Selinux

**1. 简介**

    SELinux带给[**Linux**](http://lib.csdn.net/base/linux)的主要价值是：提供了一个灵活的，可配置的MAC机制。

    Security-Enhanced Linux (SELinux)由以下两部分组成：

    1) Kernel SELinux模块(/kernel/security/selinux)

    2) 用户态工具

    SELinux是一个安全体系结构，它通过LSM(Linux Security Modules)框架被集成到Linux Kernel 2.6.x中。它是NSA (United States National Security Agency)和SELinux社区的联合项目。

    SELinux提供了一种灵活的强制访问控制(MAC)系统，且内嵌于Linux Kernel中。SELinux定义了系统中每个【用户】、【进程】、【应用】和【文件】的访问和转变的权限，然后它使用一个安全策略来控制这些实体(用户、进程、应用和文件)之间的交互，安全策略指定如何严格或宽松地进行检查。

    SELinux对系统用户(system users)是透明的，只有系统管理员需要考虑在他的服务器中如何制定严格的策略。策略可以根据需要是严格的或宽松的。

    只有同时满足了【标准Linux访问控制】和【SELinux访问控制】时，主体才能访问客体。

**1.1 DAC与MAC的关键区别(root用户)**

      安 全增强型Linux(SELinux)开始是由NSA（国家安全局）启动并加入到Linux系统中的一套核心组件及用户工具，可以让应用程序运行在其所需的最低权限上。未 经修改过的Linux系统是使用自主访问控制的，用户可以自己请求更高的权限，由此恶意软件几乎可以访问任何它想访问的文件，而如果你授予其root权 限，那它就无所不能了。

      在SELinux中没有root这个概念，安全策略是由管理员来定义的，任何软件都无法取代它。这意味着那些潜在的恶意软件所能造成的损害可以被控制在最小。一般情况下只有非常注重数据安全的企业级用户才会使用SELinux。

[**操作系统**](http://lib.csdn.net/base/operatingsystem)有两类访问控制：自主访问控制（DAC）和强制访问控制（MAC）。标准Linux安全是一种DAC，SELinux为Linux增加了一个灵活的和可配置的的MAC。

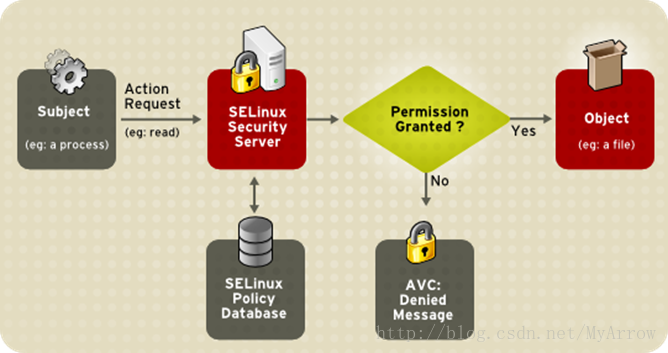
      所有DAC机制都有一个共同的弱点，就是它们不能识别自然人与计算机程序之间最基本的区别。简单点说就是，如果一个用户被授权允许访问，意味着程序也被授权访问，如果程序被授权访问，那么恶意程序也将有同样的访问权。 DAC最根本的弱点是主体容易受到多种多样的恶意软件的攻击，MAC就是避免这些攻击的出路，大多数MAC特性组成了多层安全模型。

      SELinux实现了一个更灵活的MAC形式，叫做类型强制(Type Enforcement)和一个非强制的多层安全形式(Multi-Level Security)。

      在Android4.2中，SELinux是个可选项，谷歌并没有直接取消root权限或其他功能。这是一个为企业级用户或是对隐私数据极为重视的用户提供的选项，普通消费者则完全可以关闭它。

**2. SELinux的运行机制**

    SELinux决策过程如下图所示：



      当一个subject(如: 一个应用)试图访问一个object(如：一个文件)，Kernel中的策略执行服务器将检查AVC (Access Vector Cache), 在AVC中，subject和object的权限被缓存(cached)。如果基于AVC中的数据不能做出决定，则请求安全服务器，安全服务器在一个矩阵中查找“应用+文件”的安全环境。然后根据查询结果允许或拒绝访问，拒绝消息细节位于/var/log/messages中。

**3. SELinux伪文件系统**

*/selinux/*伪文件系统kernel子系统通常使用的命令，它类似于*/proc/*伪文件系统。系统管理员和用户不需要操作这部分。*/selinux/*目录举例如下：

**[cpp]** [view plain](http://blog.csdn.net/myarrow/article/details/9856095/) [copy](http://blog.csdn.net/myarrow/article/details/9856095/)

-rw-rw-rw-  1 root root 0 Sep 22 13:14 access

dr-xr-xr-x  1 root root 0 Sep 22 13:14 booleans

--w-------  1 root root 0 Sep 22 13:14 commit\_pending\_bools

-rw-rw-rw-  1 root root 0 Sep 22 13:14 context

-rw-rw-rw-  1 root root 0 Sep 22 13:14 create

--w-------  1 root root 0 Sep 22 13:14 disable

-rw-r--r--  1 root root 0 Sep 22 13:14 enforce

-rw-------  1 root root 0 Sep 22 13:14 load

-r--r--r--  1 root root 0 Sep 22 13:14 mls

-r--r--r--  1 root root 0 Sep 22 13:14 policyvers

-rw-rw-rw-  1 root root 0 Sep 22 13:14 relabel

-rw-rw-rw-  1 root root 0 Sep 22 13:14 user

   如cat enforce其值可能如下：

   1： enforcing mode

   0： permissive mode

**4. SELinux配置文件**

    SELinux配置文件(configuration)或策略文件(policy)位于*/etc/*目录下。

**4.1 /etc/sysconfig/selinux配置文件**

     /etc/sysconfig/selinux是一个符号链接，真正的配置文件为：*/etc/selinux/config*

     配置SELinux有如下两种方式：

      1) 使用配置工具：Security Level Configuration Tool (system-config-selinux)

      2) 编辑配置文件 (/etc/sysconfig/selinux).  
  
      /etc/sysconfig/selinux中包含如下配置选项：

     1) 打开或关闭SELinux

     2) 设置系统执行哪一个策略(policy)

     3) 设置系统如何执行策略(policy)

**4.2 配置文件选项**

**4.2.1 SELINUX**

        SELINUX=enforcing|permissive|disabled —定义SELinux的高级状态  
        • enforcing — The SELinux security policy is enforced.  
        • permissive — The SELinux system prints warnings but does not enforce policy.  
        • disabled — SELinux is fully disabled. SELinux hooks are disengaged from the kernel and the pseudo-file system is unregistered.

**4.2.2 SELINUXTYPE(安全策略)**

         SELINUXTYPE=targeted|strict — 指定SELinux执行哪一个策略  
         • targeted — 只有目标网络daemons保护。每个daemon是否执行策略，可通过system-config-selinux进行配置。保护常见的网络服务，为SELinux默认值。  
         可使用如下工具设置每个daemon的布尔值：

         1) getsebool -a： 列出SELinux的所有布尔值

         2) setsebool： 设置SELinux布尔值，如：setsebool -P dhcpd\_disable\_trans=0，-P表示即使用reboot之后，仍然有效。  
  
         • strict — 对SELinux执行完全的保护。为所有的subjects和objects定义安全环境，且每一个Action由策略执行服务器处理。提供符合Role-based-Access Control(RBAC)之policy，具备完整的保护功能，保护网络服务、一般指令及应用程序。

**4.2.3 SETLOCALDEFS**

         SETLOCALDEFS=0|1 — 控制如何设置本地定义(users and booleans)。

         • 1：这些定义由load\_policy控制，load\_policy来自于文件/etc/selinux/<policyname>

         • 0：由semanage控制

**4.3 /etc/selinux/目录**

      /etc/selinux/是存放所有策略文件和主要配置文件的目录。其例子如下：   

**[cpp]** [view plain](http://blog.csdn.net/myarrow/article/details/9856095/) [copy](http://blog.csdn.net/myarrow/article/details/9856095/)

-rw-r--r--  1 root root  448 Sep 22 17:34 config

drwxr-xr-x  5 root root 4096 Sep 22 17:27 strict

drwxr-xr-x  5 root root 4096 Sep 22 17:28 targeted

**5. SELinux工具**

1) /usr/sbin/setenforce — 修改SELinux运行模式，例子如下：

         • setenforce 1 — SELinux以强制(enforcing)模式运行  
         • setenforce 0 — SELinux以警告(permissive)模式运行

    为了关闭SELinux，你可以修改配置文件：/etc/selinux/config或/etc/sysconfig/selinux  
  
  
2) /usr/sbin/sestatus -v — 显示系统的详细状态，例子如下：

**[cpp]** [view plain](http://blog.csdn.net/myarrow/article/details/9856095/) [copy](http://blog.csdn.net/myarrow/article/details/9856095/)

SELinux status:                 enabled

SELinuxfs mount:                /selinux

Current mode:                   enforcing

Mode from config file:          enforcing

Policy version:                 21

Policy from config file:        targeted

Process contexts:

Current context:                user\_u:system\_r:unconfined\_t:s0

Init context:                   system\_u:system\_r:init\_t:s0

/sbin/mingetty                  system\_u:system\_r:getty\_t:s0

3) /usr/bin/newrole — 在一个新的context或role中运行一个新的shell

4) /sbin/restorecon — 通过为适当的文件或安全环境标记扩展属性，设置一个或多个文件的安全环境

5) /sbin/fixfiles — 检查或校正文件系统中的安全环境[**数据库**](http://lib.csdn.net/base/mysql)

6) getsebool — getsebool -a：查看所有布尔值

7) setsebool — 参数-P，永久性设置

8) chcon 修改文件、目录的安全上下文  
      chcon –u[user]  
      chcon –r[role]  
      chcon –t[type]   
      chcon –R  递归

**6. 类型强制的安全上下文(Type Enforcement Security Context)**

    安全上下文是一个简单的、一致的访问控制属性，在SELinux中，类型标识符是安全上下文的主要组成部分，由于历史原因，一个进程的类型通常被称为一个域（domain），"域"和"域类型"意思都一样，我们不必苛刻地去区分或避免使用术语域，通常，我们认为【域】、【域类型】、【主体类型】和【进程类型】都是同义的，即都是安全上下文中的“TYPE”。

    SELinux对系统中的许多命令做了修改，通过添加一个-Z选项显示客体和主体的安全上下文。

    1) 系统根据PAM子系统中的pam\_selinux.so模块设定登录者运行程序的安全上下文；  
    2) 文件的Security Contex规则如下：

        • rpm包安装的：会根据rpm包内记录来生成安全上下文；

        • 手动创建的文件：会根据policy中规定的来设置安全上下文；

        • cp：会重新生成安全上下文；

        • mv：安全上下文则不变。

    3) id -Z   
        显示了你的shell的安全上下文；  
    4) ps -Z

        检查进程的安全上下文；  
    5) ls -Z  
        检查文件、目录的安全上下文；

**6.1 安全上下文格式**

      所有操作系统访问控制都是以关联的客体和主体的某种类型的访问控制属性为基础的。在SELinux中，访问控制属性叫做安全上下文。所有客体（文件、进程间通讯通道、套接字、网络主机等）和主体（进程）都有与其关联的安全上下文，一个安全上下文由三部分组成：用户、角色和类型标识符。常常用下面的格式指定或显示安全上下文：

      USER：ROLE：TYPE[LEVEL[：CATEGORY]]

      安全上下文中的用户和角色标识符除了对强制有一点约束之外对类型强制访问控制策略没什么影响，对于进程，用户和角色标识符显得更有意义，因为它们是用于控制类型和用户标识符的联合体，这样就会与Linux用户账号关联起来；然而，对于客体，用户和角色标识符几乎很少使用，为了规范管理，客体的角色常常是object\_r，客体的用户常常是创建客体的进程的用户标识符，它们在访问控制上没什么作用。

      标准Linux安全中的用户ID和安全上下文中的用户标识符之间的区别，就技术而论，它们是正交标识符，分别用于标准的和安全增强的访问控制机制，这两者之间的任一相互关联都是通过登陆进程按照规范严格规定的，而不是通过SELinux策略直接强制实施的。

**6.1.1 USER**

         1) user identity：类似Linux系统中的UID，提供身份识别，用来记录身份；安全上下文的一部分；  
         2) 三种常见的 user:

              • user\_u ：普通用户登录系统后的预设；  
             • system\_u ：开机过程中系统进程的预设；  
             • root ：root 登录后的预设；

        3) 在 targeted policy 中 users 不是很重要；  
        4) 在strict policy 中比较重要，所有预设的 SELinux Users 都是以 “\_u” 结尾的，root 除外。

**6.1.2 ROLE**

        1) 文件、目录和设备的role：通常是 object\_r；  
        2) 程序的role：通常是 system\_r；  
        3) 用户的role：targeted policy为system\_r； strict policy为sysadm\_r、staff\_r、user\_r；用户的role，类似系统中的GID，不同角色具备不同的的权限；用户可以具备多个role；但是同一时间内只能使用一个role；

        4) 使用基于RBAC(Roles Based Access Control) 的strict和mls策略中，用来存储角色信息

**6.1.3 TYPE**

        1) type：用来将主体(subject)和客体(object)划分为不同的组，给每个主体和系统中的客体定义了一个类型；为进程运行提供最低的权限环境；  
        2) 当一个类型与执行中的进程相关联时，其type也称为domain；  
        3) type是SElinux security context 中最重要的部位，是 SELinux Type Enforcement 的心脏，预设值以\_t结尾；

        LEVEL和CATEGORY：定义层次和分类，只用于mls策略中  
             • LEVEL：代表安全等级,目前已经定义的安全等级为s0-s15,等级越来越高  
             • CATEGORY：代表分类，目前已经定义的分类为c0-c1023

**6.2 对比SELinux和标准Linux的访问控制属性**

      在标准Linux中，主体的访问控制属性是与进程通过在内核中的进程结构关联的真实有效的用户和组ID，这些属性通过内核利用大量工具进行保护，包括登陆进程和setuid程序，对于客体（如文件），文件的inode包括一套访问模式位、文件用户和组ID。以前的访问控制基于读/写/执行这三个控制位，文件所有者、文件所有者所属组、其他人各一套。

      在SELinux中，访问控制属性总是安全上下文三人组(用户：角色：类型)形式，所有客体和主体都有一个关联的安全上下文。需要特别指出的是，因为SELinux的主要访问控制特性是类型强制，安全上下文中的类型标识符决定了访问权。

      注意：SELinux是在标准Linux基础上增加了类型强制(TE: Type Enforcement)，这就意味着标准Linux和SELinux访问控制都必须满足先要能访问一个客体，例如：如果我们对某个文件有SELinux写入权限，但我们没有该文件的w许可，那么我们也不能写该文件。下表总结了标准Linux和SELinux之间访问控制属性的对比：

|  |  |  |
| --- | --- | --- |
|  | 标准Linux | SELInux |
| 进程安全属性 | 真实有效的用户和组ID | 安全上下文 |
| 客体安全属性 | 访问模式、文件用户和组ID | 安全上下文 |
| 访问控制基础 | 进程用户/组ID和文件的访问模式， 此访问模式基于文件的用户/组ID | 在进程类型和文件类型 之间允许的许可 |

**6.3 小结**

      1) 系统中每个文件、目录、网络端口等都被指定一个安全上下文，policy 则给出各安全上下文之间的作用规则。  
      2) SELinux根据policy及security context规则来决定存取行为是否可执行；  
      3) Subject(主体)：系统进程，比如/usr/sbin/httpd；  
      4) Object(客体)：被存取的项目，比如File、Directory、IP、Socket等；

**7. 类型强制(TE)访问控制**

     在SELinux中，所有访问都必须明确授权，SELinux默认不允许任何访问，不管Linux用户/组ID是什么。这就意味着在SELinux中，没有默认的超级用户了，与标准Linux中的root不一样，通过指定主体类型（即域）和客体类型使用allow规则授予访问权限，allow规则由四部分组成：

     • 源类型（Source type(s) ） 通常是尝试访问的进程的域类型  
     • 目标类型（Target type(s) ） 被进程访问的客体的类型  
     • 客体类别（Object class(es)） 指定允许访问的客体的类型  
     • 许可（Permission(s)） 象征目标类型允许源类型访问客体类型的访问种类  
     举例如下：

**[cpp]** [view plain](http://blog.csdn.net/myarrow/article/details/9856095/) [copy](http://blog.csdn.net/myarrow/article/details/9856095/)

allow user\_t bin\_t : file {read execute getattr};

     这个例子显示了TE allow规则的基础语法，这个规则包含了两个类型标识符：源类型(或主体类型或域)user\_t，目标类型(或客体类型)bin\_t。标识符file是定义在策略中的客体类别名称(在这里，表示一个普通的文件)，大括号中包括的许可是文件客体类别有效许可的一个子集，这个规则解释如下：

     拥有域类型user\_t的进程可以读/执行或获取具有bin\_t类型的文件客体的属性。

     SELinux allow规则如之前的例子在SELinux中实际上都是授予访问权的，真正的挑战是如何保证数以万计的访问正确授权，只授予必须的权限，实现尽可能的安全。

**7.1 标准Linux安全中的setuid程序**

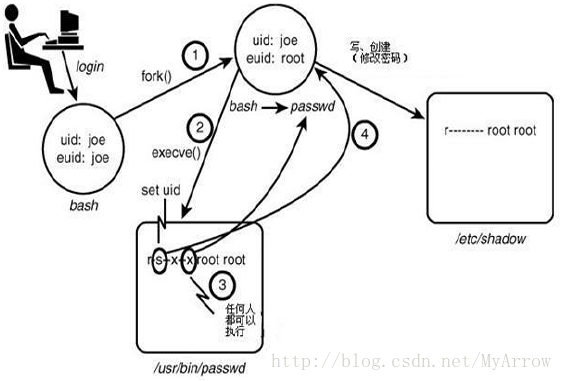
      精通用户joe想安全地修改现有的密码问题，Linux解决这个问题的方法是通过给passwd赋一个setuid值，使其执行时具有root权限，如果你在一个普通Linux系统上列出密码文件，你看到的会是：

**[cpp]** [view plain](http://blog.csdn.net/myarrow/article/details/9856095/) [copy](http://blog.csdn.net/myarrow/article/details/9856095/)

# ls -l /usr/bin/passwd

-rwsr-xr-x. 1 root root 41292 Sep  7  2012 /usr/bin/passwd

       这里注意两件事，第一个是在所有者权限的x位置被设置为s了，这就是所谓的setuid位，意思是任何执行这个文件的进程，它的有效UID（即用户ID）将会被改为文件所有者。这里，root是文件所有者，因此当执行密码程序时实际上将会以root用户的ID运行。其执行过程如下图所示：



       从上面的分析中可以看出，passwd以root权限的身份运行， 它可以访问系统的任何资源，这给系统带来了安全问题，其实它只需要访问shadow及其相关的文件就可以了。而且shadow只需要接受passwd的访问即可。这在标准Linux中是无法做到的，而TE(类型强制)可实现此功能。

**8. 基于角色的访问控制**

    SELinux也提供了一种基于角色的访问控制（RBAC），SELinux的RBAC特性是依靠类型强制建立的，SELinux中的访问控制主要是通过类型实现的，角色基于进程安全上下文中的角色标识符限制进程可以转变的类型，如此，策略编写器可以创建一个角色，允许它转变为一套域类型（假设类型强制规则允许转变），从而定义角色的限制。

**9. SELinux中的多级安全(Multi-Level Security)**

    类型强制(Type Enforcement)无疑是SELinux引入的最重要的强制访问控制（MAC）机制，然而，在某些情况下，主要是保密控制应用程序的一个子集，传统的多级安全（MLS）MAC与类型强制一起使用显得更有价值，在这些情况下，SELinux总是包括某种格式的MLS功能，MLS特性是可选的，在SELinux的两个MAC机制中，它通常不是最重要的那个，对大多数安全应用程序而言，包括许多非保密数据应用程序，类型强制是最适合的安全增强的机制，尽管如此，MLS对部分应用程序还是增强了安全性。

     在大多数SELinux策略中，敏感度(s0，s1，...)和范畴(c0，c1，...)使用通配名，将它留给用户空间程序和程序库，以指定有意义的用户名。(例如：s0可能与UNCLASSIFIED 关联，s1可能与SECRET关联)  
     为了支持MLS，安全上下文被扩展了，包括了安全级别，如：

**[cpp]** [view plain](http://blog.csdn.net/myarrow/article/details/9856095/) [copy](http://blog.csdn.net/myarrow/article/details/9856095/)

user:role:type:sensitivity[:category,...] [-sensitivity[:category,...]]

     例子如下所示：

**[cpp]** [view plain](http://blog.csdn.net/myarrow/article/details/9856095/) [copy](http://blog.csdn.net/myarrow/article/details/9856095/)

root@luohj-**virtual**-machine:~# ps -aZ

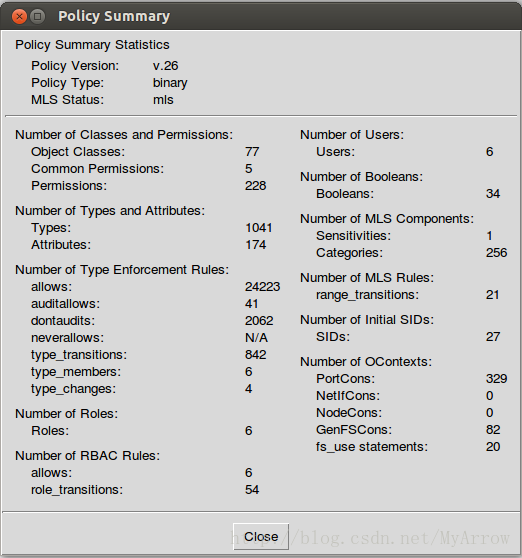
LABEL                                        PID   TTY   TIME    CMD

unconfined\_u:system\_r:insmod\_t:s0-s0:c0.c255 4940 pts/0 00:00:00 passwd

     注意MLS安全上下文至少必须有一个安全级别（它由单个敏感度和0个或多个范畴组成），但可以包括两个安全级别，这两个安全级别分别被叫做低（或进程趋势）和高（或进程间隙），如果高安全级别丢失，它会被认为与低安全级别的值是相同的（最常见的情况），实际上，对于客体和进程而言，低和高安全级别通常都是相同的，通常用于进程的级别范围被认为是受信任的主体（即进程信任降级信息）或多层客体，如一个目录，它又包括了不同安全级别的客体。为了使描述简单，假设所有的进程和客体都只有一个安全级别。

**10. 策略分析工具apol**

      apol(即analyze policy【分析策略】)工具是一个成熟的SELinux策略分析工具，它位于setools工具包中。使用它打开policy.xx文件即可分析所有的相关策略。xx为策略编译器(checkpolicy)的版本号。



**11. 小结**

      SELinux访问控制是基于与所有系统资源(包括进程)关联的安全上下文的，安全上下文包括三个组件：用户、角色和类型标识符。类型标识符是访问控制的主要基础。

      在SELinux中，访问控制的主要特性是类型强制，在主体(即进程)与客体之间通过指定allow规则(主体的类型【也叫做域类型】是源，客体的类型是目标)进行访问授权，访问被授予特定的客体类别，为每个客体类别设置细粒度的许可。

      类型强制的一个关键优势是它可以控制哪个程序可能运行在给定的域类型上，因此，它允许对单个程序进行访问控制（比起用户级的安全控制要安全得多了），使程序进入另一个域（即以一个给定的进程类型运行）叫做域转变，它是通过SELinux的allow规则紧密控制的，SELinux也允许通过type\_transition 文件使域转变自动发生。

      SELinux在访问控制安全上下文中不直接使用角色标识符，相反，所有的访问都是基于类型的，角色用于关联允许的域类型，这样可以设置类型强制允许的功能组合到一起，将用户作为一个角色进行认证。

      SELinux提供了一个可选的MLS访问控制机制，它提供了更多的访问限制，MLS特性依靠TE机制建立起来的，MLS扩展了安全上下文的内容，包括了一个当前的（或低）安全级别和一个可选的高安全级别。

参考：<http://www.centos.org/docs/5/html/Deployment_Guide-en-US/selg-overview.html>

<http://wenku.baidu.com/view/df89fe235901020207409c49.html>

<http://wenku.baidu.com/view/fed9abb569dc5022aaea002d.html>

[http://selinux.sourceforge.net](http://selinux.sourceforge.net/) Homepage for the SELinux community.

<http://www.nsa.gov/selinux/> Homepage for the NSA SELinux development team. Many resources are available in HTML and PDF formats. Although many of these links are not SELinux specific, some concepts may apply.

<http://fedora.redhat.com/docs/> Homepage for the Fedora documentation project, which contains Fedora Core specific materials that may be more timely, since the release cycle is much shorter.

<http://wenku.baidu.com/view/4d26594fc850ad02de804189.html>

原文地址<http://blog.csdn.net/myarrow/article/details/9856095/>

2. 在android里面，有两个类型，一种是文件，一种是进程。  
针对这两种类型，我们可以先来看看他们的不同。  
在android上面，adb shell之后进入手机，ps -Z可以查看当前进程所拥有的selinux的权限。  
  
举例：

[?](http://www.2cto.com/kf/201504/390742.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | LABEL                          USER     PID   PPID  NAME  u:r:init:s0                    root      1     0     /init  u:r:kernel:s0                  root      2     0     kthreadd  ...  u:r:kernel:s0                  root      258   2     irq/322-HPH\_R O  u:r:logd:s0                    logd      259   1     /system/bin/logd  u:r:healthd:s0                 root      260   1     /sbin/healthd  u:r:lmkd:s0                    root      261   1     /system/bin/lmkd  u:r:servicemanager:s0          system    262   1     /system/bin/servicemanager  u:r:vold:s0                    root      263   1     /system/bin/vold  u:r:surfaceflinger:s0          system    264   1     /system/bin/surfaceflinger  u:r:tctd:s0                    root      265   1     /system/bin/tctd  u:r:rfs\_access:s0              system    268   1     /system/bin/rfs\_access  u:r:tee:s0                     system    271   1     /system/bin/qseecomd  u:r:kernel:s0                  root      280   2     kworker/3:1H  u:r:kernel:s0                  root      290   2     kauditd  u:r:rmt\_storage:s0             nobody    291   1     /system/bin/rmt\_storage  u:r:shell:s0                   shell     292   1     /system/bin/sh  u:r:netd:s0                    root      295   1     /system/bin/netd  u:r:debuggerd:s0               root      296   1     /system/bin/debuggerd  u:r:tee:s0                     system    297   271   /system/bin/qseecomd |

在这个例子中，我们可以进行分析。  
在android中，只定义了一个user即为u. 另外，如果是进程的话，都会统一定义为r，如果是文件的话，会被定义为object\_r. 第三个是这个进程type，在andorid里面，定义了100多个type.按照目前我的理解，这个是进程所属的>类型。第四个是s0，这个是一个安全的等级。但是暂时还没有接触到配置这个的地方。  
  
另外就是文件，文件想要查看相关SELINUX权限的话，需要去执行ls -Z

[?](http://www.2cto.com/kf/201504/390742.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | drwxr-x--x root     sdcard\_r          u:object\_r:rootfs:s0 storage  drwx--x--x root     root              u:object\_r:tmpfs:s0 synthesis  dr-xr-xr-x root     root              u:object\_r:sysfs:s0 sys  drwxr-xr-x root     root              u:object\_r:system\_file:s0 system  drwxrwxr-x system   tctpersist          u:object\_r:tct\_persist\_file:s0 tctpersist  lrwxrwxrwx root     root              u:object\_r:rootfs:s0 tombstones -> /data/tombstones  -rw-r--r-- root     root              u:object\_r:rootfs:s0 ueventd.qcom.rc  -rw-r--r-- root     root              u:object\_r:rootfs:s0 ueventd.rc |

在这个例子中，结合上面的分析，我们知道了object\_r是代表的文件，u是android的唯一的用户，rootfs是这个文件所对应的类型，s0是一个安全的等级限制。  
  
3. 如何配置selinux  
首先，按照Google的官方文档：  
需要linux内核首先是支持selinux的，另外需要android的selinux的配置文件，也就是extern/sepolicy里面的内容。  
然后就是修改BoardConfig.mk  
Google的nexus的sepolicy的支持就放在了**device/lge/mako/sepolicy**  
首先会包含厂商定制的sepolicy的文件夹：**BOARD\_SEPOLICY\_DIRS**  
然后将规则添加到了sepolicy中：**BOARD\_SEPOLICY\_DIRS**  
  
这样的话，我们编译出来的image其实就是具有了selinux的功能。  
其实如果没有厂商定制的话，也是会编译到external/sepolicy的，这样的话，就是使用andriod所有默认的sepolicy（It defines the domains and types for the AOSP services and apps common to all devices. ）  
  
然后理解了这个之后，我们可以看到其实很多的厂商也是有自己的配置规则在device/\*\*\*/\*\*\*/sepolicy下面的.

4. selinux的配置规则：  
首先要了解sepolicy的结构：  
a. App进程 -> mac\_permissions.xml  
b. App数据文件 -> seapp\_contexts  
c. 系统文件 -> file\_contexts  
d. 系统属性 -> property\_contexts  
  
在te文件中，我们一般遇到的语法是这样的：  
rule\_name source\_type target\_type:class perm\_set  
解读为： 为source\_type设置一个rule\_name的规则，规则是对target\_type的class 进行 perm\_set的操作。  
  
然后是一些特殊的配置文件：  
a. external/sepolicy/attributes -> 所有定义的attributes都在这个文件  
b. external/sepolicy/access\_vectors -> 对应了每一个class可以被允许执行的命令  
c. external/sepolicy/roles -> [Android](http://www.2cto.com/kf/yidong/Android/)中只定义了一个role，名字就是r，将r和attribute domain关联起来  
d. external/sepolicy/users -> 其实是将user与roles进行了关联，设置了user的安全级别，s0为最低级是默认的级别，mls\_systemHigh是最高的级别  
e. external/sepolicy/security\_classes -> 指的是上文命令中的class，个人认为这个class的内容是指在android运行过程中，程序或者系统可能用到的操作的模块  
f. external/sepolicy/te\_macros -> 系统定义的宏全在te\_macros文件  
g. external/sepolicy/\*\*\*.te -> 一些配置的文件，包含了各种运行的规则  
  
另外，selinux有两种工作模式：  
“permissive”：所有操作都被允许（即没有MAC），但是如果有违反权限的话，会记录日志  
“enforcing”：所有操作都会进行权限检查  
  
最后，type的命令如下：  
type type\_id [alias alias\_id,] [attribute\_id] # 将type\_id（别名为alias）关联到attribute. 这样的话，方便用attribute来管理不同的type中包含相同的属性的部分。

class命令的格式为：  
class class\_name [ inherits common\_name ] { permission\_name ... }  
inherits表示继承了common定义的权限，然后自己额外实现了permission\_name的权限  
  
在te文件中常见的四种命名的规则：  
allow：赋予某项权限。  
allowaudit：audit含义就是记录某项操作。默认情况下是SE[Linux](http://www.2cto.com/os/linux/)只记录那些权限检查失败的操作。allowaudit则使得权限检查成功的操作也被记录。注意，allowaudit只是允许记录，它和赋予权限没关系。赋予权限必须且只能使  
用allow语句。  
dontaudit：对那些权限检查失败的操作不做记录。  
neverallow：前面讲过，用来检查安全策略文件中是否有违反该项规则的allow语句。如例子5所示：  
  
举例：

[?](http://www.2cto.com/kf/201504/390742.html)

|  |  |
| --- | --- |
| 1 | type init, domain; |

将init关联到domain，即将domain设置为init类型的属性

[?](http://www.2cto.com/kf/201504/390742.html)

|  |  |
| --- | --- |
| 1 | allow init unlabeled:filesystem mount; |

允许init类型对unlabeled类型的filesystem进行mount的操作

[?](http://www.2cto.com/kf/201504/390742.html)

|  |  |
| --- | --- |
| 1 | allow init fotad:unix\_stream\_socket { bind create }; |

允许init类型对fotad类型的unix\_stream\_socket 进行bind和create的操作

[?](http://www.2cto.com/kf/201504/390742.html)

|  |  |
| --- | --- |
| 1  2 | allow appdomain anr\_data\_file:dir search;  allow appdomain anr\_data\_file:file { open append }; |

首先appdomain是定义在te\_macros里面的一个宏，很多的app规则会使用类似app\_domain(shell)的命令将其添加进去  
这两句话的意思是：1. 允许app去对anr\_data\_file类型的目录进行查找的操作  
2. 允许app对anr\_data\_file类型的file进行打开和添加操作 其实就是规定了出现anr时候，app往/data/anr/里面写入的权限限制

[?](http://www.2cto.com/kf/201504/390742.html)

|  |  |
| --- | --- |
| 1 | neverallow { appdomain -unconfineddomain } kmem\_device:chr\_file { read write }; |

绝对不允许app(除了有unconfineddomain属性的app)对kmem\_device类型的字符设备进行读写的操作

[?](http://www.2cto.com/kf/201504/390742.html)

|  |  |
| --- | --- |
| 1 | neverallow { appdomain -unconfineddomain } self:capability2 \*; |

绝对不允许除了unconfineddomain以外的app对self类型的capability2进行任何的操作

[?](http://www.2cto.com/kf/201504/390742.html)

|  |  |
| --- | --- |
| 1 | type httpd\_user\_content\_t, file\_type, httpdcontent; |

声明一个httpd\_user\_content\_t的类型，具有file\_type和httpdcontent的属性

[?](http://www.2cto.com/kf/201504/390742.html)

|  |  |
| --- | --- |
| 1  2 | type httpd\_user\_content\_t;  typeattribute httpd\_user\_content\_t file\_type, httpdcontent; |

声明一个httpd\_user\_content\_t的类型  
定义httpd\_user\_content\_t具有file\_type, httpdcontent的属性

[?](http://www.2cto.com/kf/201504/390742.html)

|  |  |
| --- | --- |
| 1 | allow appdomain self:rawip\_socket create\_socket\_perms; |

所有可以设置类型的地方其实都可以设置为属性。  
比如这个例子，我们允许所有具有app属性的内容可以去对self属性的rawip\_socket进行create的操作

[?](http://www.2cto.com/kf/201504/390742.html)

|  |  |
| --- | --- |
| 1 | allow {user\_t domain} {bin\_t file\_type sbin\_t}:file execute ; |

允许user\_t和domain属性的类对bin\_t, file\_type, sbin\_t类型的file进行可执行的操作

[?](http://www.2cto.com/kf/201504/390742.html)

|  |  |
| --- | --- |
| 1  2 | allow user\_t user\_t:process signal;  allow user\_t self:process signal; |

这两条语句的表述其实是一致的，其实self指的是目标的类型和发起人的类型是一致的  
所以不能声明一个类型或者属性叫做self

[?](http://www.2cto.com/kf/201504/390742.html)

|  |  |
| --- | --- |
| 1 | allow user\_t bin\_t:file ~{ write setattr ioctl }; |

允许user\_t对bin\_t类型的file进行除了write setattr ioctl相关的操作

[?](http://www.2cto.com/kf/201504/390742.html)

|  |  |
| --- | --- |
| 1 | type\_transition system wifi\_data\_file:sock\_file system\_wpa\_socket; |

当一个类型为system的类别去进行wifi\_data\_file类型的sock\_file访问时，类型默认切换到system\_wpa\_socket  
  
如果下面这条语句想要执行成功  
type\_transition init\_t apache\_exec\_t:process apache\_t;  
至少首先声明下面的三条规则：  
allow init\_t apache\_exec\_t:file execute;  
allow init\_t apache\_t:process transition;  
allow apache\_t apache\_exec\_t:file entrypoint;  
  
type\_transition和type\_change的语法规则是一样的， type\_change规则的影响不会在内核中生效，而是依赖于用户空间应用程序，如login或sshd

链接<http://www.2cto.com/kf/201504/390742.html>

开发android项目时，需要在init.rc中加入我们自己的native service xxx；但是在android 5.x系统上添加服务后发现服务没有正常启动，log中会报avc: denied错误。被SELinux拒绝权限了。所以我们就需要添加权限了。

与之前android版本一样，修改init rc文件添加我们需要的服务

service xxx/system/vendor/bin/xxx

class main

user root

group root

好啦！这样开机就可以执行服务了，但是不能正常启动，因为没有权限....,所以就要添加相应的权限策略使服务正常启动。

1.查看文件的安全上下文

先要用ls -Z命令查看安全上下文：

ls -Z

-rwxr-xr-x root shell u:object\_r:system\_file:s0 xxx

type是system\_file并不是所期望的xxx\_exec，这就要修改file\_contexts，将它进行labeling。

file\_contexts策略文件中添加：

/system/vendor/bin/xxx u:object\_r:xxx\_exec:s0

添加xxx.te策略文件：

type xxxr, domain;

type xxx\_exec, exec\_type, file\_type;

修改了sepolicy策略文件就需要重新编译sepolicy：

mmm external/sepolicy/

因为编译生成的目标在root根文件系统中，需要重新编译boot.img:

make bootimage

重新烧录boot.img：

dd if=/tmp/boot.img of=/dev/block/by-name/boot

这样重新编译，再用ls -Z查看的时候，type就是xxx\_exec了。

注意：进行labeling的文件不能使用adb push 到系统中，否则可能用ls -Z查看时没有打上预期的标签。

2.根据log来写出需要的权限

<36>[ 28.725369] type=1400 audit(1481091758.000:6): avc: denied { execute\_no\_trans } for pid=2861 comm="init" path="/system/vendor/bin/xxx" dev="mmcblk0p7" ino=1984 scontext=u:r:init:s0 tcontext=u:object\_r:xxx\_exec:s0 tclass=file permissive=0

从日志看是没有权限execute\_no\_trans，所以在xxx.te策略文件添加：

init\_daemon\_domain(xxx)

init\_daemon\_domain这是个宏，可以在te\_macros文件中找到声明，里面就有trans相关，当然不用宏自己编写也可以，那样要写多条allow语句就太繁琐了。

<36>[ 14.175149] type=1400 audit(1481092361.220:3): avc: denied { call } for pid=1658 comm="xxx" scontext=u:r:xxx:s0 tcontext=u:r:servicemanager:s0 tclass=binder permissive=0

从日志看是没有权限进行binder call，不要急着写allow，有宏可以快速搞定，在xxx.te文件添加：

binder\_use(xxx)

E/SELinux ( 1531): avc: denied { add } for service=xxx.service scontext=u:r:xxx:s0 tcontext=u:object\_r:default\_android\_service:s0 tclass=service\_manager

根据日志可以写出相应的allow语句:

allow xxx default\_android\_service:service\_manager add;

但是这样sepolicy编译会报错：

libsepol.check\_assertions: 1 neverallow failures occurred

Error while expanding policy

default\_android\_service会导致权限放大，所以要写在service.te文件中添加：

type xxx\_service, service\_manager\_type;

相应的allow语句改成这样:

allow xxx xxx\_service:service\_manager add;

service\_contexts文件中添加：

xxx.service u:object\_r:xxx\_service:s0

其中的xxx.service就是Native Service代码中defaultServiceManager()->addService的名字。

至此服务可以正常启动了.

完整的xxx.te：

# xxx - xxx daemontype

xxx, domain;

type xxx\_exec, exec\_type, file\_type;

init\_daemon\_domain(xxx)

binder\_use(xxx)

allow xxx xxx\_service:service\_manager add;

链接<http://blog.csdn.net/dpppppp/article/details/53513389>

[例子1]

/\*

from external/sepolicy/netd.te

下面这条SELinux语句表示 允许（allow ）netd域（domain）中的进程 ”写（write）“

类型为proc的文件

注意，SELinux中安全策略文件有自己的一套语法格式，下文我们将详细介绍它

\*/

allow netd proc:file write

如果没有在netd.te中使用上例中的权限配置allow语句，则netd就无法往/proc目录下得任何文件中写数据，即使netd具有root权限。

显然，MAC比DAC在权限管理这一块要复杂，要严格，要细致得多。

那么，关于DAC和MAC，此处笔者总结了几个知识点：

Linux系统先做DAC检查。如果没有通过DAC权限检查，则操作直接失败。通过DAC检查之后，再做MAC权限检查。

SELinux中也有用户的概念，但它和Linux中原有的user概念不是同一个东西。什么意思呢？比如，Linux中的超级用户root在SELinux中可能就是一个没权限，没地位，打打酱油的”路人甲“。当然，这一切都由SELinux安全策略的制定者来决定。

通过上述内容，读者应该能感觉到，在SELinux中，安全策略文件是最重要的。确实如此。事实上，对本文的读者而言，学习SELinux的终极目标应该是：

看懂现有的安全策略文件。

编写符合特定需求的安全策略文件。

前面也曾提到，SELinux有自己的一套规则来编写安全策略文件，这套规则被称之为SELinux Policy语言。它是掌握SELinux的重点。

2. SELinux Policy语言介绍

Linux中有两种东西，一种死的（Inactive），一种活的（Active）。死的东西就是文件（Linux哲学，万物皆文件。注意，万不可狭义解释为File），而活的东西就是进程。此处的“死”和“活”是一种比喻，映射到软件层面的意思是：进程能发起动作，例如它能打开文件并操作它。而文件只能被进程操作。

SELinux中，每种东西都会被赋予一个安全属性，官方说法叫Security Context。Security Context（以后用SContext表示）是一个字符串，主要由三部分组成。例如SEAndroid中，进程的SContext可通过ps -Z命令查看，如图1所示：

图1 Nexus 7 ps -Z结果图

图1中最左边的那一列是进程的SContext，以第一个进程/system/bin/logwrapper的SContext为例，其值为u:r:init:s0，其中：

u为user的意思。SEAndroid中定义了一个SELinux用户，值为u。

r为role的意思。role是角色之意，它是SELinux中一种比较高层次，更方便的权限管理思路，即Role Based Access Control（基于角色的访问控制，简称为RBAC）。简单点说，一个u可以属于多个role，不同的role具有不同的权限。RBAC我们到最后再讨论。

init，代表该进程所属的Domain为init。MAC的基础管理思路其实不是针对上面的RBAC，而是所谓的Type Enforcement Accesc Control（简称TEAC，一般用TE表示）。对进程来说，Type就是Domain。比如init这个Domain有什么权限，都需要通过[例子1]中allow语句来说明。

S0和SELinux为了满足军用和教育行业而设计的Multi-Level Security（MLS）机制有关。简单点说，MLS将系统的进程和文件进行了分级，不同级别的资源需要对应级别的进程才能访问。后文还将详细介绍MLS。

再来看文件的SContext，读者可通过ls -Z来查看，如图2所示：

图2 Nexus 7 ls -Z结果图

图2中，倒数第二列所示为Nexus 7根目录下几个文件和目录的SContext信息，以第一行root目录为例，其信息为u:object\_r:rootfs:s0：

u：同样是user之意，它代表创建这个文件的SELinux user。

object\_r：文件是死的东西，它没法扮演角色，所以在SELinux中，死的东西都用object\_r来表示它的role。

rootfs：死的东西的Type，和进程的Domain其实是一个意思。它表示root目录对应的Type是rootfs。

s0：MLS的级别。

根据SELinux规范，完整的SContext字符串为：

user:role:type[:range]

注意，方括号中的内容表示可选项。s0属于range中的一部分。下文再详细介绍range所代表的Security Level相关的知识。

看，SContext的核心其实是前三个部分：user:role:type。

刚才说了，MAC基本管理单位是TEAC（Type Enforcement Accesc Control），然后是高一级别的Role Based Accesc Control。RBAC是基于TE的，而TE也是SELinux中最主要的部分。

下面来看看TE。

2.1 TE介绍

在例子1中，大家已经见过TE的allow语句了，再来细致研究下它：

[例子2]

allow netd proc:file write

这条语句的语法为：

allow：TE的allow语句，表示授权。除了allow之外，还有allowaudit、dontaudit、neverallow等。

netd：source type。也叫subject，domain。

proc：target type。它代表其后的file所对应的Type。

file：代表Object Class。它代表能够给subject操作的一类东西。例如File、Dir、socket等。在Android系统中，有一个其他Linux系统没有的Object Class，那就是Binder。

write：在该类Object Class中所定义的操作。

根据SELinux规范，完整的allow相关的语句格式为：

rule\_name source\_type target\_type : class perm\_set

我们直接来看几个实例：

[例子3]

//SEAndroid中的安全策略文件policy.conf

#允许zygote域中的进程向init type的进程（Object Class为process）发送sigchld信号

allow zygote init:process sigchld;

#允许zygote域中的进程search或getattr类型为appdomain的目录。注意，多个perm\_set

#可用{}括起来

allow zygote appdomain:dir { getattr search };

#来个复杂点的：

#source\_type为unconfineddomain target\_type为一组type，由

#{ fs\_type dev\_type file\_type }构成。object\_class也包含两个，为{ chr\_file file }

#perm\_set语法比较奇特，前面有一个~号。它表示除了{entrypoint relabelto}之外，{chr\_file #file}这两个object\_class所拥有的其他操作

allow unconfineddomain {fs\_type dev\_type file\_type}:{ chr\_file file } \

~{entrypoint relabelto};

#特殊符号除了~外，还有-号和\*号，其中：

# 1）：-号表示去除某项内容。

# 2）：\*号表示所有内容。

#下面这条语句中，source\_type为属于appdomain，但不属于unconfinedomain的进程。

#而\*表示所有和capability2相关的权限

#neverallow：表示绝不允许。

neverallow { appdomain -unconfineddomain } self:capability2 \*;

特别注意，前面曾提到说权限必须显示声明，没有声明的话默认就没有权限。那neverallow语句就没必要存在了。因为”无权限“是不需要声明的。确实如此，neverallow语句的作用只是在生成安全策略文件时进行检查，判断是否有违反neverallow语句的allow语句。例如，笔者修改shell.te中一个语句后，生成安全策略文件时就检测到了冲突，如图3所示：

图3 neverallow的作用

如图3所示，笔者修改shell.te后，意外导致了一条allow语句与neverallow语句冲突，从而生成安全策略文件失败。

下面我们来看上述allow语句中所涉及到的object class和perm set。

（1） Object class和Perm Set

Object class很难用语言说清楚它到底是怎么定义的，所以笔者也不废话，直接告诉大家常见的Object class有哪些。见下面的SEPolicy示例：

[external/sepolicy/security\_classes示例]

.......

#此文件定义了Android平台中支持的Object class

#根据SELinux规范，Object Class类型由class关键字申明

# file-related classes

classfilesystem

classfile #代表普通文件

class dir #代表目录

class fd #代表文件描述符

class lnk\_file #代表链接文件

class chr\_file #代表字符设备文件

......

# network-related classes

class socket #socket

class tcp\_socket

class udp\_socket

......

class binder #Android平台特有的binder

class zygote #Android平台特有的zygote

#Android平台特有的属性服务。注意其后的userspace这个词

class property\_service # userspace和用户空间中的SELinux权限检查有关，下文再解释

上述示例展示了SEAndroid中Object Class的定义，其中：

Object Class需要通过class语句申明。这些申明一般放在一个叫security\_class的文件中。

另外，这些class和kernel中相关模块紧密结合。

据说：在kernel编译时会根据security\_class文件生成对应的头文件。从这里可以看出，SELinux需要根据发行平台来做相应修改。同时可以看出，该文件一般也不需要我们去修改。

再来看Perm set。Perm set指得是某种Object class所拥有的操作。以file这种Object class而言，其拥有的Perm set就包括read，write，open，create,execute等。

和Object class一样，SELinux或SEAndroid所支持的Perm set也需要声明，来看下面的例子：

[external/sepolicy/access\_vectors]

#SELinux规范中，定义perm set有两种方式，一种是使用下面的common命令

#其格式为：common common\_name { permission\_name ... } common定义的perm set能

#被另外一种perm set命令class所继承

#以下是Android平台中，file对应的权限（perm set）。其大部分权限读者能猜出是干什么的。

#有一些权限需要结合文后的参考文献来学习

common file {

ioctl read write create getattr setattr lock relabelfrom relabelto

append unlink link rename execute swapon quotaon mounton }

#除了common外，还有一种class命令也可定义perm set，如下面的例子：

#class命令的完整格式是：

#class class\_name [ inherits common\_name ] { permission\_name ... }

#inherits表示继承了某个common定义的权限 注意，class命令定义的权限其实针对得就是

#某个object class。它不能被其他class继承

class dir inherits file {

add\_name remove\_name reparent search rmdir open audit\_access execmod

}

#来看SEAndroid中的binder和property\_service这两个Object class定义了哪些操作权限

classbinder {

impersonate call set\_context\_mgr transfer }

classproperty\_service { set }

提示：Object class和Perm set的具体内容（SELinux中其实叫Access Vector）都和Linux系统/Android系统密切相关。所以，从知识链的角度来看，Linux编程基础很重要。

（2） type，attribute和allow等

现在再来看type的定义，和type相关的命令主要有三个，如下面的例子所示：

[external/sepolicy相关文件]

#type命令的完整格式为：type type\_id [alias alias\_id,] [attribute\_id]

#其中，方括号中的内容为可选。alias指定了type的别名，可以指定多个别名。

#下面这个例子定义了一个名为shell的type，它和一个名为domain的属性（attribute）关联

type shell, domain; #本例来自shell.te，注意，可以关联多个attribute

#属性由attribute关键字定义，如attributes文件中定义的SEAndroid使用的属性有：

attribute domain

attribute file\_type

#可以在定义type的时候，直接将其和某个attribute关联，也可以单独通过

#typeattribue将某个type和某个或多个attribute关联起来，如下面这个例子

#将前面定义的system类型和mlstrustedsubject属性关联了起来

typeattribute system mlstrustedsubject

特别注意：对初学者而言，attribute和type的关系最难理解，因为“attribute”这个关键词实在是没取好名字，很容易产生误解：

实际上，type和attribute位于同一个命名空间，即不能用type命令和attribute命令定义相同名字的东西。

其实，attribute真正的意思应该是类似type（或domain） group这样的概念。比如，将type A和attribute B关联起来，就是说type A属于group B中的一员。

使用attribute有什么好处呢？一般而言，系统会定义数十或数百个Type，每个Type都需要通过allow语句来设置相应的权限，这样我们的安全策略文件编起来就会非常麻烦。有了attribute之后呢，我们可以将这些Type与某个attribute关联起来，然后用一个allow语句，直接将source\_type设置为这个attribute就可以了：

这也正是type和attribute位于同一命名空间的原因。

这种做法实际上只是减轻了TE文件编写者的烦恼，安全策略文件在编译时会将attribute拓展为其包含的type。如例子4所示：

[例子4]

#定义两个type，分别是A\_t和B\_t，它们都管理到attribute\_test

type A\_t attribute\_test;

type B\_t attribute\_test;

#写一个allow语句，直接针对attribute\_test

allow attribute\_test C\_t:file {read write};

#上面这个allow语句在编译后的安全策略文件中会被如下两条语句替代：

allow A\_t C\_t:file {read write};

allow B\_t C\_t:file {read write};

前面讲过，TE的完整格式为：

rule\_name source\_type target\_type : class perm\_set

所以，attribute可以出现在source\_type中，也可以出现在target\_type中。

提示：一般而言，定义type的时候，都会在名字后添加一个\_t后缀，例如type system\_t。而定义attribute的时候不会添加任何后缀。但是Android平台没使用这个约定俗成的做法。不过没关系，SEAndroid中定义的attribute都在external/sepolicy/attribute这个文件中，如果分不清是type还是attribute，则可以查看这个文件中定义了哪些attribute。

最后我们来看看TE中的rule\_name，一共有四种：

allow：赋予某项权限。

allowaudit：audit含义就是记录某项操作。默认情况下是SELinux只记录那些权限检查失败的操作。allowaudit则使得权限检查成功的操作也被记录。注意，allowaudit只是允许记录，它和赋予权限没关系。赋予权限必须且只能使用allow语句。

dontaudit：对那些权限检查失败的操作不做记录。

neverallow：前面讲过，用来检查安全策略文件中是否有违反该项规则的allow语句。如例子5所示：

[例子5]

#来自external/sepolicy/netd.te文件

#永远不允许netd域中的进程 读写 dev\_type类型的 块设备文件（Object class为blk\_file）

neverallow netd dev\_type:blk\_file { read write }

（3） RBAC和constrain

绝大多数情况下，SELinux的安全配置策略需要我们编写各种各样的xx.te文件。由前文可知，.te文件内部应该包含包含了各种allow，type等语句了。这些都是TEAC，属于SELinux MAC中的核心组成部分。

在TEAC之上，SELiunx还有一种基于Role的安全策略，也就是RBAC。RBAC到底是如何实施相关的权限控制呢？我们先来看SEAndroid中Role和User的定义。

[external/sepolicy/roles]

#Android中只定义了一个role，名字就是r

role r;

#将上面定义的r和attribute domain关联起来

role r types domain;

再来看user的定义。

[external/sepolicy/users]

#支持MLS的user定义格式为：

#user seuser\_id roles role\_id level mls\_level range mls\_range;

#不支持MLS user定义格式为：

#user seuser\_id roles role\_id;

#SEAndroid使用了支持MLS的格式。下面定义的这个user u，将和role r关联。

#注意，一个user可以和多个role关联。

#level之后的是该user具有的安全级别。s0为最低级，也就是默认的级别，mls\_systemHigh

#为u所能获得的最高安全级别（security level）。此处暂且不表MLS

user u roles { r } level s0 range s0 - mls\_systemhigh;

那么，Roles和User中有什么样的权限控制呢？

1）首先，我们应该允许从一个role切换（SELinux用Transition表达切换之意）到另外一个role，例如：

#注意，关键字也是allow，但它和前面TE中的allow实际上不是一种东西

#下面这个allow允许from\_role\_id切换到to\_role\_id

allow from\_role\_id to\_role\_id;

2) 角色之间的关系。SELinux中，Role和Role之间的关系和公司中的管理人员的层级关系类似，例如：

#dominance语句属于deprecated语句，MLS中有新的定义层级相关的语句。不过此处要介绍的是

#selinux中的层级关系

#下面这句话表示super\_r dominate（统治，关键词dom） sysadm\_r和secadm\_r这两个角色

#反过来说，sysadm\_r和secadm\_r dominate by (被统治，关键词domby) super\_r

#从type的角度来看，super\_r将自动继承sysadm\_r和secadm\_r所关联的type（或attribute）

dominance { role super\_r {role sysadm\_r; role secadm\_r; }

3）其他内容，由于SEAndroid没有使用，此处不表。读者可阅读后面的参考文献。

话说回来，怎么实现基于Role或User的权限控制呢？SELinux提供了一个新的关键词，叫constrain，来看下面这个例子：

[例子6]

#constrain标准格式为:

# constrain object\_class\_set perm\_set expression ;

#下面这句话表示只有source和target的user相同，并且role也相同，才允许

#write object\_class为file的东东

constrain file write (u1 == u2 and r1 == r2) ;

前面已经介绍过object\_class和perm\_set了，此处就不再赘述。constrain中最关键的是experssion，它包含如下关键词:

u1,r1,t1：代表source的user，role和type。

u2,r2,t2：代表target的user,role和type。

==和!=：==表示相等或属于，!=表示不等或不属于。对于u,r来说，==和!=表示相等或不等，而当诸如t1“==或!=”某个attribute时，表示源type属于或不属于这个attribute。

dom,domby,incomp,eq：仅针对role，表示统治，被统治，没关系和相同（和==一样）

关于constrain，再补充几个知识点：

SEAndroid中没有使用constrain，而是用了MLS中的mlsconstrain。所以下文将详细介绍它。

constrain是对TEAC的加强。因为TEAC仅针对Type或Domain，没有针对user和role的，所以constrain在TEAC的基础上，进一步加强了权限控制。在实际使用过程中，SELinux进行权限检查时，先检查TE是否满足条件，然后再检查constrain是否也满足条件。二者都通过了，权限才能满足。

关于RBAC和constrain，我们就介绍到此。

提示：笔者花了很长时间来理解RBAC和constrain到底是想要干什么。其实这玩意很简单。因为TE是Type Enforcement，没user和role毛事，而RBAC则可通过constrain语句来在user和role上再加一些限制。当然，constrain也可以对type进行限制。如此而已！

2.2 Labeling介绍

前面陆陆续续讲了些SELinux中最常见的东西。不过细心的人可能会问这样一个问题：这些SContext最开始是怎么赋给这些死的和活的东西的？Good Question！

提示：SELinux中，设置或分配SContext给进程或文件的工作叫Security Labeling，土语叫打标签。

（1） sid和sid\_context

这个问题的回答嘛，其实也蛮简单。Android系统启动后（其他Linux发行版类似），init进程会将一个编译完的安全策略文件传递给kernel以初始化kernel中的SELinux相关模块（姑且用Linux Security Module:LSM来表示它把），然后LSM可根据其中的信息给相关Object打标签。

提示：上述说法略有不准，先且表述如此。

LSM初始化时所需要的信息以及SContext信息保存在两个特殊的文件中，以Android为例，它们分别是：

initial\_sids：定义了LSM初始化时相关的信息。SID是SELinux中一个概念，全称是Security Identifier。SID其实类似SContext的key值。因为在实际运行时，如果老是去比较字符串（还记得吗，SContext是字符串）会严重影响效率。所以SELinux会用SID来匹配某个SContext。

initial\_sid\_context：为这些SID设置最初的SContext信息。

来看这两个文件的内容：

[external/sepolicy/initial\_sids和initial\_sid\_context]

#先看initial\_sids

sid kernel #sid是关键词，用于定义一个sid

sid security

sid unlabeled

sid fs

sid file

sid file\_labels

sid init

......

#再来看initial\_sid\_context

sid kernel u:r:kernel:s0 #将initial\_sids中定义的sid和初始的SContext关联起来

sid security u:object\_r:kernel:s0

sid unlabeled u:object\_r:unlabeled:s0

sid fs u:object\_r:labeledfs:s0

sid file u:object\_r:unlabeled:s0

sid file\_labels u:object\_r:unlabeled:s0

sid init u:object\_r:unlabeled:s0

提示：sid的细节需要查看LSM的实现。此处不拟深究它。另外，这两个文件也是和Kernel紧密相关的，所以一般不用修改它们。

（2） Domain/Type Transition和宏

SEAndroid中，init进程的SContext为u:r:init:s0，而init创建的子进程显然不会也不可能拥有和init进程一样的SContext（否则根据TE，这些子进程也就在MAC层面上有了和init一样的权限）。那么这些子进程的SContext是怎么被打上和其父进程不一样的SContext呢？

SELinux中，上述问题被称为Domain Transtition，即某个进程的Domain切换到一个更合适的Domain中去。Domain Transition也是需要我们在安全策略文件中来配置的，而且有相关的关键词，来看例子7。

[例子7-1]

#先要使用type\_transition语句告诉SELinux

#type\_transition的完整格式为：

# type\_transition source\_type target\_type : class default\_type;

#对Domain Transition而言有如下例子：

type\_transition init\_t apache\_exec\_t : process apache\_t;

上面这个例子的解释如下，请读者务必仔细：

当init\_t Domain中的进程执行type为apache\_exec\_t类型的可执行文件（fork并execv）时，其class（此处是process）所属Domain（对process而言，肯定是指Domain）需要切换到apache\_t域。

明白了吗？要做DT，肯定需要先fork一个子进程，然后通过execv打开一个新的可执行文件，从而进入变成那个可执行文件对应的活物！所以，在type\_transition语句中，target\_type往往是那个可执行文件（死物）的type。default\_type则表示execv执行后，这个活物默认的Domain。另外，对DT来说，class一定会是process。

请注意，DT属于Labeling一部分，但这个事情还没完。因为打标签也需要相关权限。所以，上述type\_transition不过是开了一个头而已，要真正实施成功这个DT，还需要下面至少三个allow语句配合：

[例子7-2]

#首先，你得让init\_t域中的进程能够执行type为apache\_exec\_t的文件

allow init\_t apache\_exec\_t : file execute;

#然后，你还得告诉SELiux，允许init\_t做DT切换以进入apache\_t域

allow init\_t apache\_t : process transition;

#最后，你还得告诉SELinux，切换入口（对应为entrypoint权限）为执行apache\_exec\_t类型

#的文件

allow apache\_t apache\_exec\_t : file entrypoint;

为什么会需要上述多达三个权限呢？这是因为在Kernel中，从fork到execv一共设置了三处Security检查点，所以需要三个权限。

提示：读者不必纠结这个了，按照规范做就完了。不过...，这导致我们写TE文件时候会比较麻烦啊！

确实比较麻烦，不过SELinux支持宏，这样我们可以定义一个宏语句把上述4个步骤全部包含进来。在SEAndroid中，系统定义的宏全在te\_macros文件中，其中和DT相关的宏定义如下：

[external/sepolicy/te\_macros]

#定义domain\_trans宏。$1,$2等等代表宏的第一个，第二个....参数

define(`domain\_trans', `

# SEAndroid在上述三个最小权限上，还添加了自己的一些权限

allow $1 $2:file { getattr open read execute };

allow $1 $3:process transition;

allow $3 $2:file { entrypoint read execute };

allow $3 $1:process sigchld;

dontaudit $1 $3:process noatsecure;

allow $1 $3:process { siginh rlimitinh };

')

#定义domain\_auto\_trans宏，这个宏才是我们在te中直接使用的

#以例子7而言，该宏的用法是：

#domain\_auto\_trans(init\_t,apache\_exec\_t, apache\_t)

define(`domain\_auto\_trans', `

# 先allow相关权限

domain\_trans($1,$2,$3)

# 然后设置type\_transition

type\_transition $1 $2:process $3;

')

在external/sepolicy/init\_shell.te中就有上述宏的用法：

./init\_shell.te:4:domain\_auto\_trans(init, shell\_exec, init\_shell)

除了DT外，还有针对Type的Transition。举个例子，假设目录A的SContext为u:r:dir\_a，那么默认情况下在该目录下创建的文件都具有u:r:dir\_a这个SContext。所以我们也要针对死得东西进行打标签。

和DT类似，TT的语句也是type\_transition，而且要顺利完成Transition，也需要申请相关权限。废话不再多说，我们直接看te\_macros是怎么定义TT所需要的宏的：

[external/sepolicy/te\_macros]

# 定义file\_type\_trans(domain, dir\_type, file\_type)宏

#

define(`file\_type\_trans', `

# ra\_dir\_perms是一个宏，由global\_macros文件定义，其值为：

#define(`ra\_dir\_perms', `{ r\_dir\_perms add\_name write }')

allow $1 $2:dir ra\_dir\_perms;

# create\_file\_perms也是一个宏，定义在global\_macros文件中，其值为：

# define(`create\_file\_perms', `{ create setattr rw\_file\_perms

# link\_file\_perms }')

#而r\_dir\_perms=define(`r\_dir\_perms', `{ open getattr read search ioctl }

allow $1 $3:notdevfile\_class\_set create\_file\_perms;

allow $1 $3:dir create\_dir\_perms;

')

# 定义file\_type\_auto\_trans(domain, dir\_type, file\_type)宏

#该宏的含义是：当domain域中的进程在某个Type为dir\_type的目录中创建文件时，该文件的

#SContext应该是file\_type

define(`file\_type\_auto\_trans', `

file\_type\_trans($1, $2, $3)

type\_transition $1 $2:dir $3;

#notdevfile\_class\_set也是一个宏，由global\_macros文件定义，其值为

# define(`notdevfile\_class\_set', `{ file lnk\_file sock\_file fifo\_file }')

type\_transition $1 $2:notdevfile\_class\_set $3;

')

WoW，SEAndroid太这两个宏定义太复杂了，来看看官方文档中的最小声明是什么：

[例子8]

type\_transition acct\_t var\_log\_t:file wtmp\_t;

allow acct\_t var\_log\_t:dir { read getattr lock search ioctl

add\_name remove\_name write };

allow acct\_t wtmp\_t:file { create open getattr setattr read

write append rename link unlink ioctl lock };

在SEAndroid的app.te中，有如下TT设置：

./app.te:86:file\_type\_auto\_trans(appdomain, download\_file, download\_file)

DT和TT就介绍到这，翻来覆去就这么点东西，多看几遍就“柜”（用柜字，打一成语，参考2014年中国首次猜谜大会）了

=======未完，待续========

版权声明：本文为博主原创文章，未经博主允许不得转载。

原文地址: http://blog.csdn.net/innost/article/details/19299937