



## 使用基于状态的特权控制实施最小特权原则

---

梁 彬

北京启明星辰信息技术有限公司

特权 (Privilege) 是进程执行一些安全相关操作所必须具备的权限。

滥用特权将导致非常严重的后果。特权进程的可信性面临巨大的挑战：

- 漏洞
- 恶意代码
- 缓冲区溢出

解决方法：有效支持支持最小特权原则 (Principle of Least Privilege)

特权使用的控制可用最小特权原则概括：

系统仅赋予主体（用户或进程）完成当前工作所必不可少的特权。

实施最小特权原则可使得由可能的事故、错误和非授权访问等原因造成的损失降到最低！

# X'con 2004 操作系统特权机制的归纳比较

操作系统	特权机制类型	进程特权与程序逻辑关系	是否存在超级用户	对最小特权原则支持程度
传统UNIX	基于UID的特权机制	无	是	差
OpenServer 5	基于UID的特权机制	无	是	差
UnixWare 7	基于文件的特权机制	有，但受超级用户机制影响	是	有限度支持
Linux	基于UID的特权机制	无	是	差
Windows	基于UID的特权机制	无	无，但在默认设置下系统管理员为变相的超级用户	差
Trusted Xenix	混合类型	部分特权有	无，但存在访问关键文件的特权用户	有限度支持
LIDS	混合类型	有，但不是强制要求，仅对被禁止功能起作用	是	有限度支持

注：混合类型是指同时具有基于UID的特权机制和基于文件的特权机制的特点

- 要较好地支持最小特权原则，应遵循以下原则：
  1. 以进程（程序）逻辑为中心进行特权控制；
  2. 以进程特权相关属性为依据对进程生命周期进行划分，进程在各个阶段中根据进程逻辑分配不同的特权；
  3. 用户特权属性仅作为一种全局性的约束；
  4. 在部分特权中引入特权参数提供更细粒度的特权控制；
  5. 与原有机制的尽量兼容，对应用程序透明。

基于以上原则，提出了一种新的特权控制模型——基于状态的特权控制模型SBPC (State-based Privilege Control)，作为实施特权控制机制的基础模型。

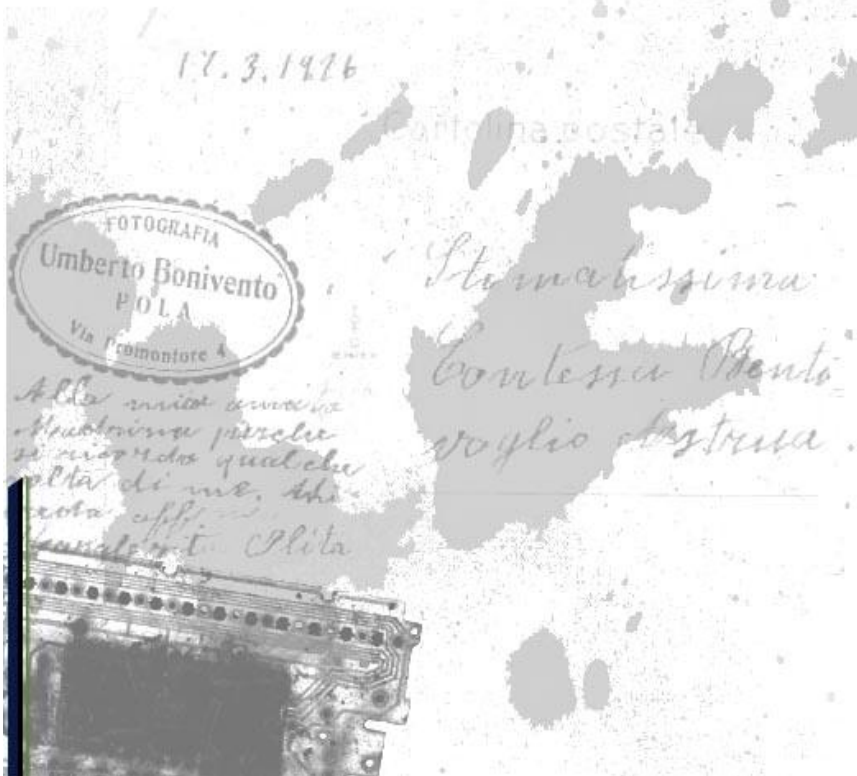
SBPC目标：

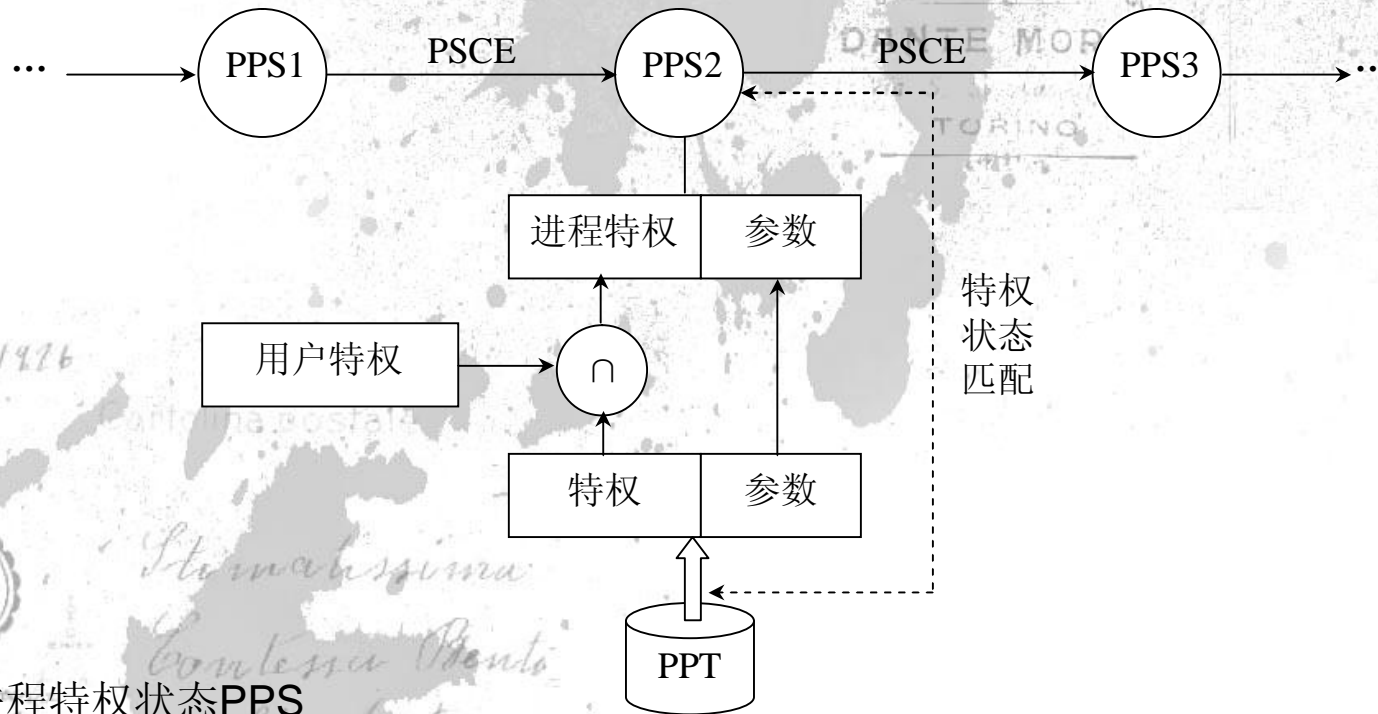
进程特权与进程应用逻辑、进程用户、进程状态（当前工作）相关

SBPC 中引入以下概念：

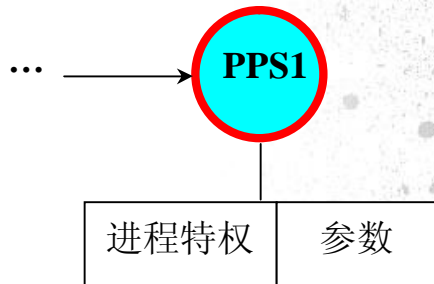
1. **进程特权状态PPS**，用来对进程的特权行为进行分块控制。进程特权状态由进程属性中的特权相关部分构成，是进程特权变化的标志；
2. **特权状态改变事件PSCE**，是使得进程特权状态发生变化的进程行为；
3. **程序特权表PPT**，存放了程序的所有可能的特权状态和其对应特权的信息，若有必要还包括相应的特权参数，其内容由程序逻辑决定。

SBPC将进程整个生命周期在特权状态空间内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，当特权状态改变事件使得进程特权状态发生变化时，根据程序特权表和用户特权为新特权状态下的进程分配特权。



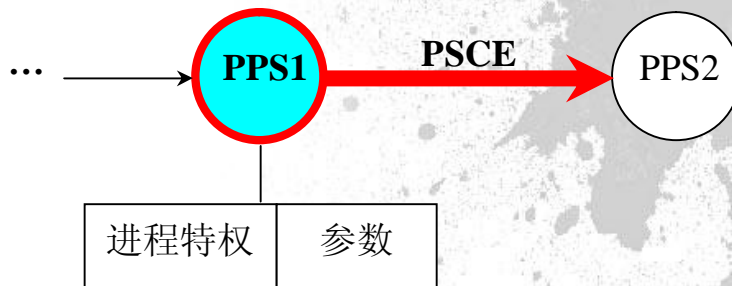


进程特权状态PPS  
特权状态改变事件PSCE  
程序特权表PPT



进程特权状态PPS  
特权状态改变事件PSCE  
程序特权表PPT

进程当前特权状态为PPS1，具有相应的特权及许可参数



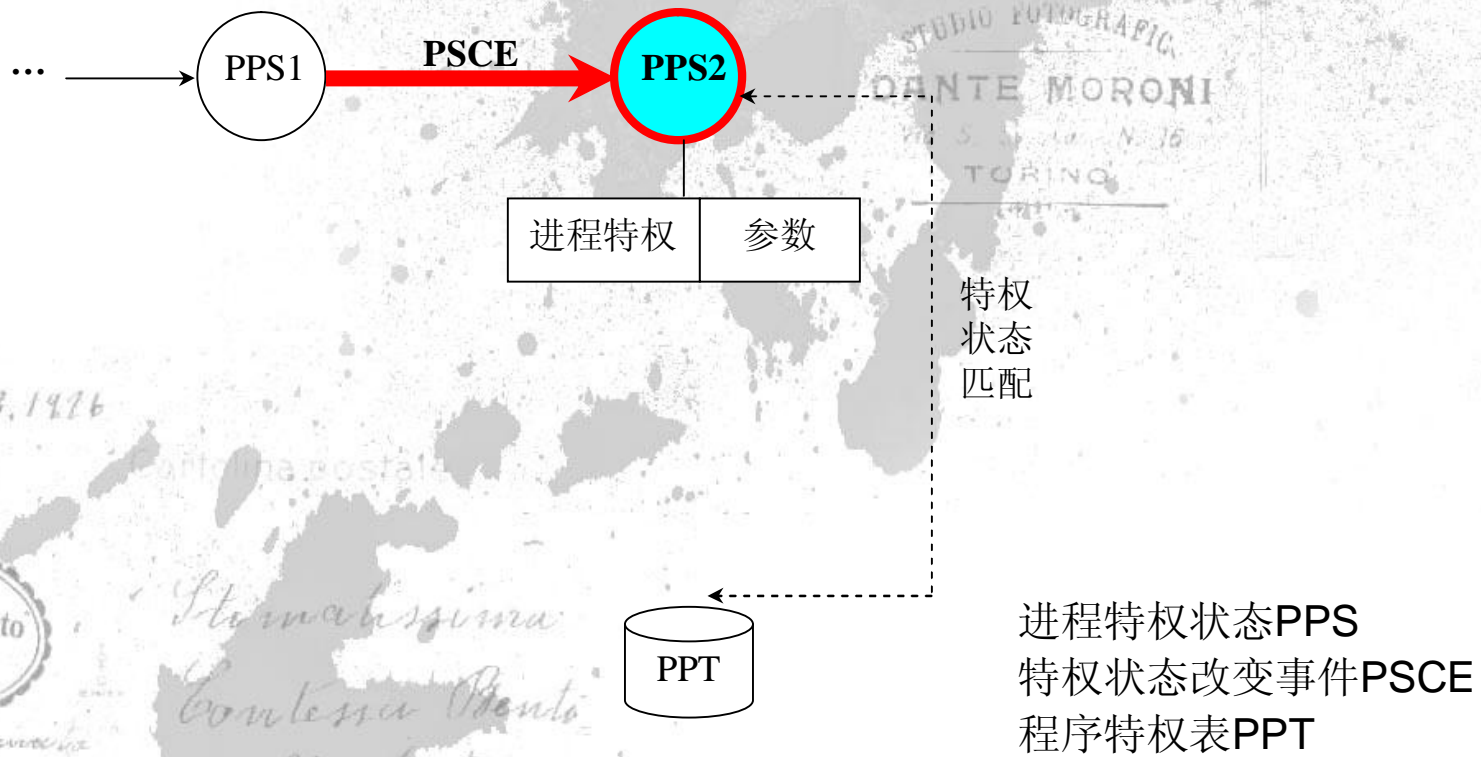
进程特权状态PPS  
特权状态改变事件PSCE  
程序特权表PPT

一个状态改变事件（PSCE）导致了进程特权状态转换（从PPS1到PPS2）

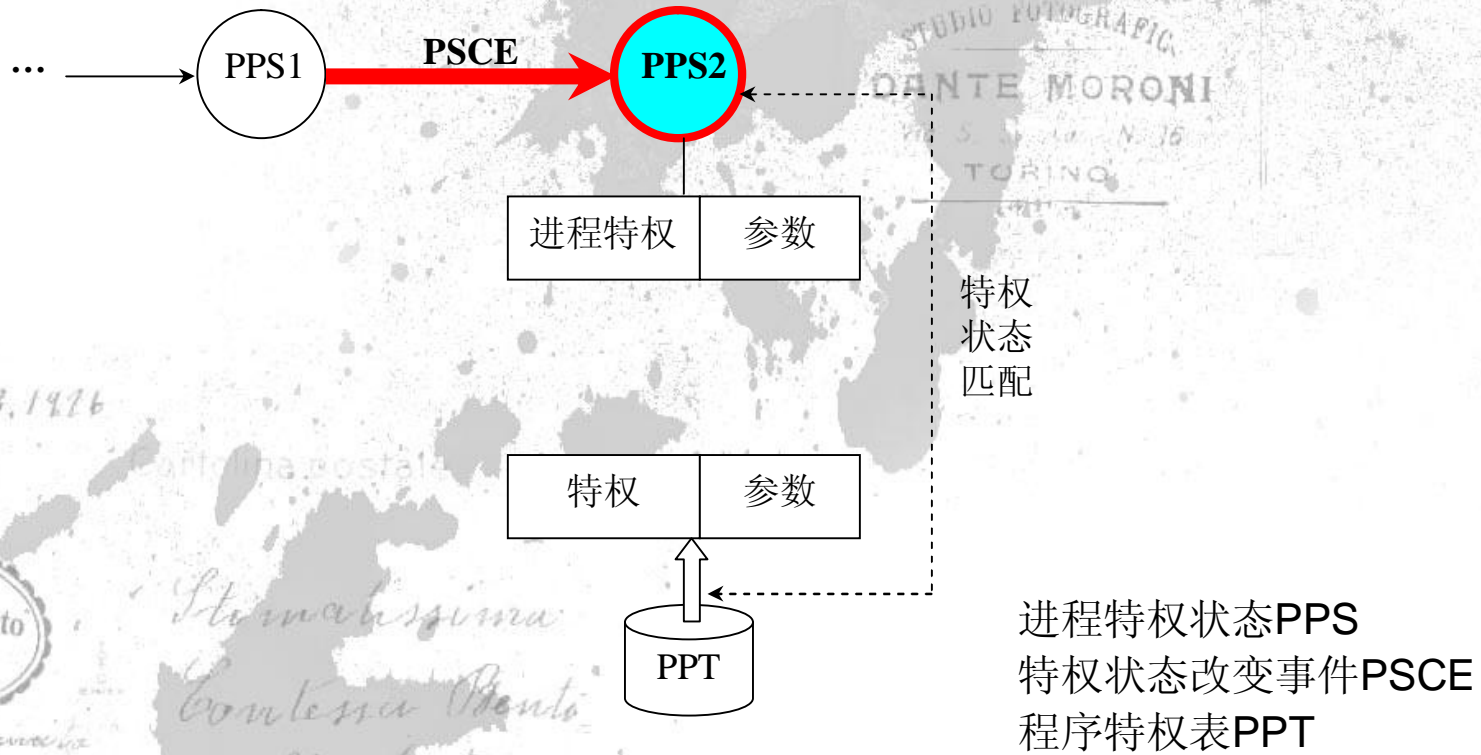


进程特权状态PPS  
特权状态改变事件PSCE  
程序特权表PPT

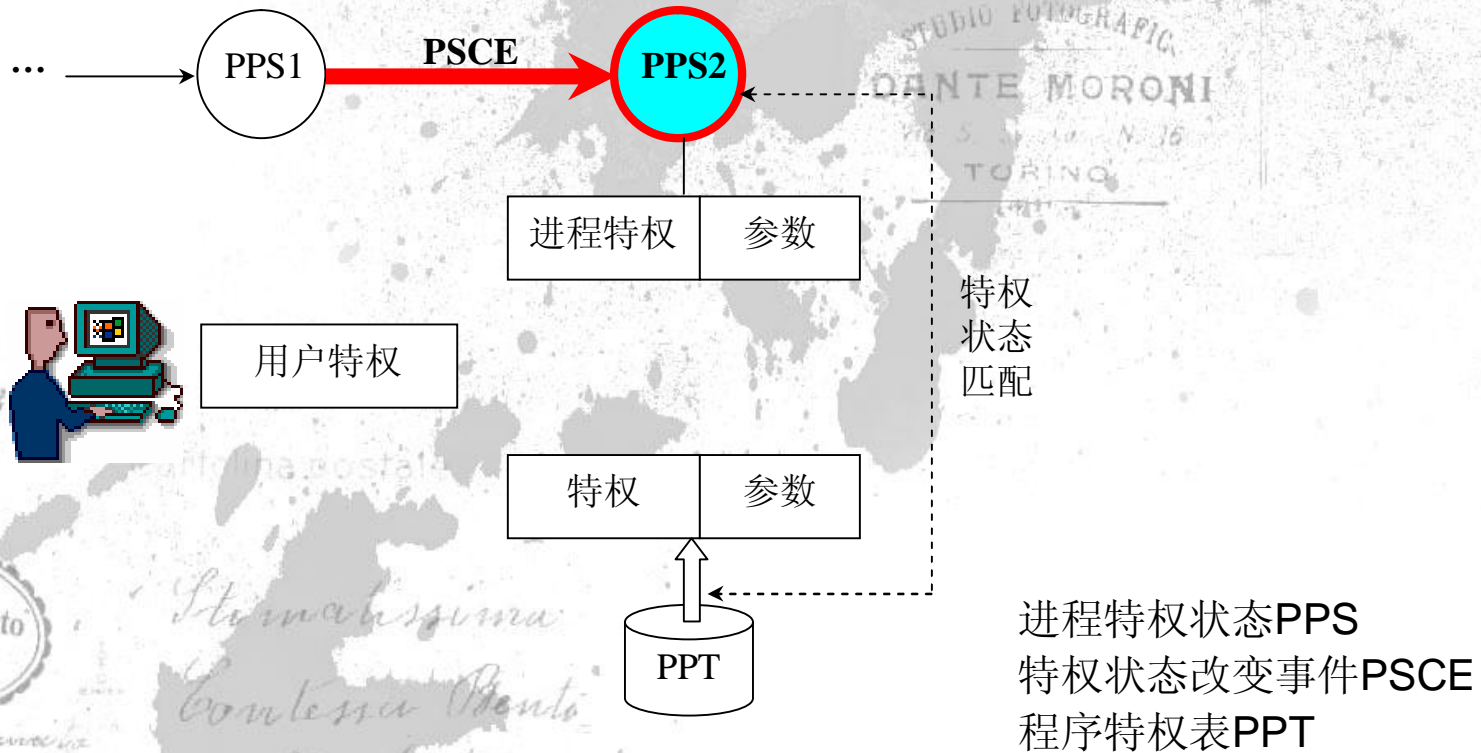
进程当前特权状态变为PPS2，需要为其计算此状态下的特权及许可参数



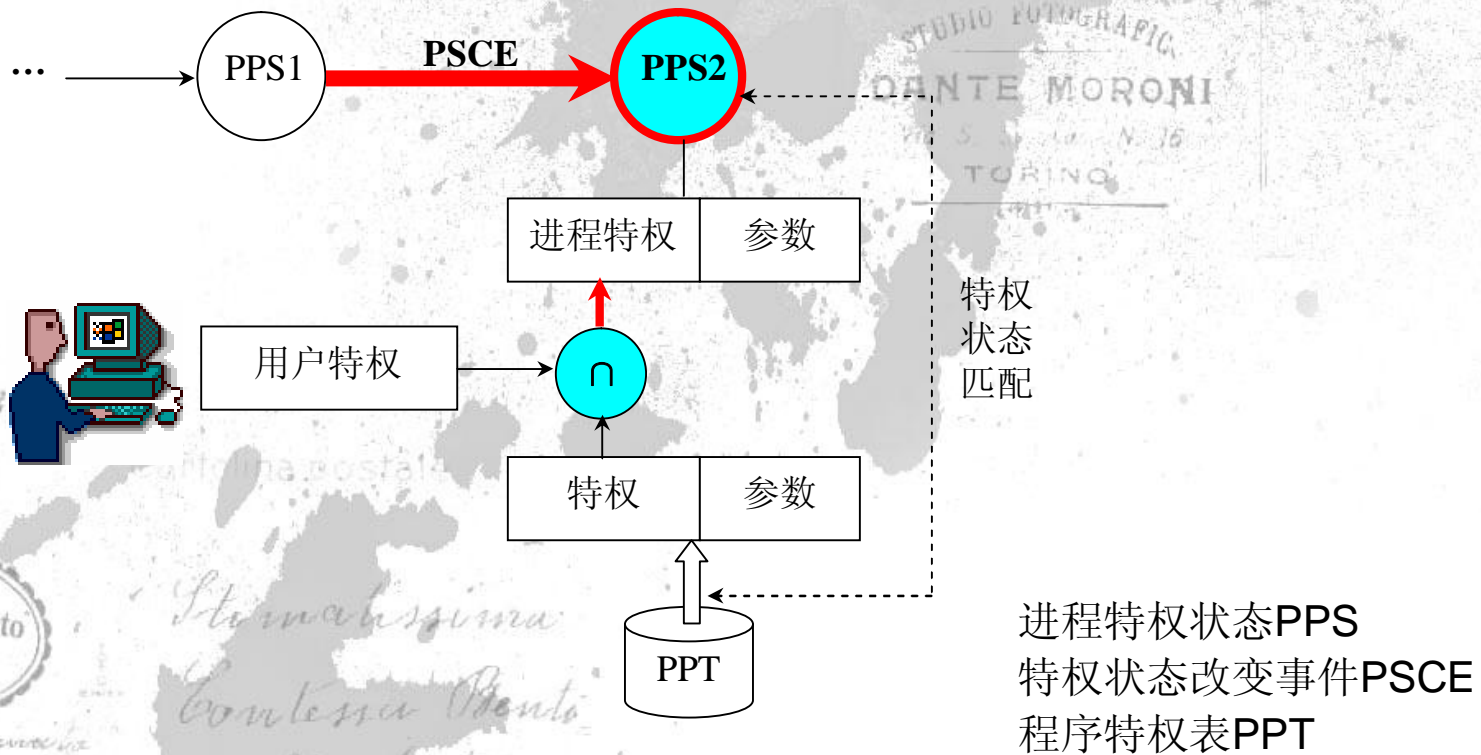
与程序特权表（PPT）项进行匹配



若匹配成功，则存在此状态下的相应PPT表项

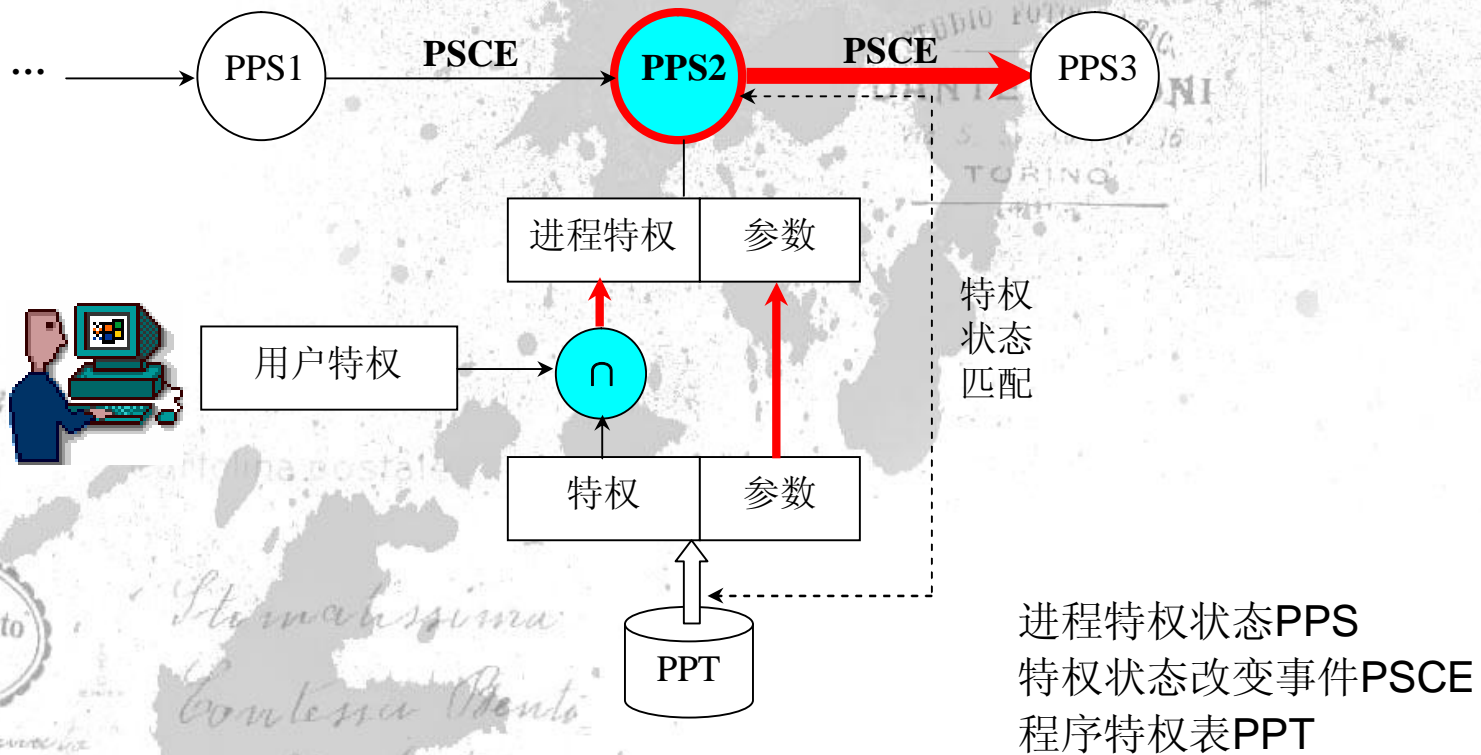


进程特权计算还涉及到进程当前用户的特权属性

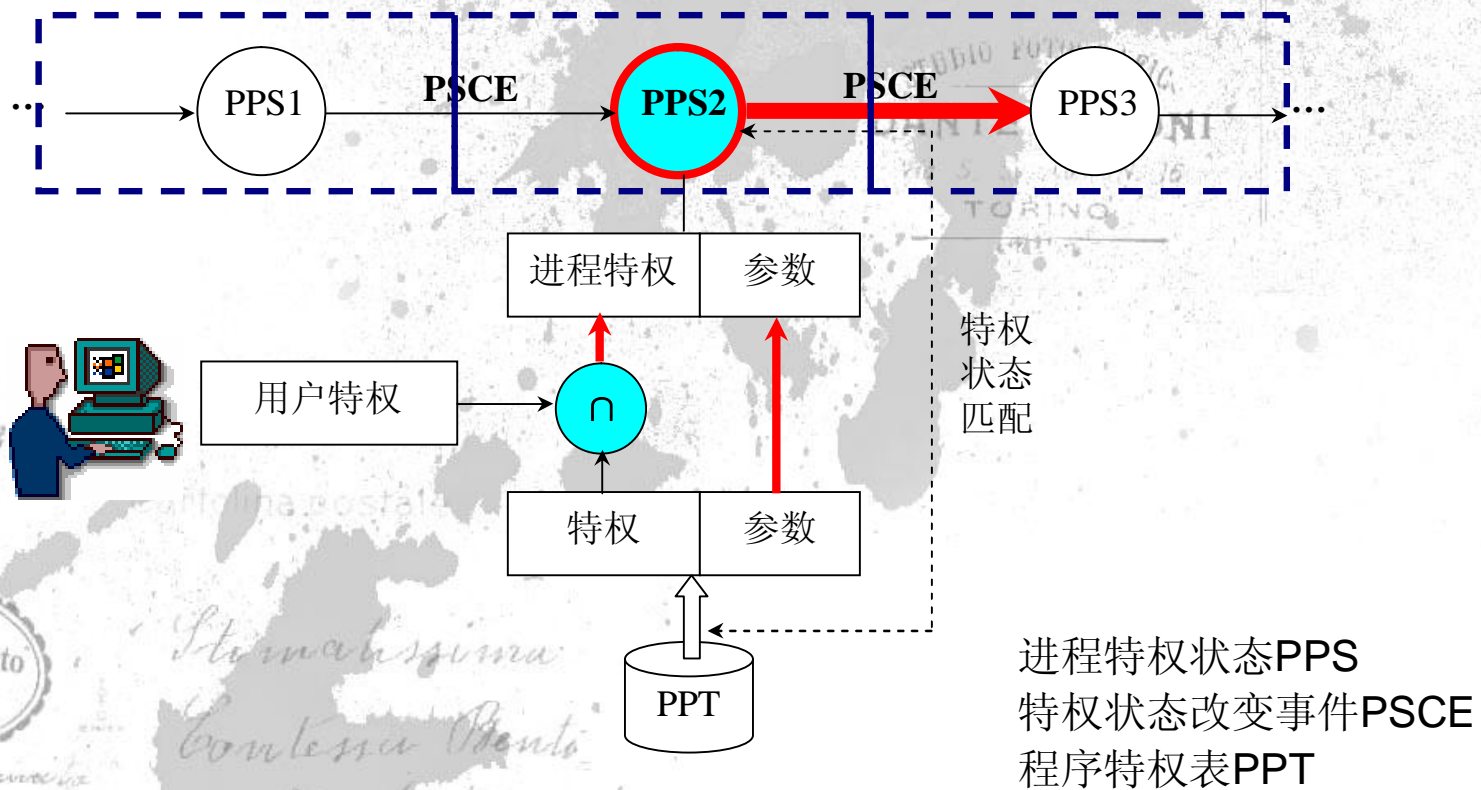


相应PPT表项中的特权与用户特权之交成为新状态下的进程特权





若未来进程的 privilege 状态又发生了变化，以上过程将重复进行

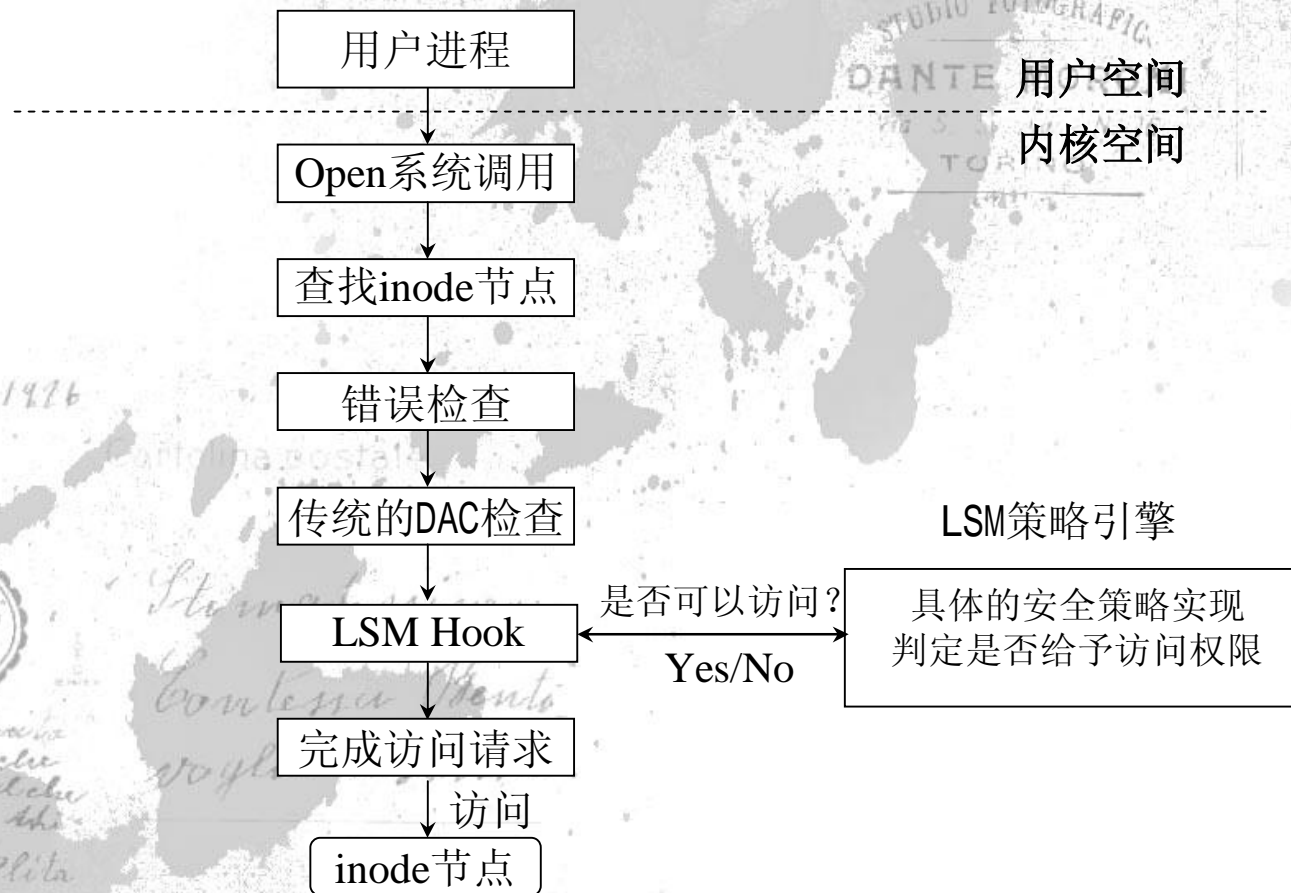


效果：进程生存周期被划分为多个阶段（特权状态），在不同的特权状态下根据其逻辑和当前用户被分配给不同的特权，进程特权被限制在一个较小的范围内

在Linux中实现了SBPC的一个原型系统——受控特权框架CPF（Controlled Privilege Framework）。

CPF所采用的基本技术是截获Linux系统调用（内核函数），由一个引用监控机制进行检查。

CPF选用了LSM（Linux Security Module）框架作为CPF实施的基础。



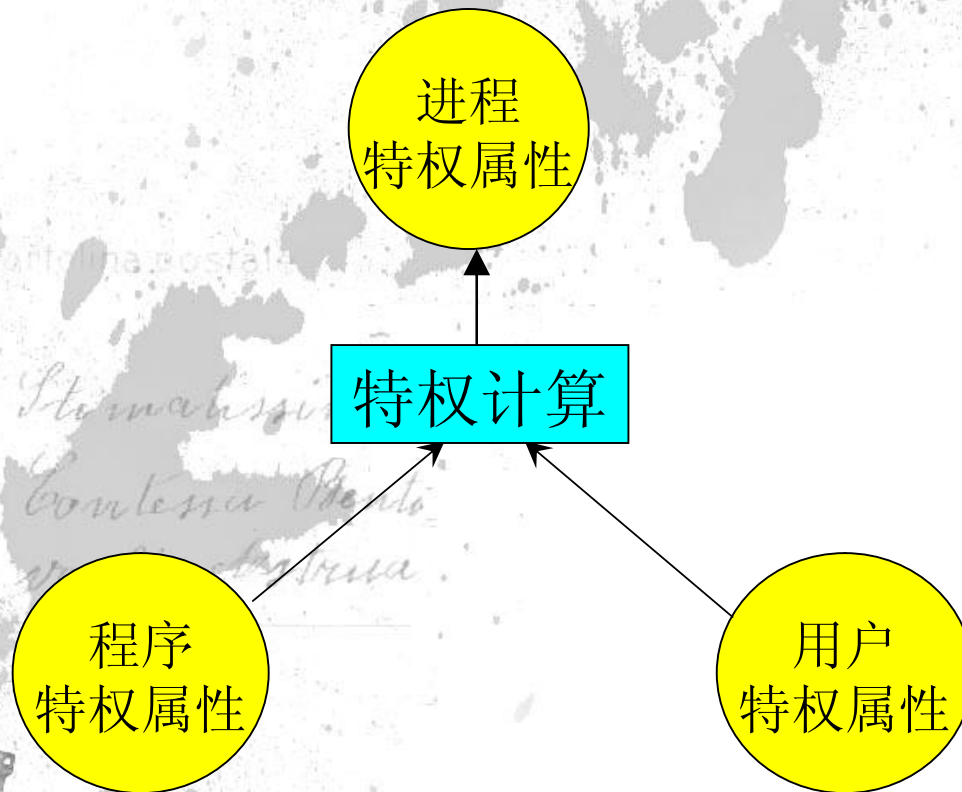
在CPF中，基于系统调用安全相关性分析，设置了71个特权（No.0~64, 95~101），大致可以分为以下9类：

1. Linux原有权能的细化；
2. 等价于Linux原有权能；
3. 调用特权系统调用（例如\_sysctl）；
4. 安全策略配置文件访问特权；
5. 系统文件访问特权；
6. CPF机制自身控制特权；
7. 审计特权；
8. 强制访问控制特权；
9. 危险环境下调用敏感系统调用的特权(No.95~101)

CPF的特权设置具有以下特点：

- 较细粒度的特权划分；
- 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，并可根据具体的可信进程进行配置，进一步保证了特权操作的正确性（限定了特权操作的相关参数）；
- 将在危险环境中进行敏感系统调用也作为特权统一处理，通过特权机制进行调用检查；
- CPF特权设置完全包含了Linux Capability权能，为其一超集，并新增了一些特权。

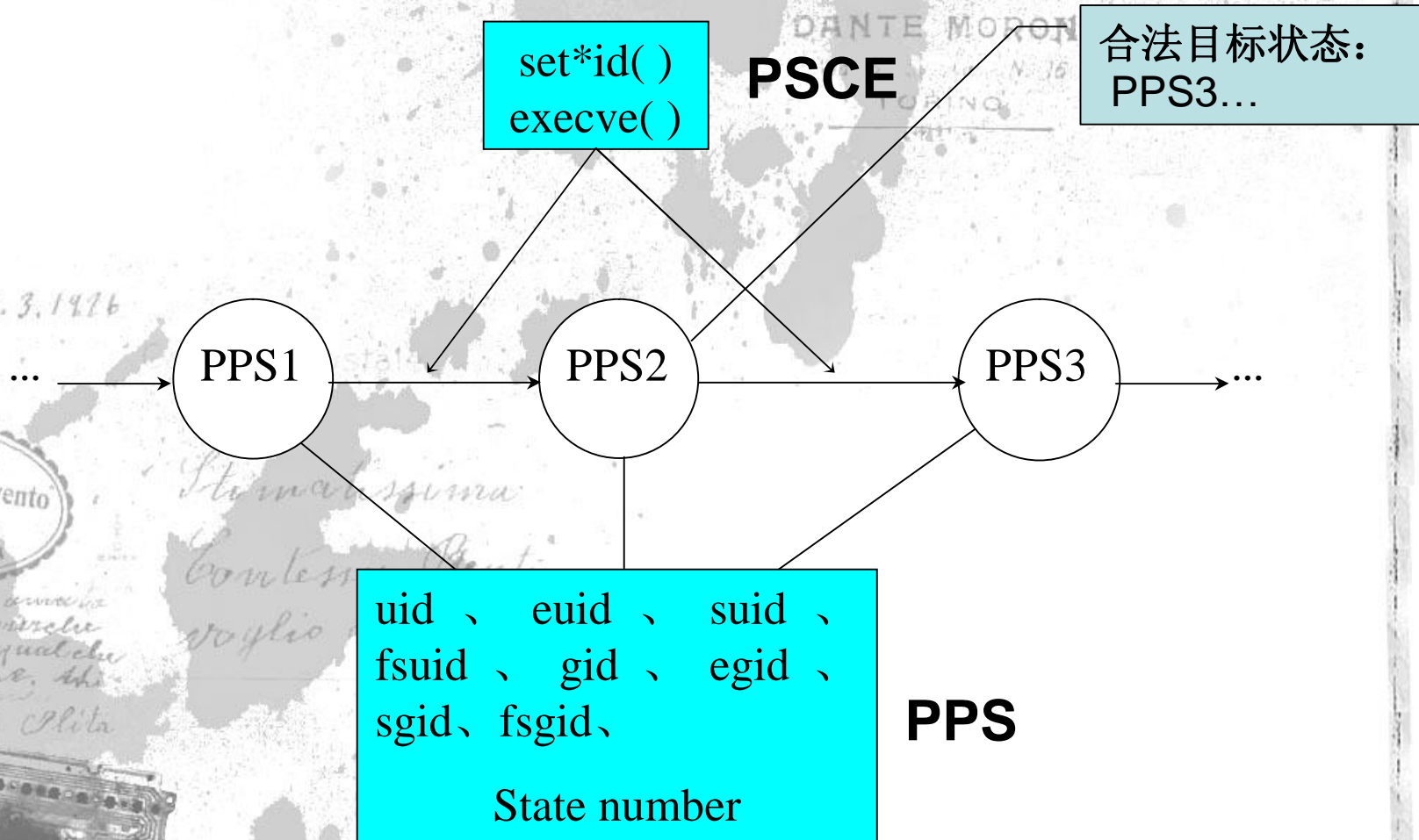
用户、程序文件和进程三种实体拥有特权属性

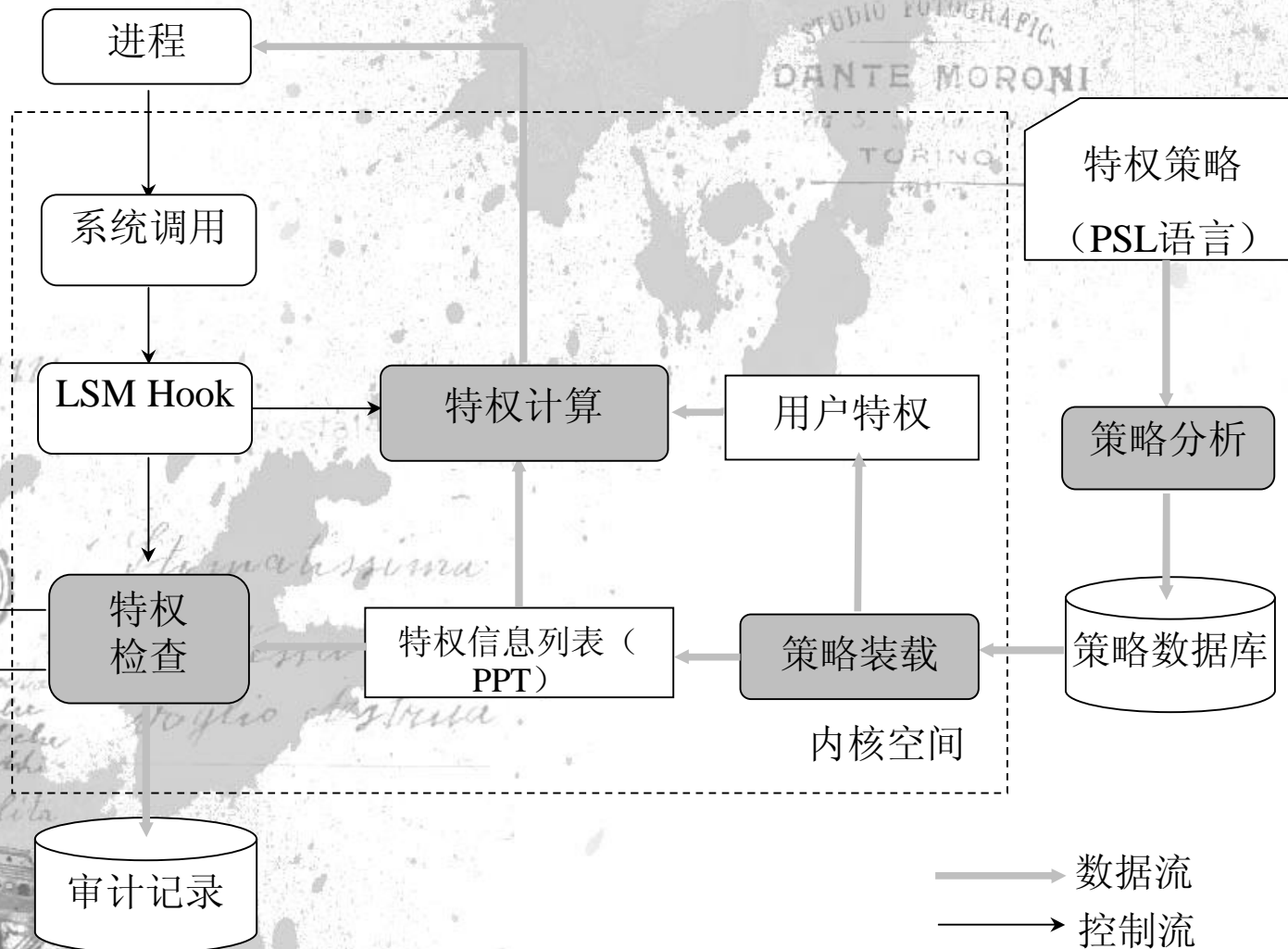


为了和数量巨大的Linux重要应用向下兼容

- 采用整个进程用户和组ID集 (*uid*、*eid*、*suid*、*fsuid*、*gid*、*egid*、*sgid* 和 *fsgid*) 作为特权状态的划分依据。另外，对每个特权状态进行了编号。
- 进程对 *execve* 和 *set\*id* (*setuid*、*setreuid*、*setresuid*、*setfsuid*、*setgid*、*setregid*、*setresgid*、*setfsuid* 和 *setfsgid* 等) 的调用导致了特权状态之间的转换，构成了PSCE集。

- 每个PPS都被赋予一个由合法目标状态编号组成的合法目标状态集。
- 在进行特权状态转换时，除了匹配ID外，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。





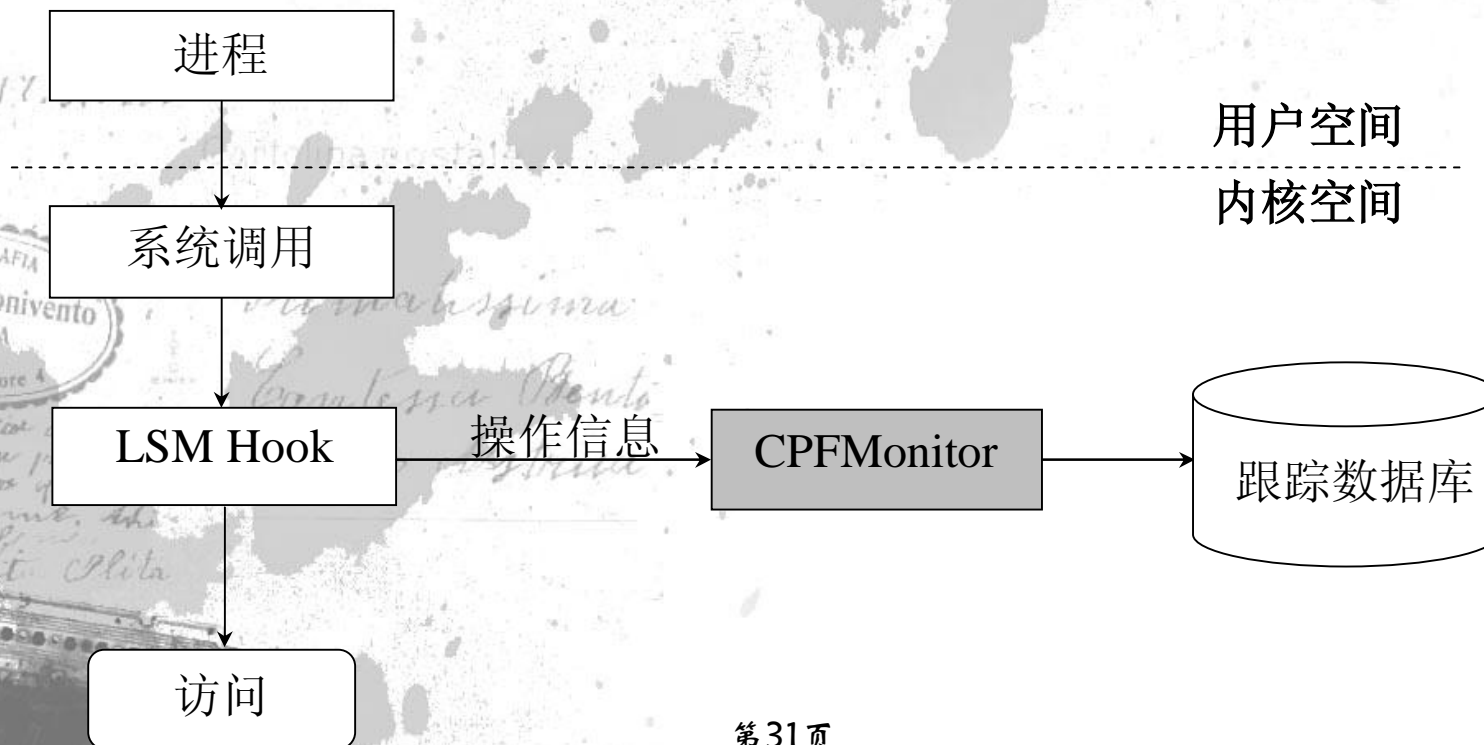
- 特权策略
  - | 用户特权属性
  - | 程序特权属性
  - | 安全策略文件与系统文件的标识
- 配置
  - 17 | 设计实现了一门配置语言PSL (Privilege Specification Language) 对特权策略进行说明，特权策略被存放在特权策略文件中
  - 策略文件通过策略分析器分析（编译）后存放在一个二进制的特权数据库中。

- 程序特权属性 (PSL配置语法)

```
#begin_prog
  path:程序路径名
  #begin_state
    stateno:状态编号
    canswitchto:{ 此状态可达状态的编号列表 }
    users:用户列表
    groups:组列表
    controlled_syscalls:{ 此程序敏感调用列表 }
    privileges:{ 此状态具有的CPF特权列表 }
    #begin_param
      param:带参特权名称
           特权参数列表
    #end_param
           其余特权参数...
  #end_state
  其余状态...
#end_prog
```

## 3.5.1 特权策略说明和分析

- 进程特权配置获取
  - | 源代码分析
  - | 跟踪



- 内核空间的策略装载机读取特权数据库中的特权信息，将其装入内核中。
- 采用了4个Hash表存放特权相关信息：
  - | 用户特权列表
  - | 特权进程特权相关信息 (PPT)
  - | 安全策略配置文件列表
  - | 系统文件列表

当执行程序 (execve) 或调用set\*id设置完用户和组ID属性后，都需要重新计算进程的特权。

- LSM Hooks:

```
17 | security_bprm_compute_creds()  
   | security_task_post_setuid() ...
```

- 进程特权和其它特权相关信息被存放在LSM在进程内核数据结构中添加的安全域中。

```
>security->cpf_security
```

- 计算方法:

- |  $PP_i = UP_i \ \& \ SP_i \ \& \ GP_i \ (i = 0 \sim 95)$

- |  $PP_i = SP_i \ (i = 96 \sim 127)$

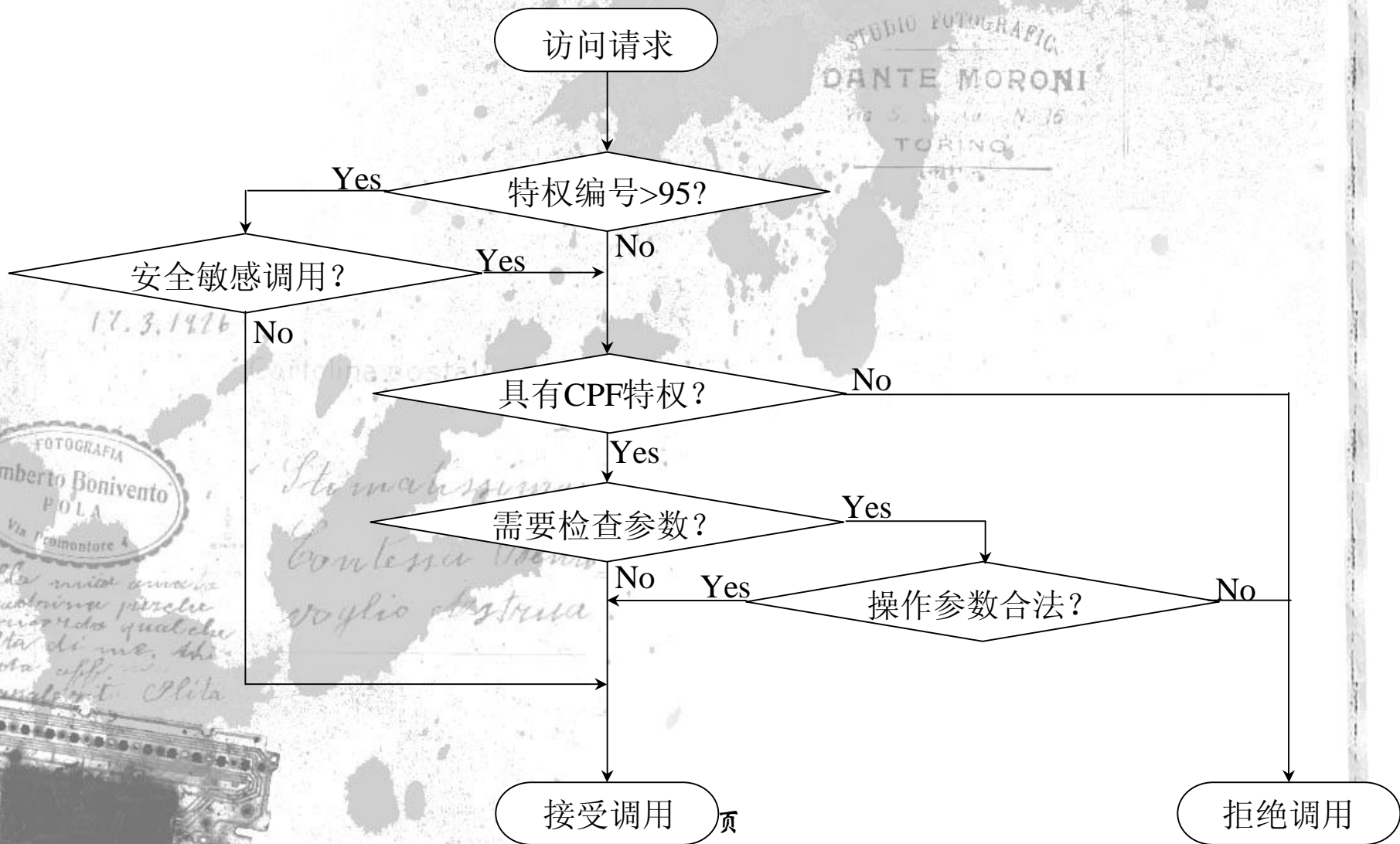
PP: Process Privilege

UP: User Privilege

SP: Program State Privilege

GP: Global Privilege

其中GP为全局性禁止某个特权操作的开关。



- 检查点

- | `Security_capable()`

- 用于检查与Linux Capability相关的CPF特权

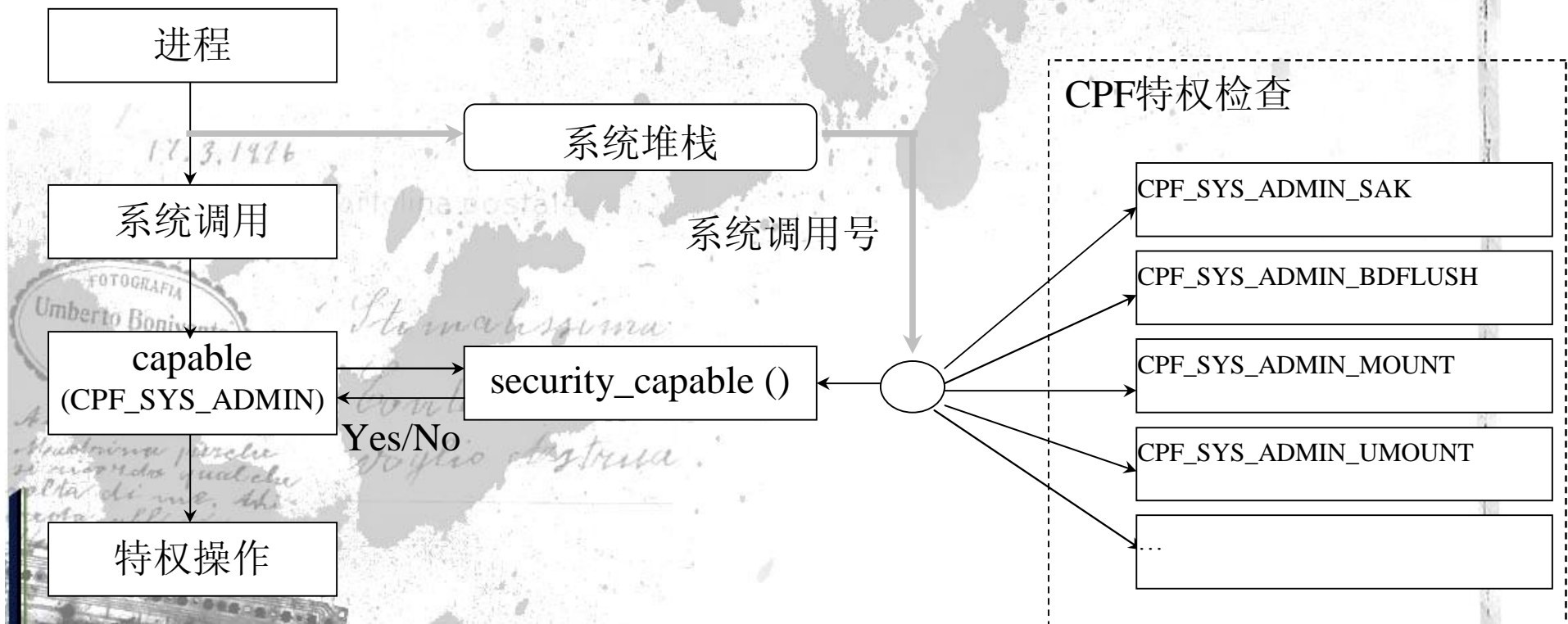
- | 系统调用钩子函数

- | `security_file_permission()`

- | `Security_sysctl()`

- | ...

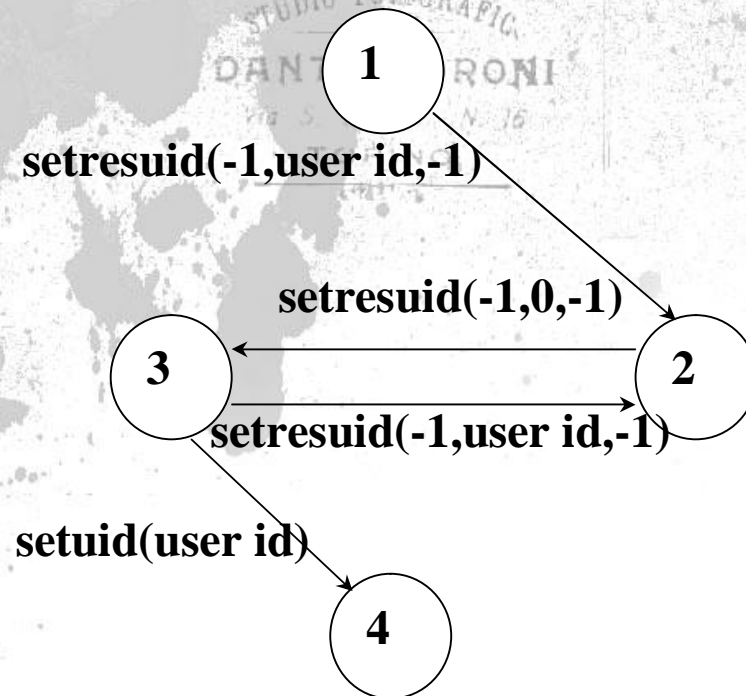
- 细化后的Linux Capability 特权检查



以下采用文件传输服务器wu-ftpd作为例子来说明如何采用CPF来进行特权检查以防御恶意攻击

- wu-ftpd特权状态
- wu-ftpd特权设置
- 攻击试验

如右图所示，根据 wu-ftpd 的服务过程，可将其划分为 4 个特权状态进行控制



1. wu-ftp后台监控进程的用户ID属性全为0 (root)，等待用户登录。当有连接产生时，为此连接创建一个新的服务进程。在用户身份鉴别完成后，设置服务进程的有效用户ID (euid) 和文件系统用户ID (fsuid) 为连接用户的ID，进程转到状态2。
2. 在余下的服务过程中，服务进程处于一个非特权状态，除非其需要进行某种特权操作。

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。

根据wu-ftpd的逻辑：

- W 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；
- W 状态2未被分配任何CPF特权，但限定其只能转换到状态3；
- W 状态3分配了改变根目录（chroot）、超越自主访问控制（override DAC）、和改变文件属主等CPF特权；
- W 在状态4中，execve、set\*uid和kill等系统调用被认为是安全敏感的系统调用，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。

```
[lb@secos rebootattack]$ ./dosattack -h192.9.200.122 -utest -p123
phase 1 - login... login succeeded
phase 2 - testing for vulnerability... vulnerable, continuing
phase 3 - finding buffer distance on stack... #####
      found: 1080 (0x00000438)
phase 4 - finding source buffer address... #####
      found: 0xbfffd79b
phase 5 - find destination buffer address... #####
      found: 0xbfffab30
phase 6 - calculating return address
      retaddr = 0xbfffd983
phase 7 - getting return address location
      found 0xbfffc4c
phase 8 - exploitation...
      using return address location: 0xbfffc4c
len = 510
```

此后远程主机立即关闭重启

```
[lb@secos shellattack]$ ./shellattack -h192.9.200.122 -utest -p123
phase 1 - login... login succeeded
phase 2 - testing for vulnerability... vulnerable, continuing
phase 3 - finding buffer distance on stack... #####
    found: 1080 (0x00000438)
phase 4 - finding source buffer address... #####
    found: 0xbfffd79b
phase 5 - find destination buffer address... #####
    found: 0xbfffab30
phase 6 - calculating return address
    retaddr = 0xbfffd983
phase 7 - getting return address location
    found 0xbfffc4c
phase 8 - exploitation...
    using return address location: 0xbfffc4c
len = 510
uid=0(root) gid=0(root) groups=501(test)
whoami
root
```

```
[lb@secos rebootattack]$ ./dosattack -h192.9.200.122 -utest -p123
phase 1 - login... login succeeded
phase 2 - testing for vulnerability... vulnerable, continuing
phase 3 - finding buffer distance on stack... #####
      found: 1080 (0x00000438)
phase 4 - finding source buffer address... #####
      found: 0xbfffd79b
phase 5 - find destination buffer address... #####
      found: 0xbfffab30
phase 6 - calculating return address
      retaddr = 0xbfffd983
phase 7 - getting return address location
      found 0xbfffc4c
phase 8 - exploitation...
      using return address location: 0xbfffc4c
      len = 510
read remote: Connection reset by peer
[lb@secos rebootattack]$
```

```
[lb@secos shellattack]$ ./shellattack -h192.9.200.122 -utest -p123
phase 1 - login... login succeeded
phase 2 - testing for vulnerability... vulnerable, continuing
phase 3 - finding buffer distance on stack... #####
    found: 1080 (0x00000438)
phase 4 - finding source buffer address... #####
    found: 0xbfffd79b
phase 5 - find destination buffer address... #####
    found: 0xbfffab30
phase 6 - calculating return address
    retaddr = 0xbfffd983
phase 7 - getting return address location
    found 0xbfffc4c
phase 8 - exploitation...
    using return address location: 0xbfffc4c
    len = 510
read remote: Connection reset by peer
[lb@secos shellattack]$
```

```
#begin_prog
  path:/usr/sbin/in.ftpd
  #begin_state
    stateno:1
    canswitchto:{ 2 }
    users:root root root root
    groups:all all all all
    controlled syscalls:{ setxuid execve }
    privileges:{
      CPF_NET_BIND_SERVICE
      CPF_SETXUID_CALL
      CPF_SETUID
    }
  #begin_param
    param:setxuid
    setresuid
    unchange !root unchange
  #end_param
#end_state
```

```
#begin_state
    stateno:2
    canswitchto:{ 3 }
    users:root !root root !root
    groups:all all all all
    controlled syscalls:{ setxuid execve kill }
    privileges:{ CPF_SETXUID_CALL }

    #begin_param
        param:setxuid
        setresuid
        unchange root unchange
    #end_param
#end_state
```

```
#begin_state
  stateno:3
  canswitchto:{ 2 4 }
  users:root root root root
  groups:all all all all
  controlled syscalls:{
    setxuid
    execve
    kill
  }
  privileges:{
    CPF_SYS_CHROOT
    CPF_CHOWN
    CPF_SETUID
    CPF_SETGID
    CPF_DAC_READ_SEARCH
    CPF_SETXUID_CALL
  }
#begin_param
  param:setxuid
  setresuid
  unchange oldeuid unchange
  setuid
  oldeuid unchange unchange
#end_param
#end_state
```

## 3.6.4 配置脚本 (4)

```
#begin_state
  stateno:4
  canswitchto:{ }
  users:!root !root !root !root
  groups:all all all all
  controlled syscalls:{ execve }
  privileges:{ CPF_EXECVE_CALL }
```

```
#begin_param
```

```
  param:execve
    /var/ftp/bin/ls
    /var/ftp/bin/compress
    /var/ftp/bin/tar
    /var/ftp/bin/pwd
    /var/ftp/bin/get
    /bin/ls
```

```
#end_param
```

```
#end_state
```

```
#end_prog
```

- 特权机制的性能测试有别于其它机制。对CPF集中测试系统调用在装载CPF前后的核心态时间开销。

系统调用 (100万次)	未装载CPF耗时 (s)	装载CPF耗时 (s)	CPF 开销 (%)
gethostname (非特权操作)	1.370	1.373	0.22%
sethostname (不检查特权参数)	1.098	1.365	24.32%
open (检查特权参数)	6.375	15.135	137.41%

由于特权操作在系统操作中的比重很小，检查参数的特权操作更少，开销属于可以接受的范围。总的来说，**CPF对系统整体性能影响较小。**

SBPC/CPF 以程序逻辑为中心，引入了特权状态的概念来进行特权控制；构建了特权与特权参数之间的显式关系；完善了特权进程的特权计算机制。可以对特权进程进行细粒度的、自动的特权控制，且对应用软件完全透明。恶意入侵的危害性被大大降低，能较好地支持最小特权原则。

POST CARD

CARTE POSTALE

Carte postale



STUDIO FOTOGRAFICO  
DANTE MORONI  
Via S. ... N. 16  
TORINO

END

谢谢各位

17.3.1926

Ufficio postale

FOTOGRAFIA  
Umberto Bonivento  
PIOLA  
Via Comandante 4

Stimolissima  
Contessa Monto  
voglio stupire

Alle uniche amiche  
che mi rimangono perché  
se ricordo qualche  
volta di me, che  
vota aff.  
L'Amante Plita

