关性分类分析,在分析的基础上进行了系统特权的划分.

BlueBox[7]为 IBM Thomas J. Watson 研究中心研发的一个策略驱动的基于主机的入侵检测系统,已在 Linux 操作系统上实现.其核心是通过截获并检查系统调用,创建一个用于在内核实施细粒度的进程权能控制机制的基础架构.进程的权能由一个规则(策略)集所指定,这些规则以可执行文件为单位规定了其访问系统资源的权

© 中国科学院软件研究所 http://www.jos.org.cn

限.从 BlueBox 系统的角度来看,CPF 也是一种策略驱动的访问控制系统,CPF 特权策略配置语言 PSL 的作用与 BlueBox 策略配置语言相当.BlueBox 在部分访问规则中引入了进程状态信息以抵御潜在的攻击.这与 CPF 中特权状态的思想类似,但 CPF 特权状态的划分面向的是进程特权操作,且提供了对状态切换的控制,划分的方法也较灵活.

5 结 语

针对传统 Linux 特权机制中存在的局限性,作者提出了一个新的 Linux 特权机制框架(CPF),并基于 LSM 以 Linux 可装载模块的形式实现了该框架.在 CPF 中,基于对系统调用的全面安全性分析,给出了一个与传统 Linux 特权划分向下兼容的细粒度的特权划分;在部分 CPF 特权引入了特权参数,特权与特权操作的参数之间的关系是显式的,并可根据具体的特权进程进行配置,大大扩展了特权机制控制的灵活性和准确度;提供了一个较为完善的特权计算机制,进程的用户和对应程序的特权属性参与了进程当前特权的计算,改变了传统机制中特权的计算完全依赖于对进程用户 ID 是否为 root 进行检查的状况;引入了进程特权状态这一新的概念,可将进程执行过程分部分进行特权控制,进一步细化了最小特权原则的支持粒度,并且这种控制完全由核心自动进行,对应用完全透明.CPF 的引入有效提高了对最小特权的支持力度,试验表明其能大大降低入侵事故的危害性.

References:

[1] IEEE Standards Department. IEEE Std 1003.1e draft. PSSG Draft 17, New York: IEEE Computer Society. 1997. 163~194.

[2] Chen H, Wagner D, Dean D. Setuid demystified. In: The 11th USENIX Security Symposium. Berkeley: USENIX Association, 2002. 171~190.

[3] Wright C, Cowan C, Morris J, Smalley S, Kroah-Hartman G. Linux security modules: general security support for the Linux kernel. In: The 11th USENIX Security Symposium. Berkeley: USENIX Association, 2002. 17~31.

[4] Benaschi M, Gabrielli E, Mancini LV. REMUS: A security-enhanced operating system. ACM Trans. on Information and System Security, 2002,5(1):36~61.

[5] Xie HG, The Linux intrusion detection system. 2001. http://ftp.engardelinux.org/pub/engarde/stable/docs/LIDS/LIDS.pdf

[6] National Computer Security Center. Final evaluation report TIS Trusted XENIX version 4.0. CSC-EPL-92/001.A, National Computer Security Center, 1994. 33~44.

[7] Chari SN, Cheng PC. BlueBox: A policy-driven, host-based intrusion detection system. ACM Trans. on Information and System Security, 2003,6(2):173~200.

© 中国科学院软件研究所 http://www.jos.org.cn