实验一：

**Linux 系统文件和目录权限设置与辨识 setuid 程序 uid 差异**

班级：1703202

学号：1170300421

姓名：贺宗磊

**要求：**

1. 设想一种场景需要进行普通用户和root用户切换，设计程序实现euid的安全管理

配合第3章 完成进程中euid的切换，实现root权限临时性和永久性管理，加强程序的安全性

说明：1学时，不分组实现

1. 搭建安全的沙盒环境，在沙盒环境中提供必须的常见工具，并提供程序验证沙盒环境的安全性

配合第3章 实现系统中的虚拟化限制方法，实现安全的系统加固，测试虚拟化空间的加固程度

说明：3学时，2人一组，分组实现

* 1. **Linux系统文件和目录权限设置与辨识setuid程序uid差异**

1. 设计并实现不同用户对不同类文件的r、w、x权限:

（1）查看系统文件的权限设置

a)查看/etc/passwd文件和/etc/bin/passwd文件的权限设置，并分析其权限为什么这么设置；







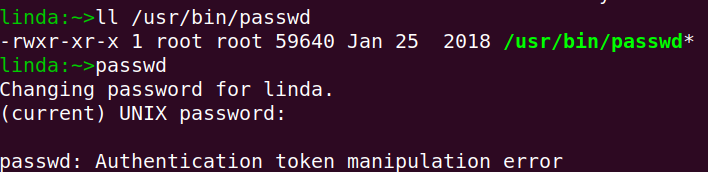
r-read-读权限w-write-写权限x-execute-执行权限

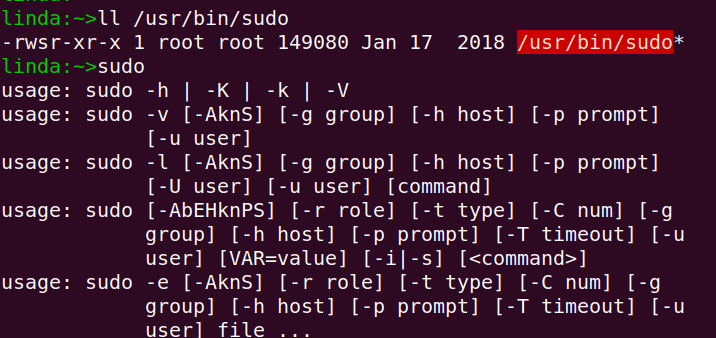
r=4，w=2，x=1

例如，rwx属性则可以表示为4+2+1=7；rw-属性则可以表示为4+2=6。

权限是三组，每组是”rwx”格式，三组分别代表文件所有者、所在组、其他组的所具有的权限。所有者具有读写权限，所有组和其他组只具有读权限，这样写的目的是保护文件不被恶意篡改，只能被所有者修改。

b)找到2个设置了setuid位的可执行程序，该程序的功能，该程序如果不设置setuid位是否能够达到相应的功能，





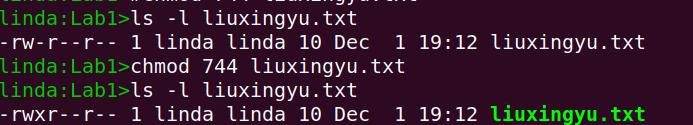
/usr/bin/passwd 文件是一个可以为用户添加、更改密码的命令。为了让每个用户都可以运行这个命令，所以设置了 s 位来提供 root 权限。但是如果需要这个命令真正生效，那么对于同组用户与其它用户，都必须拥有 x 权限才能真正使 s 位生效。如果不设置 s 位的话，用户将不可以添加、更改密码。

PS：passwd命令有一个特殊的权限标记s ，存在于文件所有者的权限位上。这是一类特殊的权限SetUID ，当一个具有执行权限的文件设置SetUID权限后，用户执行这个文件时将以文件所有者的身份执行。也就是说当普通用户使用passwd更改自己密码的时候，实际在以passwd命令所有者root的身份在执行，root当然可以将密码写入/etc/shadow文件，命令执行完成后该身份也随之消失。

Sudo 程序是用来登录 root 用户的程序，如果不设置 s 位的话，那么普通用户就不能运行这个程序，也就不能登录 root 用户了。

2.设置文件或目录权限

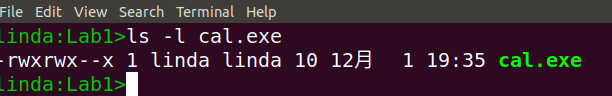
a)用户A具有文本文件”liuxingyu.txt”，该用户允许别人下载；



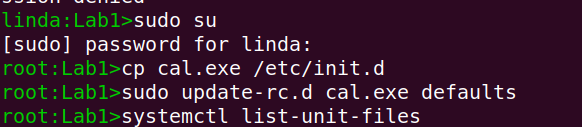
只要其它用户拥有了读权限，那么其就有能力进行下载操作

b)用户A编译了一个可执行文件”cal.exe”，该用户想在系统启动时运行

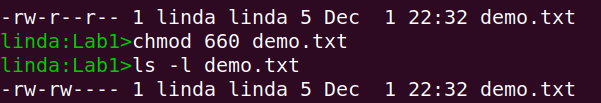
利用chmod a+x cal.exe设置后



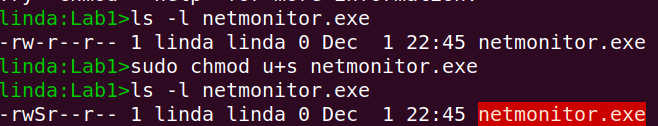
添加到系统启动项



c)用户A有起草了文件”demo.txt”，想让同组的用户帮其修改文件

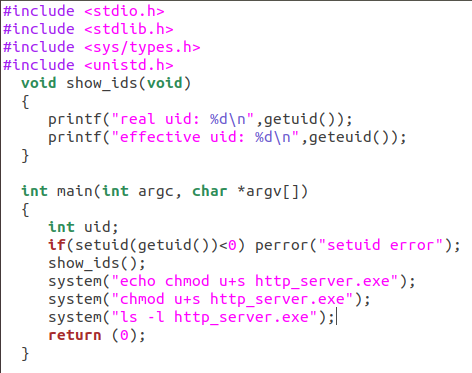


d)一个root用户拥有的网络服务程序”netmonitor.exe”，需要设置setuid位才能完成其功能。



二、一些可执行程序运行时需要系统管理员权限，在 UNIX 中可以利用 setuid 位实现其功能

1.设想一种场景，比如提供 http 网络服务，需要设置 setuid 位，并为该场景编 制相应的代码 我们假设 http\_server.exe 是一个需要 HTTP 网络服务的程序，所以我们编制以下代码设置其 setuid 位。





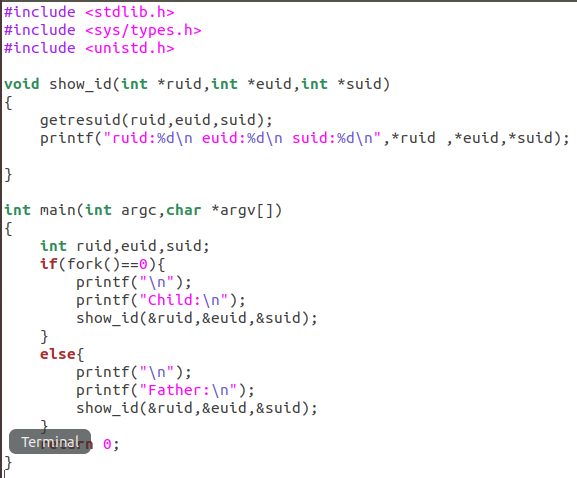
2. 如果用户 fork 进程后，父进程和子进程中 euid、ruid、suid 的差别

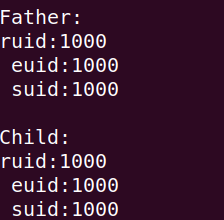
(1)我们使用 getresuid(ruid,euid,suid)获得 ruid,euid,suid，因此我们可以写 如下脚本。

(2)我们根据 fork()函数的返回值的差异来判断当前处于父进程还是子进程。

(3)当 fork()>0 时我们处于父进程，因此我们可以打印出父进程的 ruid,euid, suid

(4)当 fork()==0 时我们处于子进程，因此我们可以打印出子进程的 ruid,euid, suid

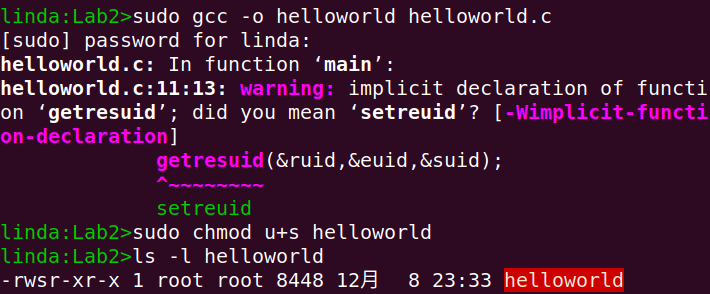


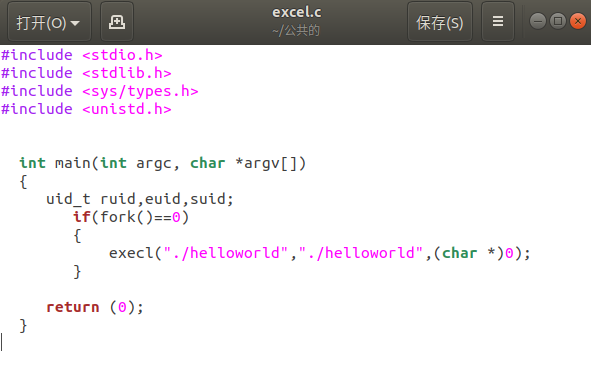


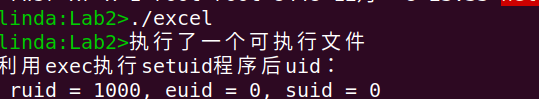
由图可知：父进程和子进程的 ruid,euid 和 suid 都是分别相等

3.利用 execl 执行 setuid 程序后，euid、ruid、suid 是否有变化





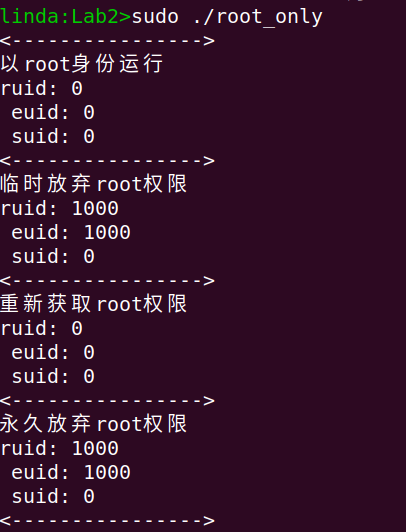




从结果我们可以得出运行 setuid 后 ruid 不变，而 euid 和 suid 变为了 0。因为文件的拥有者为 root。

4.程序何时需要临时性放弃root权限，何时需要永久性放弃root权限，并在程序中分别实现两种放弃权限方法。





可以看到临时性放弃 root 权限的时候，只有 euid 变为其他用户的 uid，ruid 和 suid 都还保持为0。

5.execl函数族中有多个函数，比较有环境变量和无环境变量的函数使用的差异。

exec 家族一共有六个函数，分别是：

(1)int execl(constchar\*path,constchar\*arg,......);

(2)int execle(constchar\*path,constchar\*arg,......,char\*constenvp[]);

(3)int execv(constchar\*path,char\*constargv[]);

(4)int execve(constchar\*filename,char\*constargv[],char\*constenvp[]);

(5)int execvp(constchar\*file,char\*constargv[]);

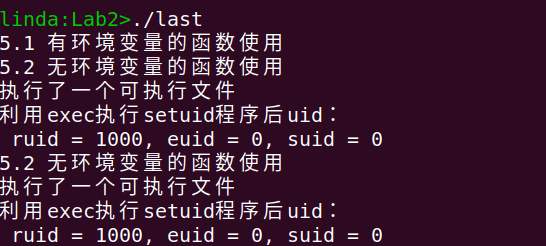
(6)int execlp(constchar\*file,constchar\*arg,......);

其中以 p 结尾的函数，可以向函数传递一个指向环境字符串指针数组的指 针。

即自个定义各个环境变量，而其它四个则使用进程中的环境变量。

在这里我们讨论execl()和execlp()





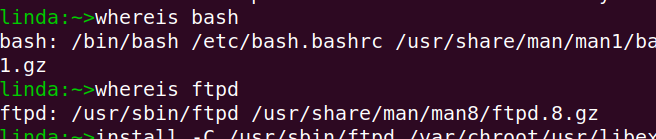
execl()用来执行参数path字符串所代表的文件路径，接下来的参数代表执行该文件时传递过去的argv(0)、argv[1]……，最后一个参数必须用空指针(NULL)作结束。

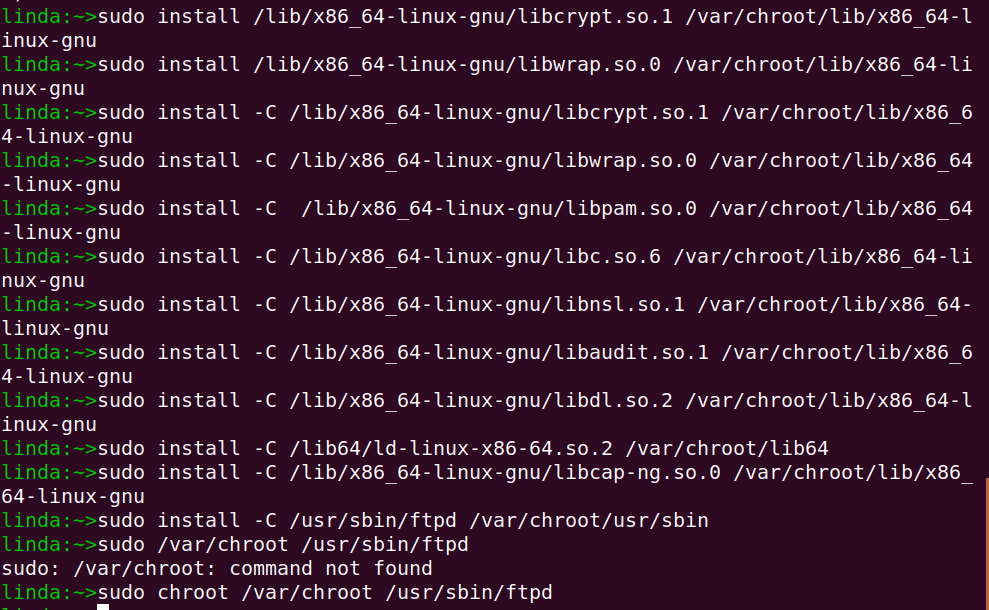
execlp()会从PATH 环境变量所指的目录中查找符合参数file的文件名，找到后便执行该文件，然后将第二个以后的参数当做该文件的argv[0]、argv[1]……，最后一个参数必须用空指针(NULL)作结束。

* 1. **chroot的配置**

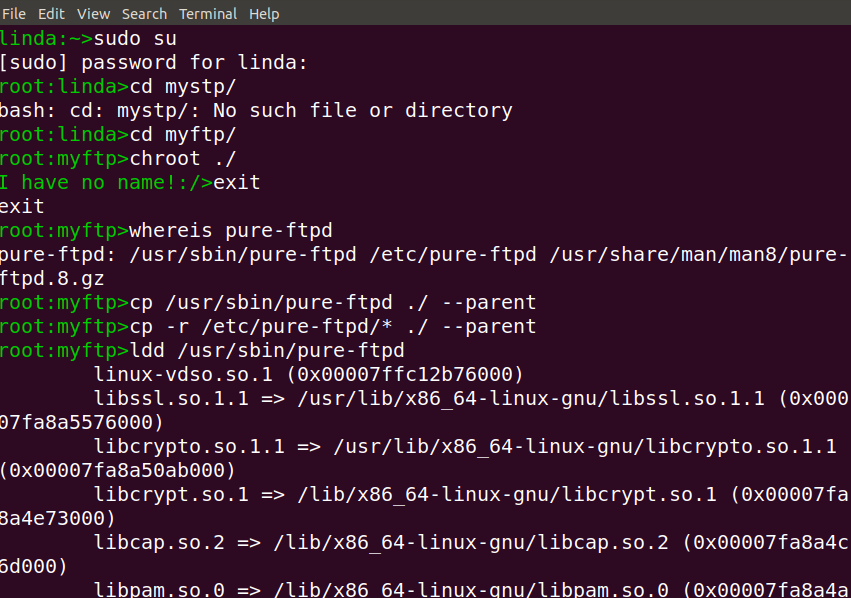
1.利用chroot工具来虚拟化管理

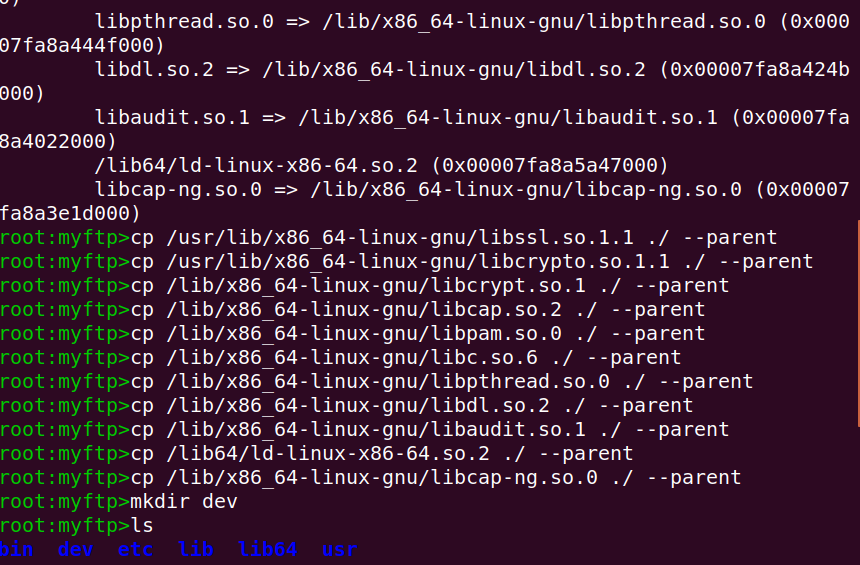
1) 实现bash或ps的配置使用；

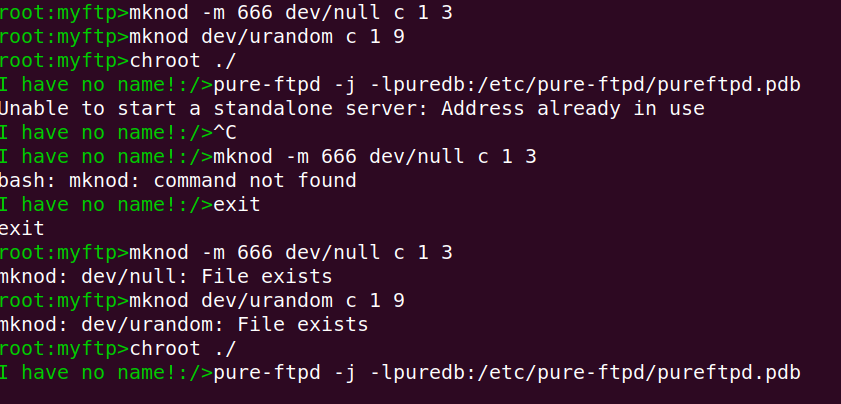


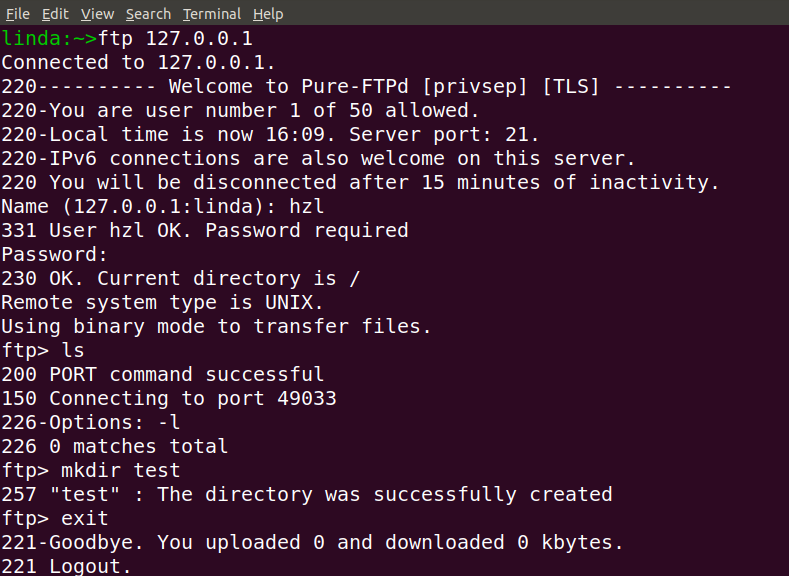


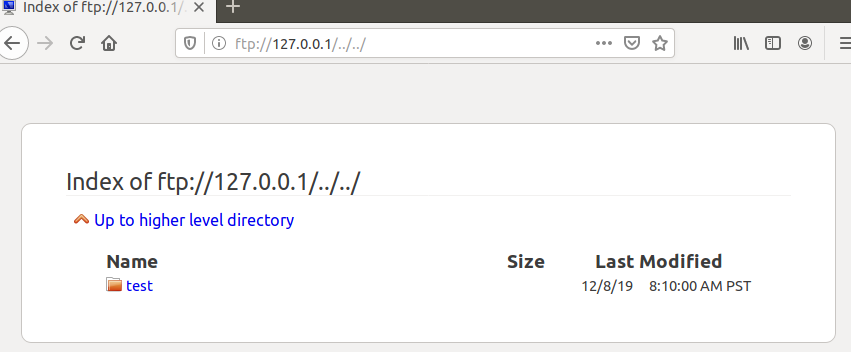
2)利用chroot实现SSH服务或FTP服务的虚拟化隔离：

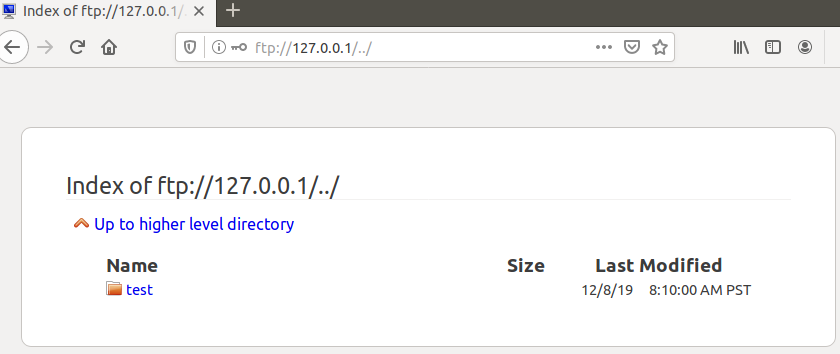








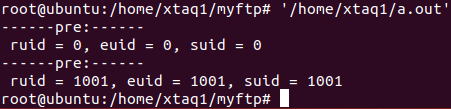


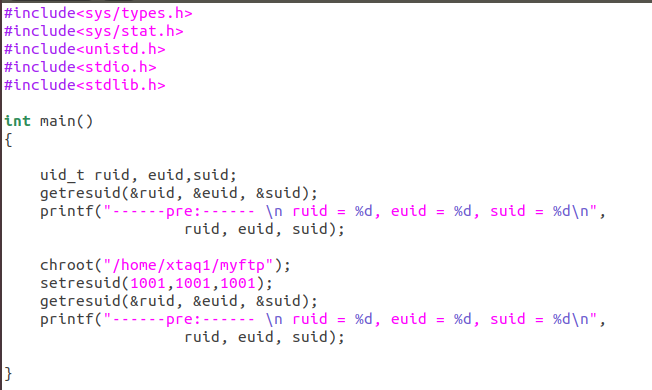


如图所示，无论点击多少次返回上一层目录，都还是在这个目录下，即实现了虚拟化隔离。

3)chroot后如何降低权限，利用实验一中编制的程序检查权限的合理性；

在chroot下直接调用程序，发现ruid,euid,suid都为0





**4.在chroot之前没有采用cd xx目录，对系统有何影响，编制程序分析其影响。**

没有在监狱目录chroot，此时创建的监狱将会拥有当前目录的访问权限，超出了监狱的范围。

1. 如果测试结果打印出上级的目录，说明可以访问所在目录的信息及文件。

（2）在代码中加一行chdir，使chroot前目录已经变为监狱目录，结果表明，最上层为/，而对于其他目录，是透明的。

**心得体会：**

本次实验我加深了对 Linux 环境下进行了目录与文件权限的理解、

掌握了setuid 位的使用 和对euid、 ruid、suid的认识、以及execl 函数族的使用

对 chroot 虚拟化隔离有了更具体的认识，对三种uid的合理使用增强系统的安全性有了深刻认识。