openEuler内核编程技术

实训指导书

第十章 第3讲

《内核安全模块之AppArmor》

软件所制

**术语表**

|  |  |  |  |
| --- | --- | --- | --- |
| 英文 | 中文 | 含义 | 别名 |
|  |  |  |  |
|  |  |  |  |

第十章 第3讲 内核安全模块

# 1 任务1：启用apparmor内核安全模块 (60min)

## 相关知识

**一、AppArmor简介**

AppArmor（Application Armor，应用盔甲）是linux内核的一个安全模块，它允许系统管理员将每个程序与一个安全配置文件关联，从而限制程序的功能。也就是说：通过AppArmor可以指定程序可以读、写或运行哪些文件，是否可以打开网络端口等。AppArmor是传统Unix的自主访问控制模块（DAC）的补充，是linux内核中的强制访问控制系统。

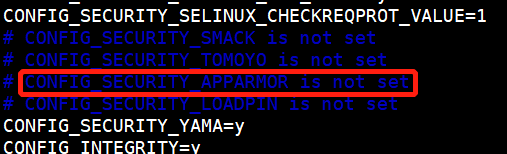
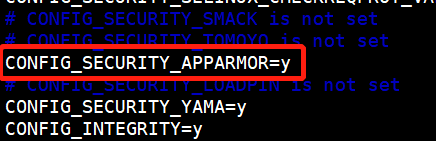
**二、如何启用AppArmor模块**

1、查看系统内核中的AppArmor模块是否开启：

|  |
| --- |
| cat /sys/module/apparmor/parameters/enabled |

* 若 /sys/module/ 下不存在 apparmor/ 目录，则说明内核未开启apparmor模块；
* 若目录文件存在，且输出为Y，说明AppArmor模块已开启、已启用；  
  若输出为N，则说明AppArmor模块已开启、但未启用。

2、若系统未开启AppArmor模块，则需在内核源码的配置文件中，开启CONFIG\_SECURITY\_APPARMOR配置选项（将左图改为右图）：

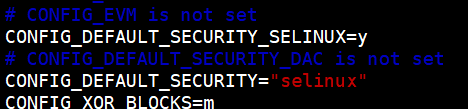
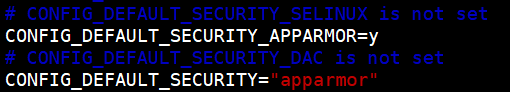
设置：CONFIG\_SECURITY\_APPARMOR=y后，重新编译、安装内核（相关内容在第一章实验课中已详细介绍）。

说明：若基于第一章实验中，已编译的openEuler内核源码，可使用make menuconfig 图形界面设置来编译内核。

3、若系统已开启AppArmor模块但是未启用

当开启AppArmor模块后，可以发现 cat /sys/module/apparmor/parameters/enabled 的输出仍然为N，也就是开启apparmor模块后默认未启用。为什么呢？

因为虽然在内核中有AppArmor、SELinux及其它安全模块，但只能有一个安全模块被激活。目前系统中默认开启的都是SELinux。因此需要修改内核配置中的默认配置项，将selinux改为apparmor（即：将左图改为右图）：

更多细节可参考官方文档：内核源码树 Documentation/admin-guide/LSM/apparmor.rst 文件。

**三、下载安装Apparmor**

参考：<https://gitlab.com/apparmor/apparmor/-/wikis/Distro_CentOS> 链接的后半部分

1、安装相关依赖（也在安装过程中，提示缺什么就安装什么）

|  |
| --- |
| # dnf install perl-libxml-perl perl-TermReadKey perl-Locale-gettext autoconf automake libtool swig bison gcc-c++ flex tetex-latex gettext-develglibc-static libstdc++-static httpd-devel pam-devel python3-devel dejagnu python3-pyflakes |

注：

* 通过 rpm -qi flex 查看flex的版本是2.6.1，比<https://sourceforge.net/projects/flex/files/> 中的最新版本还新，就不用源码安装，直接用repo源里的。
* 因为安装调试过程反复，上述依赖可能没有列全，若还缺少其他软件包，直接安装即可。

2、下载解压

|  |
| --- |
| #### 选定apparmor下载存放的目录，后续安装时配置环境变量需要使用  # mkdir -p ~/apparmor/ && cd ~/apparmor/  #### 下载最新release版本的源码包  # wget https://launchpad.net/apparmor/2.13/2.13.4/+download/apparmor-2.13.4.tar.gz  #### 解压后进入源码根目录  # tar -xzf apparmor-2.13.4.tar.gz  # cd apparmor-2.13.4 |

3、安装

在root用户下，按照源码根目录apparmor-2.13.4/README.md文件中【Building and Installing AppArmor Userspace】部分的描述进行安装：

|  |
| --- |
| #### 设置环境变量  ####（因为apparmor源码都是用python3；若不指定，当系统中有python2.7共存时，则系统会默认使用python2.7，安装过程中相关lib库等会放到python2.7的目录下，导致后续使用apparmor的工具命令时，会出现找不到module的错误）  # export PYTHONPATH=/path/to/apparmor/apparmor-2.13.4/libraries/libapparmor/swig/python  # export PYTHON=/usr/bin/python3  # export PYTHON\_VERSION=3  # export PYTHON\_VERSIONS=python3  #### 1、libapparmor：apparmor的链接库  # cd libraries/libapparmor  # sh ./autogen.sh  # sh ./configure --prefix=/usr --with-perl --with-python  # make  # make check # 不报错说明配置、编译没有问题  # make install  #### 2、Binary Utilities：安装至/usr/bin/目录下的命令行工具  # cd ../binutils  # make  # make check # 不报错再继续  # make install  #### 3、parser：安装apparmor\_parser命令，用于添加、替换和删除apparmor策略（apparmor\_parser --help）。  # cd ../parser  # make # depends on libapparmor having been built first  # make check # 耗时较长，可跳过  # make install  #### 4、Utilities：：安装至/usr/sbin/目录下的命令行工具  # cd ../utils/  # make  # make check PYFLAKES=/usr/bin/pyflakes3 # 不报错再继续  # make install  #### 5、Apache mod\_apparmor：为apache添加apparmor支持  # cd ../changehat/mod\_apparmor/  # make  # make install  #### 6、PAM AppArmor：为PAM模块添加apparmor支持  # cd ../pam\_apparmor/  # make  # make install  #### 7、Profiles：配置系统安全的AppArmor配置文件与目录  # cd ../../profiles/  # make  # make check # 不报错再继续  # make install |

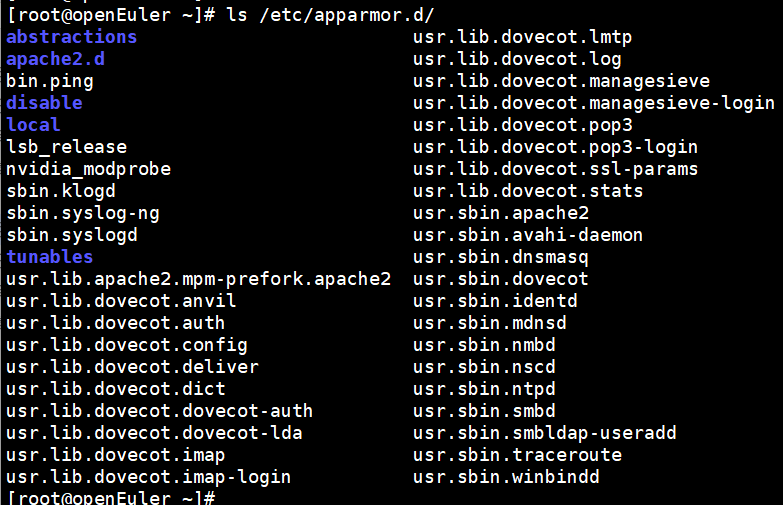
**四、原理与模式**

1、使用原理

AppArmor提供的访问控制是与程序绑定的：

例如：有一个可执行文件的路径为/home/user/appfile，如果用AppArmor对其进行访问控制的话，就要新建一个配置文件命名（与文件路径一致）为home.user.appfile，并把这个文件放到AppArmor专门放置配置文件的目录下（/etc/apparmor.d）（后文会介绍自动生成配置文件的工具）。每一个可执行文件都与一个配置文件绑定的，因此如果修改home.user.appfile文件名，配置文件将失效。

例如：安装好AppArmor后，配置文件目录默认如下：



其中，usr.sbin.apache2 配置文件，即是对 /usr/sbin/apache2文件的访问控制。

2、工作模式

Apparmor有两种工作模式：enforcement、complain/learning。用户可以随时通过命令行工具（后文介绍）对配置文件进行修改，选择自己需要的模式。

（1）enforcement

在这种模式下，配置文件里列出的限制条件都会得到执行，并且对于违反这些限制条件的程序会进行日志记录。

（2）complain

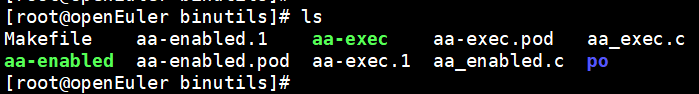
在这种模式下，配置文件里的限制条件不会得到执行，Apparmor只是对程序的行为进行记录。例如程序可以写一个在配置文件里注明只读的文件，但Apparmor不会对程序的行为进行限制，只是进行记录。

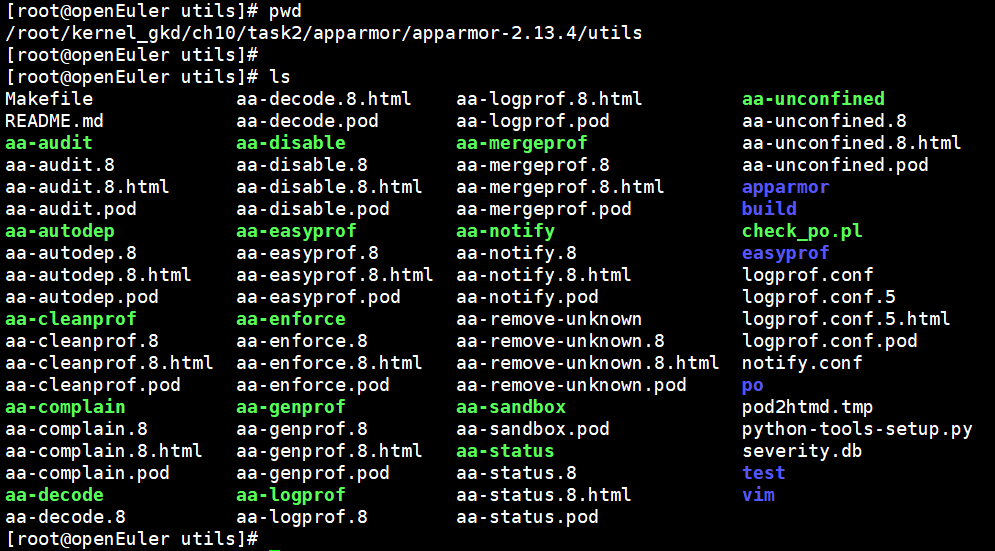
既然complain不能限制程序，为什么还需要这种模式呢，因为：如果某个程序的行为不符合其配置文件的限制，可以将其行为记录到系统日志，并且可以根据程序的行为，将日志转换成配置文件。

**五、命令行工具介绍**

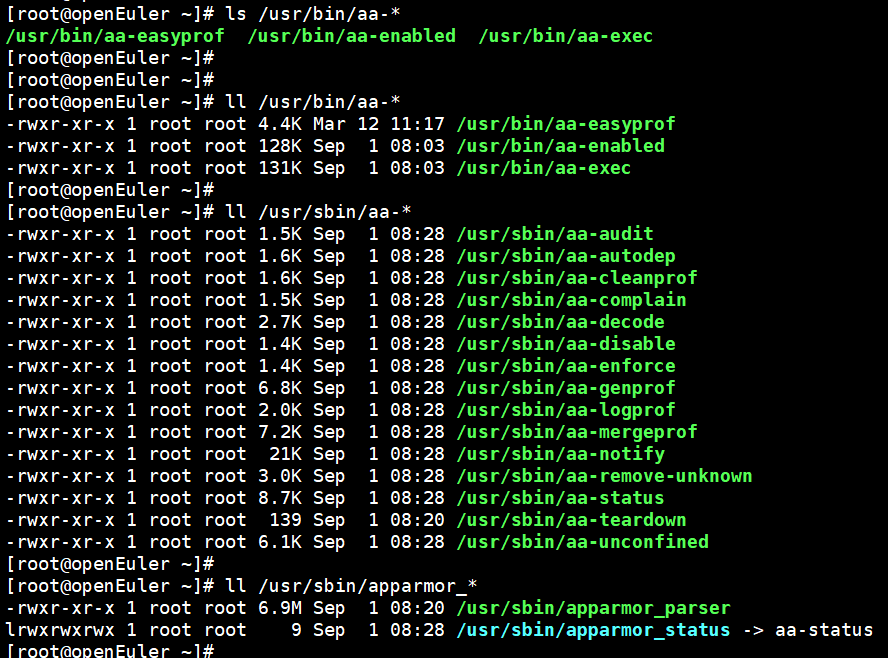
1、工具说明

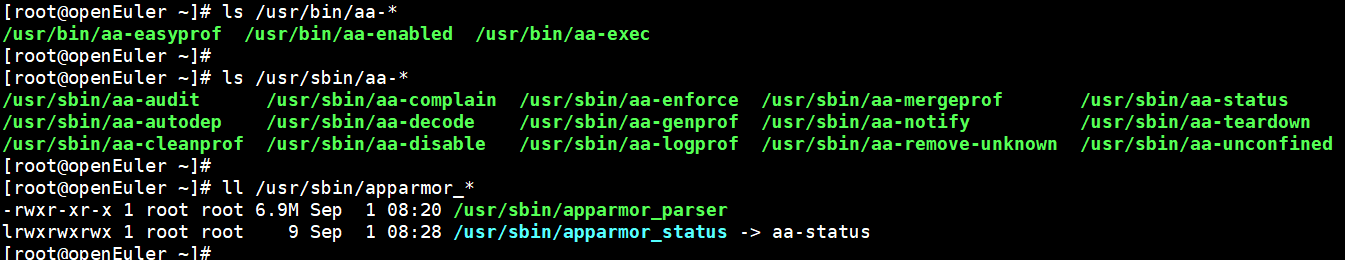
（1）在apparmor源码apparmor-2.13.4目录中，binutils、parse、utils等目录中，都包含了可用的apparmor命令工具；例如（aa-开头的绿色可执行文件都是安装时安装到系统中的命令行工具）：





（2）在/usr/bin、/usr/sbin目录下，aa-\* 开头或者apparmor\_\* 开头的命令，就是安装apparmor后可用的命令行工具：





其中 apparmor\_status 链接至 aa-status，二者效果一样。

可通过 command --help 查看具体用法：如 aa-genprof --help；

也可通过man查看具体用法：如 man aa-genprof。

2、使用说明

（1）aa-easyprof：提供了一个易于使用的界面来生成AppArmor策略。创建的策略可能会比手动或使用aa-genprof和aa-logprof创建策略的限制更少。

（2）aa-enabled：测试是否启用了AppArmor。

（3）aa-exec：用指定的AppArmor配置文件限制程序。

（4）aa-audit：将AppArmor安全配置文件设置为审核模式。

（5）aa-autodep：猜测基本的AppArmor配置文件要求。

（6）aa-cleanprof：清除程序文件现有的AppArmor安全配置文件。例如：# aa- cleanprof testfile.sh。

（7）aa-complain：将程序文件的AppArmor安全配置文件设置为complain模式。

（8）aa-decode：解码AppArmor日志文件中的十六进制编码。例如：# cat /var/log/kern.log | aa-decode。

（9）aa-disable：禁用程序文件的AppArmor安全配置文件。例如：# aa- disable testfile.sh。

（10）aa-enforce：从禁用状态或complain模式，将AppArmor安全配置文件设置为enforce模式。

（11）aa-genprof：为指定程序文件自动生成AppArmor配置文件。

（12）aa-logprof：用于更新AppArmor安全配置文件。

（13）aa-mergeprof：合并AppArmor安全配置文件。

（14）aa-notify：显示已记录的AppArmor消息的信息。

（15）aa-remove-unknown：删除未知的AppArmor配置文件。

（16）aa-status：显示当前AppArmor策略的各种信息。

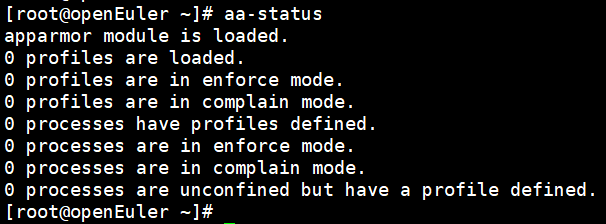
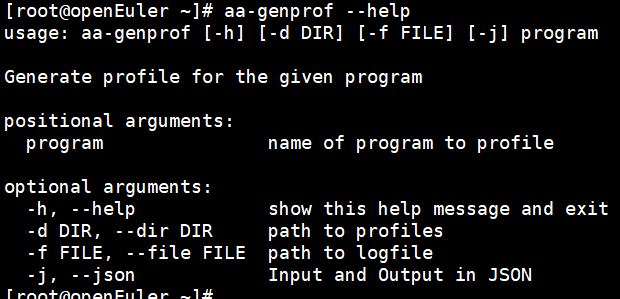
（17）aa-teardown：卸载所有AppArmor配置文件。

（18）aa-unconfined：输出带有tcp或udp端口且未加载AppArmor配置文件的进程的列表。

（19）apparmor\_parser：将AppArmor配置文件加载到内核中。**每次手动修改/etc/apparmor.d/下的配置文件内容后，需执行 apparmor\_parser -r profilename 后方可生效**。

（20）apparmor\_status：同aa-status。

例如：

**六、访问控制实例**

Apparmor可以对程序进行多方面的限制，包括文件系统的访问控制、资源限制、网络的访问控制以及capability条目。本实验主要介绍文件系统的访问控制与应用。

1、文件系统的访问控制

Apparmor可以对某一个文件，或者某一个目录下的文件进行访问控制，包括以下几种访问模式：

|  |  |
| --- | --- |
| r | Read mode |
| w | Write mode (mutually exclusive to a，与a互斥) |
| a | Append mode (mutually exclusive to w，与w互斥) |
| k | File locking mode |
| l | Link mode |
| linkfile->target | Link pair rule (cannot be combined with other access modes) |
| ix | inherit = Inherit the parent's profile. |
| px | requires a separate profile exists for the application, with environment scrubbing. |
| Px | requires a separate profile exists for the application, without environment scrubbing. |
| ux and Ux | Allow execution of an application unconfined, with and without environmental scrubbing. (use with caution if at all). |
| m | allow executable mapping. |

权限包括：可读、可写、可扩展、可链接、可执行等。

在配置文件中的写法：如 /tmp r, （表示可对/tmp目录下的文件进行读取）

注意：没在配置文件中列出的文件，程序无法访问（与白名单类似）。

2、创建简单的可执行文件，自动生成对应的配置文件

|  |
| --- |
| # touch test.sh #创建文件并写入执行程序，如：echo “Hello world!”  # chmod +x test.sh # 为该文件添加执行权限  # aa-genprof test.sh #在/etc/apparmor.d/目录下自动生成对应的配置文件 |

3、手动修改/etc/apparmor.d/目录下对应的配置文件中的文件列表的权限/访问模式并生效后，运行对应的可执行文件，即可验证apparmor的访问控制。

参考：

[1] <https://www.cnblogs.com/zlhff/p/5464862.html>

[2] <https://yq.aliyun.com/articles/80967?scm=20140722.184.2.173>

[3] <http://www.361way.com/linux-apparmor/3646.html>

[4] <https://www.cnblogs.com/cxyc005/p/13323116.html>

[5] <http://www.361way.com/apparmor-selinux/3648.html>

## 任务描述

1. 修改树莓派中openEuler的内核配置选项并编译内核，启用apparmor内核安全模块；
2. 下载apparmor源码包，安装apparmor相关应用。

## 审核要求

1. 正确启用apparmor安全模块，正确安装apparmor应用。
2. 提交相关源码与过程截图。

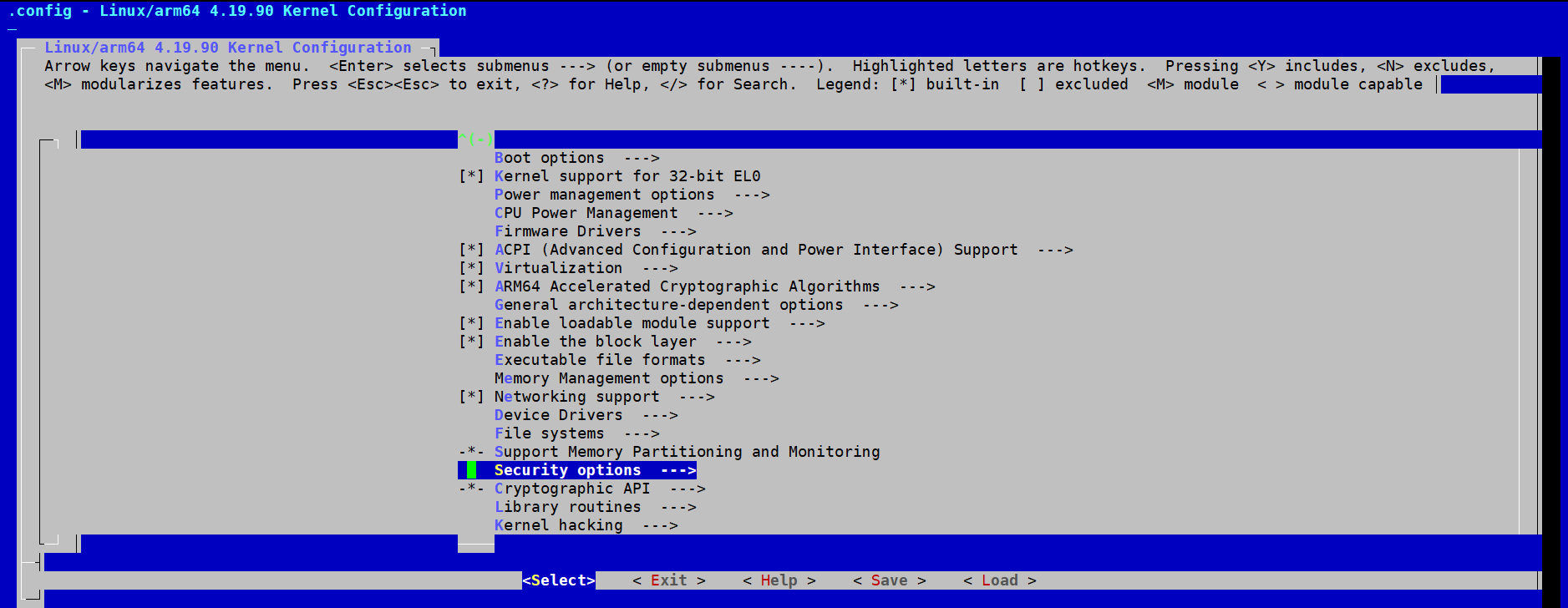
## 参考答案

一、基于第一章实验课中下载并编译完成的内核源码目录，修改内核配置选项

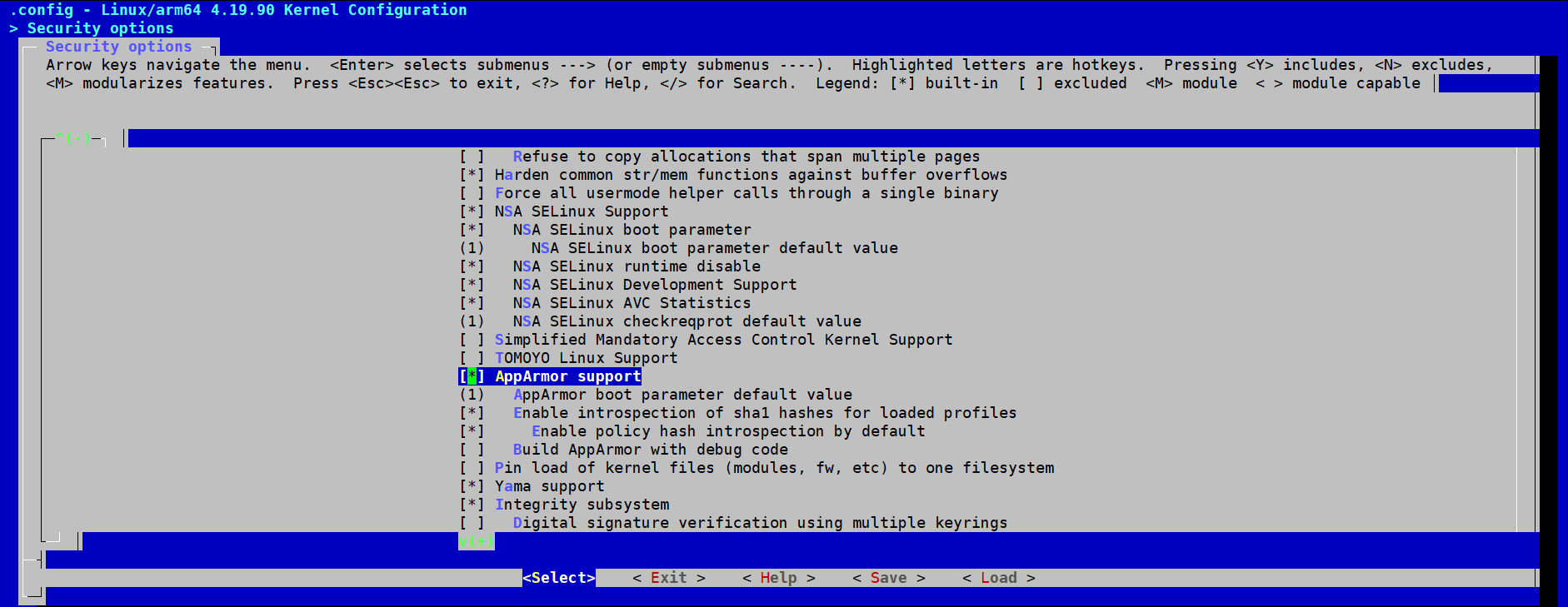
1、修改内核配置选项

|  |
| --- |
| # cd raspberrypi-kernel  # 载入默认配置，openeuler-raspi\_defconfig配置文件 在 /arch/arm64/configs 下。  # make openeuler-raspi\_defconfig //此时内核默认配置加载到源码根目录的.config文件  # make menuconfig // 通过菜单选项的设置修改内核配置选项 |

配置首页往下，找到 Security options ---> 回车进入：

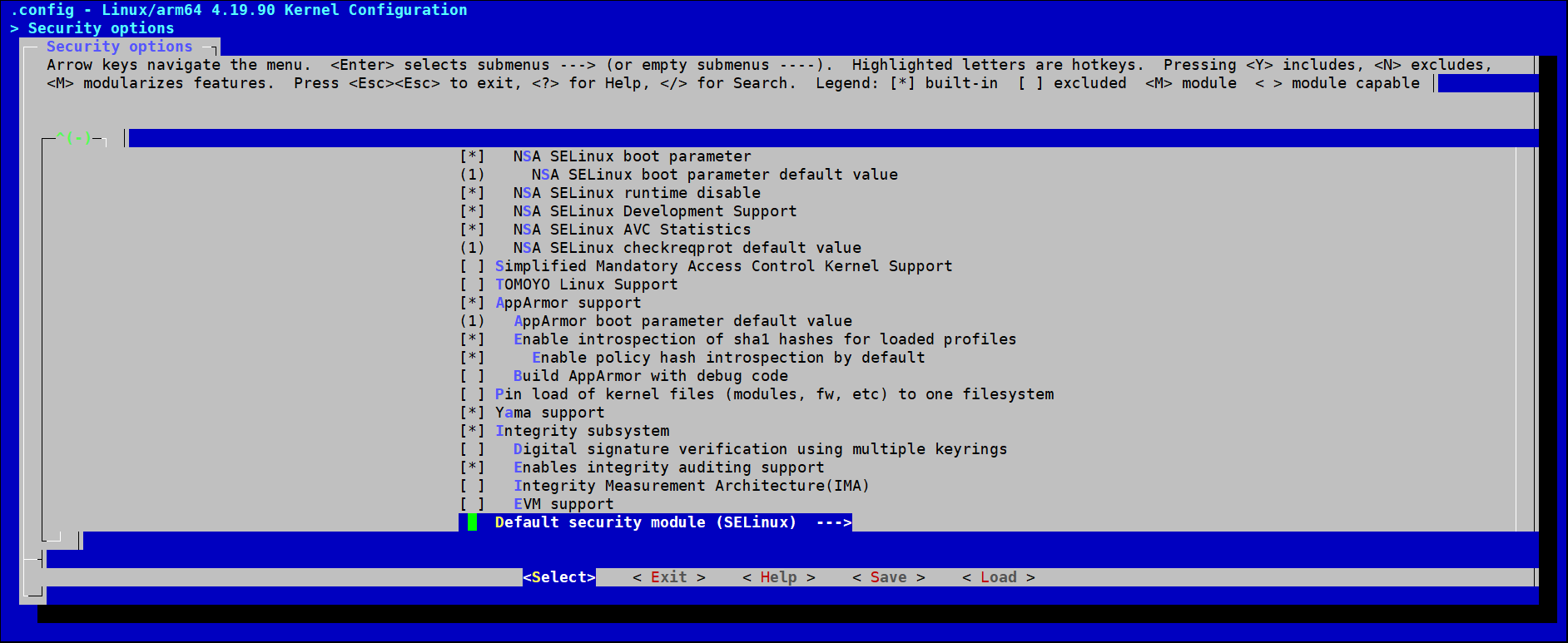


往下找到 []AppArmor support 选项，通过空格键选中（[\*]表示选中）：

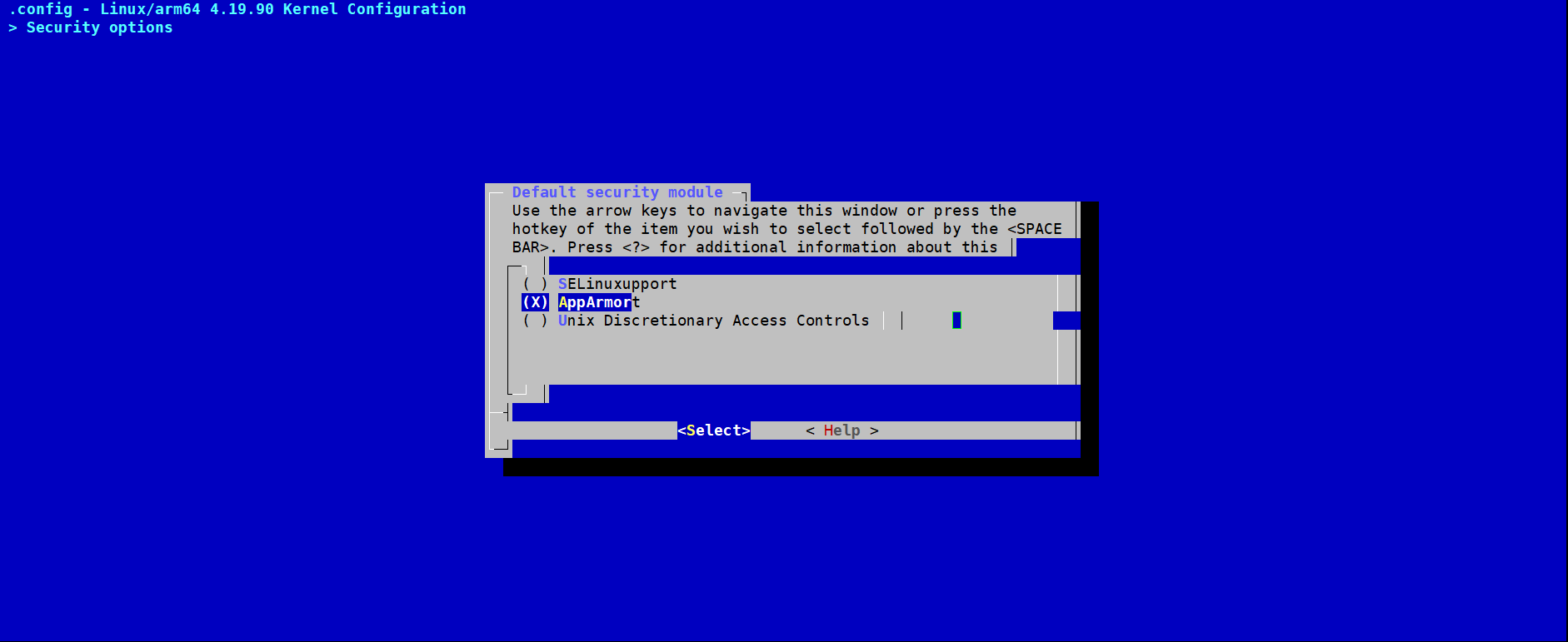


再继续往下，找到 Default security module ，这一选项默认是SELinux：

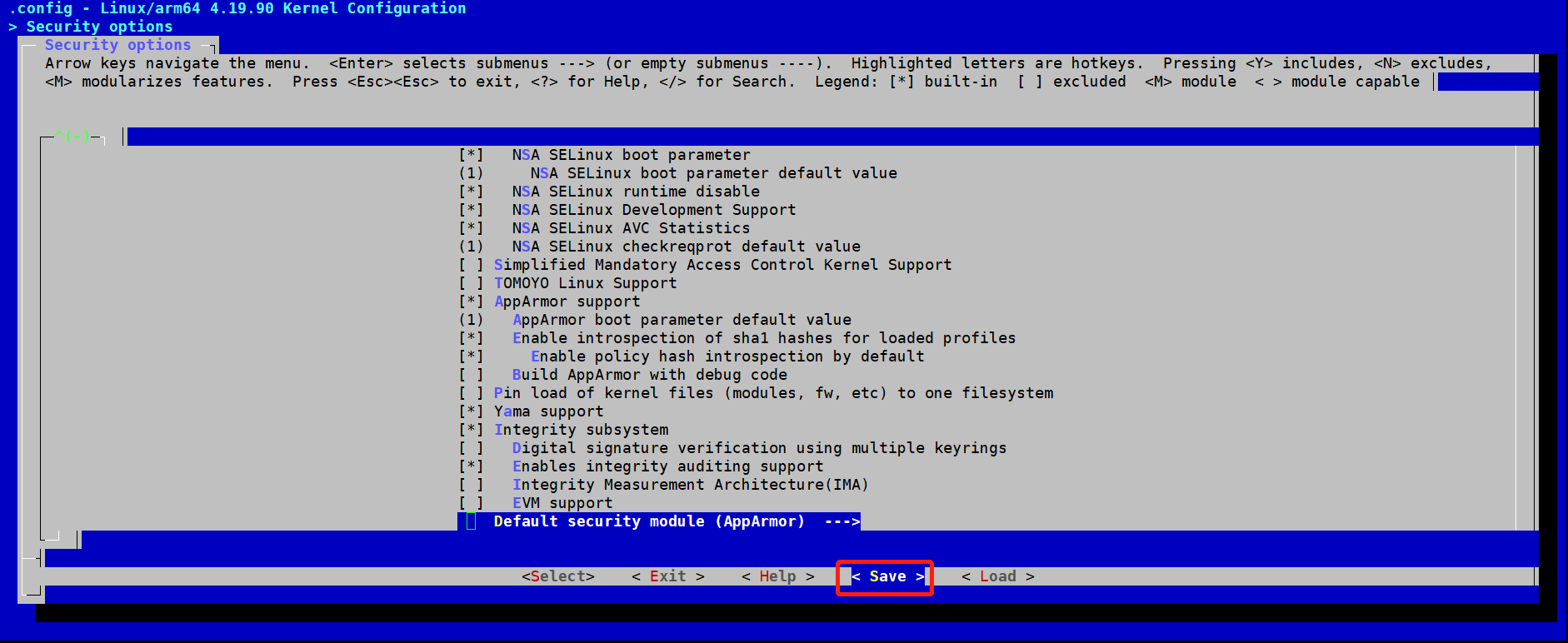
（若不更改此默认选项，那么即使开启上一步的[]AppArmor support 选项，也默认AppArmor不启用。）



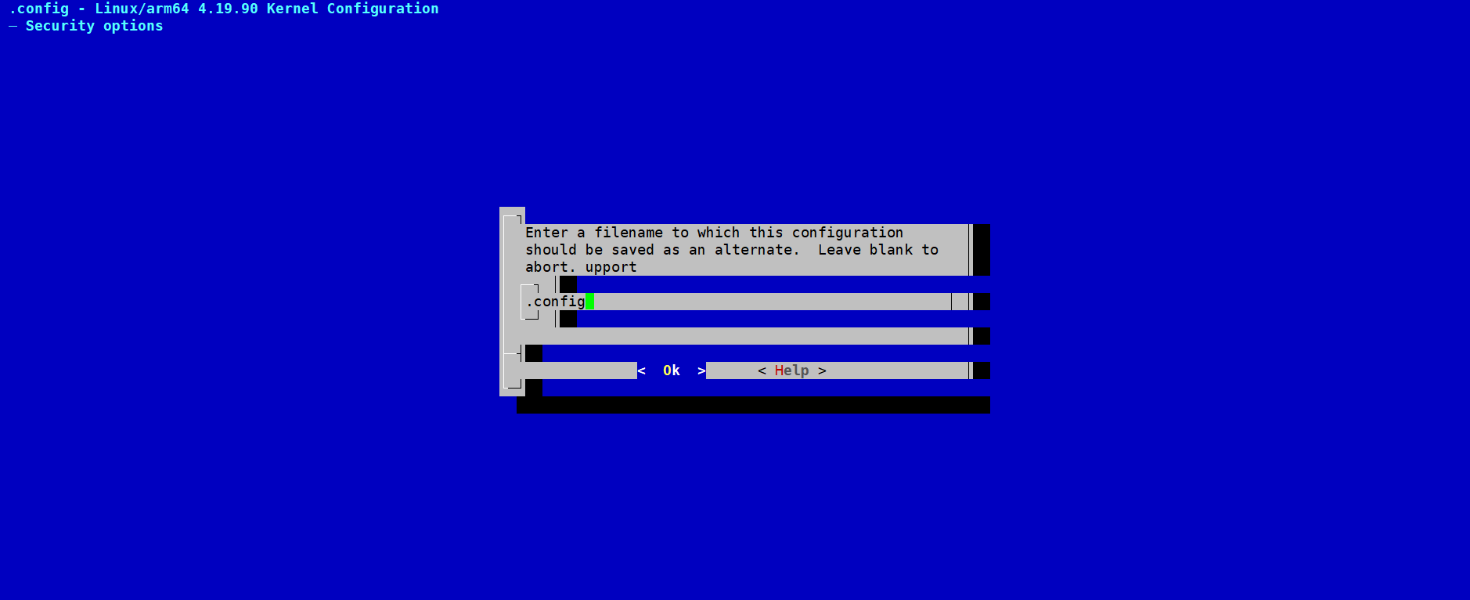
进入后，选择AppArmor：



配置完成后，选择 Save；

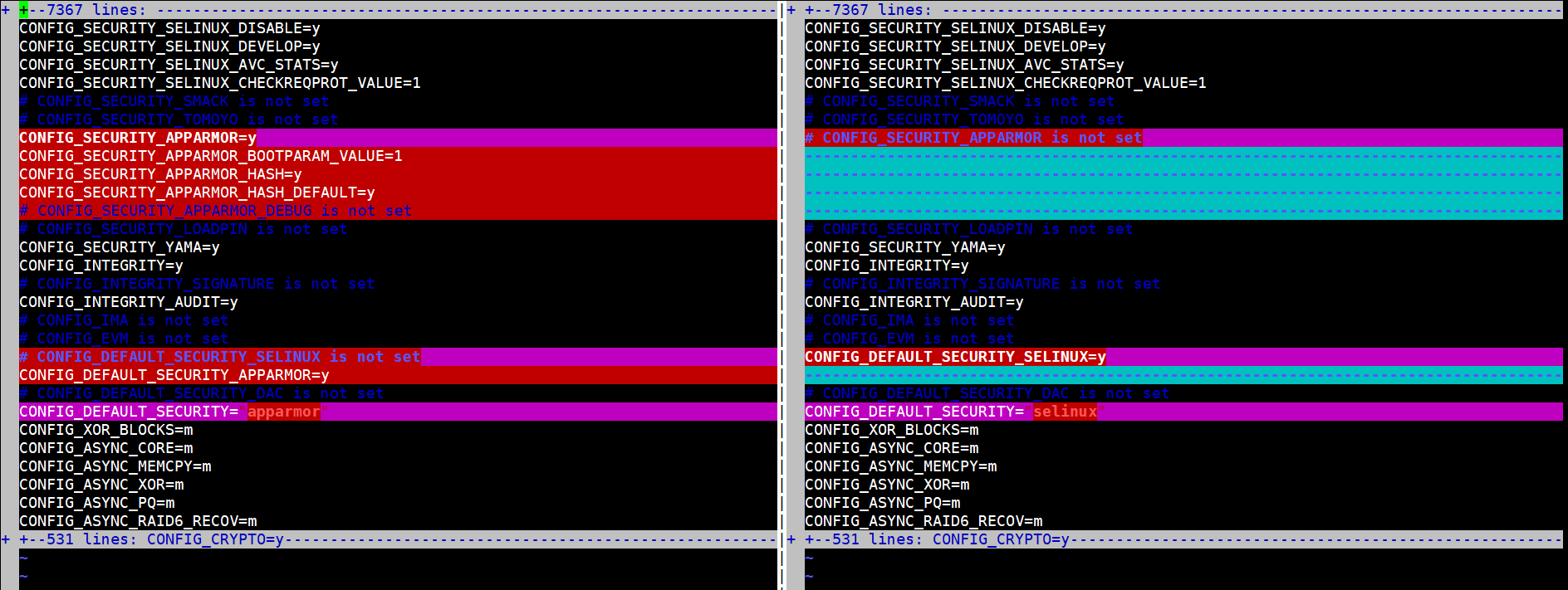


选择 OK：



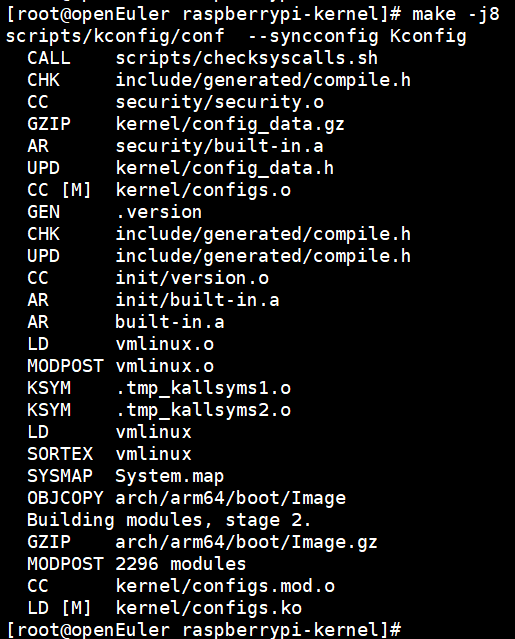
这样，修改后的内核配置选项，就保存到源码根目录的.config文件中了。

可以查看修改前后，内核配置文件的diff：



2、编译内核

|  |
| --- |
| # make -j8 #由于在实验一中，内核源码已经编译过一次内核，再次编译只编译修改部分，用时较短 |



3、安装更新内核

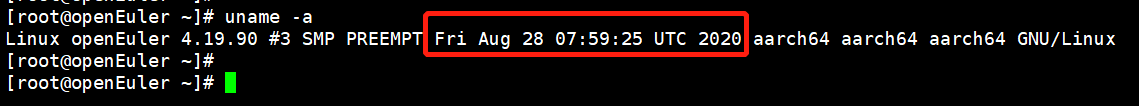
|  |
| --- |
| # 将内核放进引导  # cp arch/arm64/boot/Image /boot/kernel8.img  # 将设备树文件放进引导  # cp arch/arm64/boot/dts/broadcom/\*.dtb /boot/  # cp arch/arm64/boot/dts/overlays/\*.dtb\* /boot/overlays/  # cp arch/arm64/boot/dts/overlays/README /boot/overlays/ |

4、重启系统

|  |
| --- |
| # reboot  # 查看内核版本信息  # uname -a |

重启成功后，通过查看内核版本信息的时间可分析出内核是否更新成功。

若时间为内核编译完成的时间，或内核编译完成后arch/arm64/boot/Image文件的时间，即说明内核更新成功：



另注：若未下载内核源码，可通过以下方式下载：

|  |
| --- |
| # wget https://gitee.com/openeuler/raspberrypi-kernel/repository/archive/openEuler-20.03-LTS-raspi.zip  # 重新下载后，修改配置文件再编译内核就是重新全编译，估计耗时3-4小时 |

二、验证apparmor是否已启用

在命令行输入：

|  |
| --- |
| # cat /sys/module/apparmor/parameters/enabled |

若输出为Y，则说明已启用apparmor：



三、下载安装apparmor

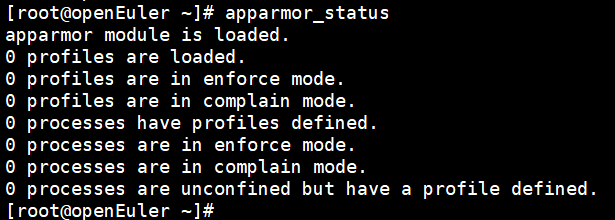
1、安装相关依赖（在安装过程中，提示缺什么就安装什么）

|  |
| --- |
| # dnf install perl-libxml-perl perl-TermReadKey perl-Locale-gettext autoconf automake libtool swig bison gcc-c++ flex tetex-latex gettext-devel glibc-static libstdc++-static httpd-devel pam-devel python3-devel dejagnu python3-pyflakes |

2、下载安装

|  |
| --- |
| # mkdir -p apparmor && cd apparmor  //下载最新release版本的压缩包  # wget https://launchpad.net/apparmor/2.13/2.13.4/+download/apparmor-2.13.4.tar.gz  # tar -xzf apparmor-2.13.4.tar.gz  # cd apparmor-2.13.4  #### 设置环境变量  # export PYTHONPATH=/root/kernel\_gkd/ch10/task2/apparmor/apparmor-2.13.4/libraries/libapparmor/swig/python  # export PYTHON=/usr/bin/python3  # export PYTHON\_VERSION=3  # export PYTHON\_VERSIONS=python3  #### 1、libapparmor:  # cd ./libraries/libapparmor  # sh ./autogen.sh  # sh ./configure --prefix=/usr --with-perl --with-python  # make  # make check  # make install  #### 2、Binary Utilities:  # cd ../../binutils  # make  # make check  # make install  #### 3、parser:  # cd ../parser  # make # depends on libapparmor having been built first  # make check # 耗时较长，可跳过  # make install  #### 4、Utilities:  # cd ../utils/  # make  # make check PYFLAKES=/usr/bin/pyflakes3  # make install  #### 5、Apache mod\_apparmor:  # cd ../changehat/mod\_apparmor/  # make  # make install  #### 6、PAM AppArmor:  # cd ../pam\_apparmor/  # make  # make install  #### 7、Profiles:  # cd ../../profiles/  # make  # make check  # make install |

安装完成后，使用 apparmor\_status 或者 aa-status命令查看 AppArmor 的当前状态：



# 任务2：使用apparmor设置访问控制 (30min)

## 相关知识

同上

**2.2 任务描述**

1. 创建简单的可执行文件，使用apparmor的命令行工具，验证apparmor模块的访问控制。

**2.3 审核要求**

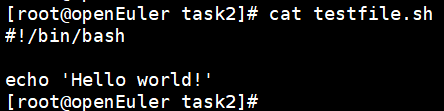
1. 正确使用apparmor工具生成配置文件，以及验证访问控制。
2. 提交相关源码与过程截图。

**2.4 参考答案**

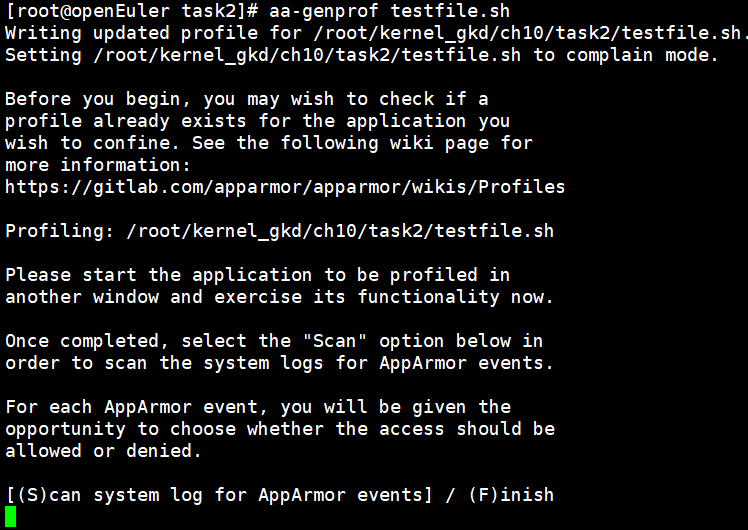
一、使用apparmor设置访问控制

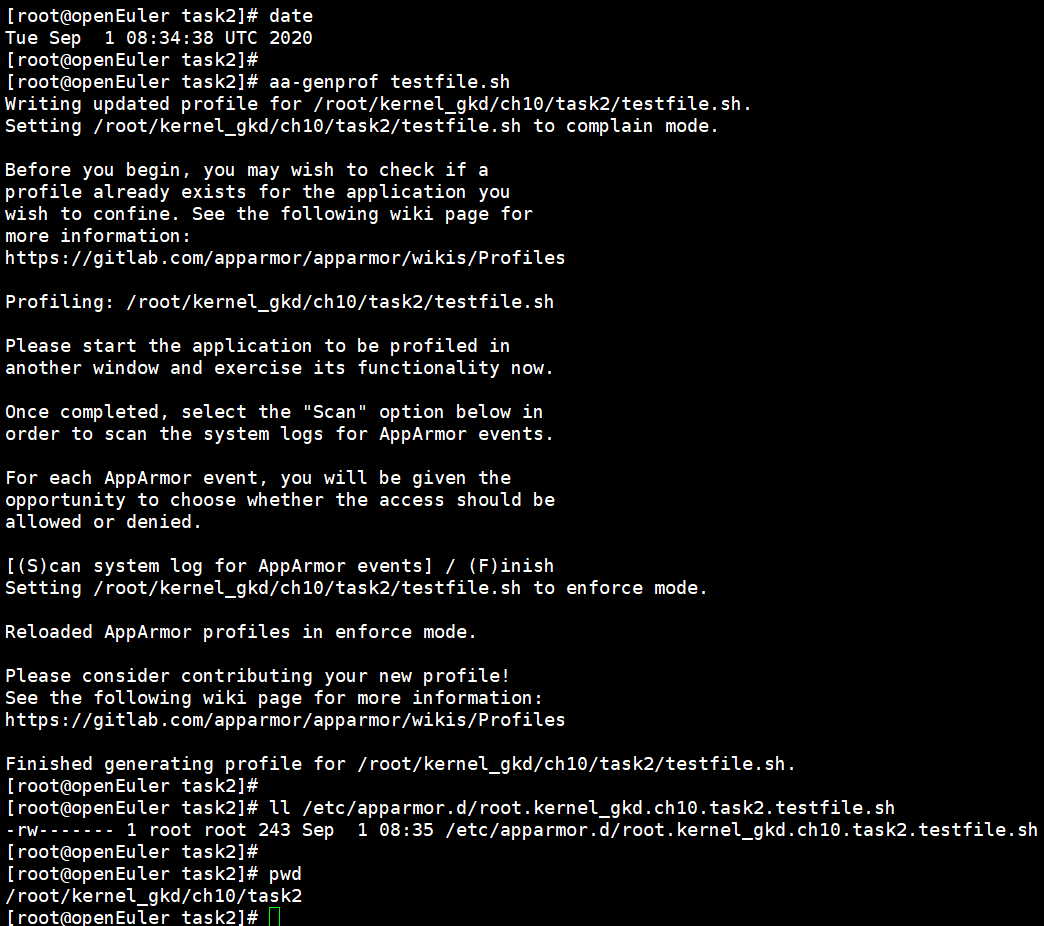
1、文件系统的访问控制

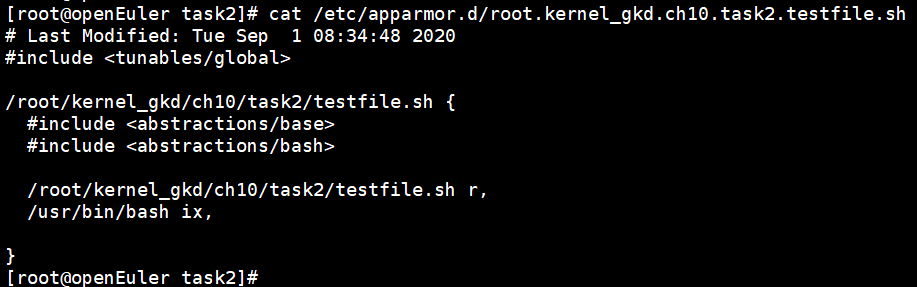
（1）创建测试文件testfile.sh，并写入执行代码



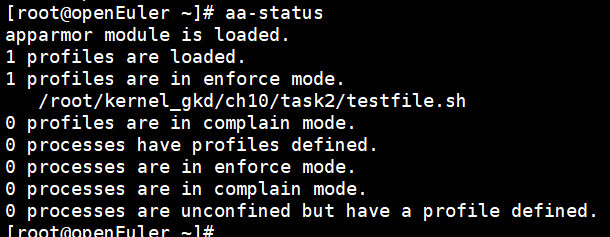
（2）使用 aa-genprof 工具，在/etc/apparmor.d/目录下自动生成对应的配置文件：





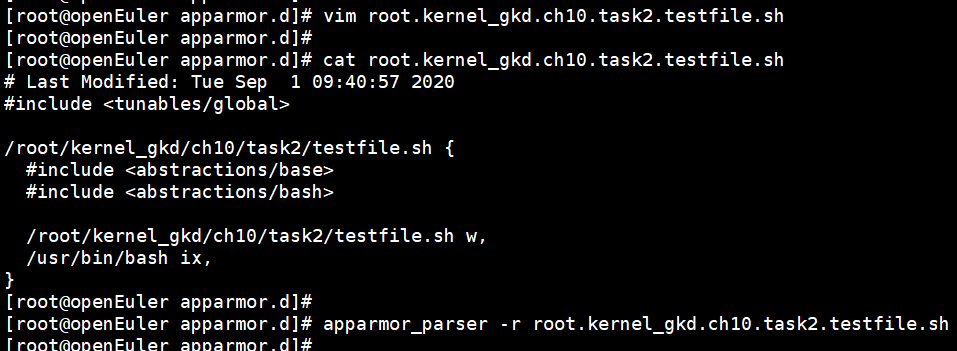


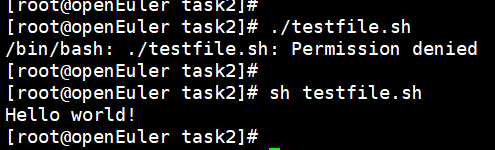
查看状态时，可列出上述受apparmor访问控制的可执行文件：



由默认生成的配置文件可见，程序对可执行文件的权限只有r（读权限）。

当手动修改该可执行文件的权限并重新载入内核，可见权限发生了相应的变化：





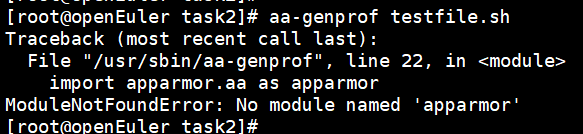
说明：

（1）当修改 /root/kernel\_gkd/ch10/task2/testfile.sh 的权限为w后，由于没有了r权限，而程序执行时又必须要r权限，所以直接执行./testfile.sh是会出现权限问题；

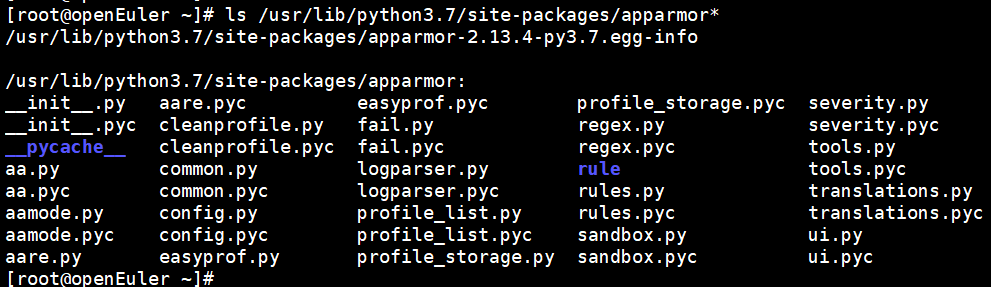
（2）而 /usr/bin/bash ix 的配置没有修改，说明bash对/root/kernel\_gkd/ch10/task2/testfile.sh有执行权限，可以正常执行输出结果。

**2.5 可能遇到的问题**

1、在安装apparmor完成后，通过aa-genprof 自动生成配置文件时，报错如下：



（1）原因：正常在按照README.md安装完成之后，会在/usr/lib/python3/site-packages目录下生成apparmor-2.13.4-py3.7.egg-info文件与apparmor 模块目录：



如果发现上述两个文件/目录位于/usr/lib/python2.7/site-packages目录下，则说明安装时的环境变量设置不正确，使用了系统默认的python2.7。

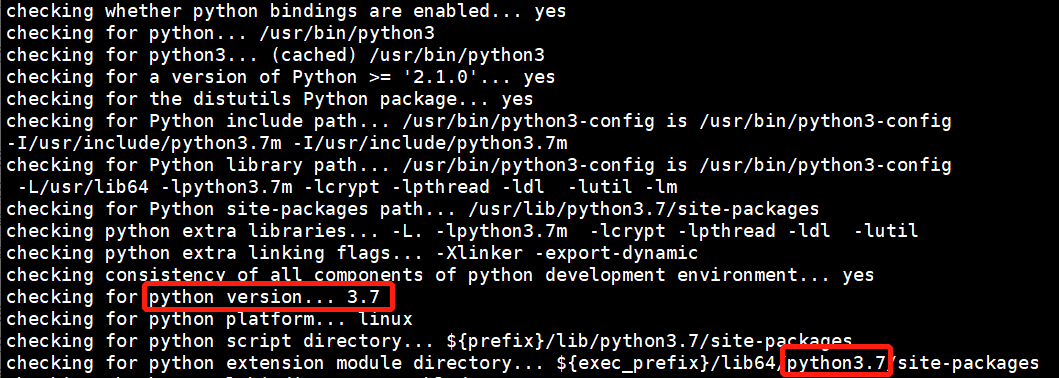
（2）解决办法：

重新卸载 libapparmor 库：

|  |
| --- |
| #### 进入apparmor的源码目录  # cd /root/kernel\_gkd/ch10/task2/apparmor/apparmor-2.13.4/libraries/libapparmor  # make uninstall #卸载 |

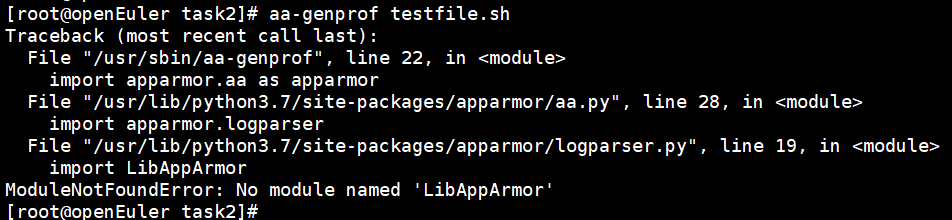
卸载完成后，再按照 apparmor-2.13.4/README.md文件中【Building and Installing AppArmor Userspace】部分的流程，重新进行环境变量设置与安装。

注：在配置安装libapparmor库时，可在./configure 配置的输出信息中，看见关联的python的版本：



如果提示缺少相关文件或模块，则安装python3对应的模块；例如，提示缺少 python3-config文件，则需安装 python3-devel。

2、在安装apparmor完成后，通过aa-genprof 自动生成配置文件时，报错如下：



（1）原因：

libapparmor库配置安装后在源码目录下：apparmor-2.13.4/libraries/libapparmor/src，系统自动查找时没有找到。

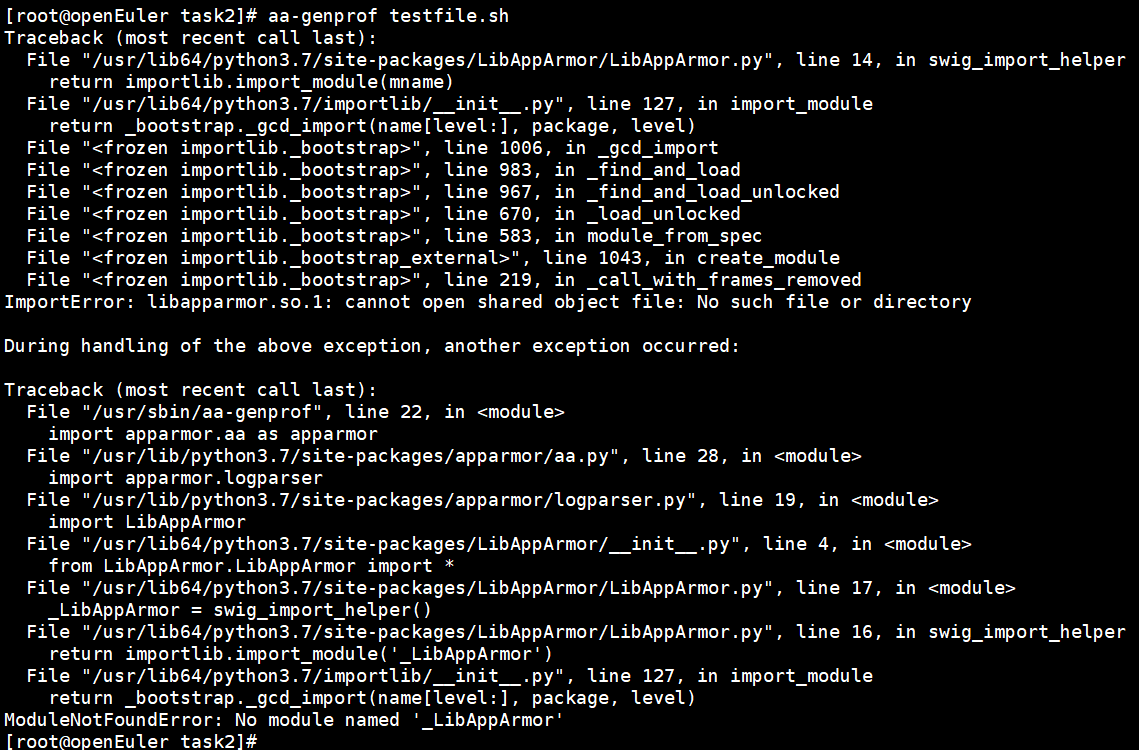
（2）解决办法：

将apparmor-2.13.4/libraries/libapparmor/src路径添加到环境变量PATH中：

|  |
| --- |
| # export PATH=$PATH:/root/kernel\_gkd/ch10/task2/apparmor/apparmor-2.13.4/libraries/libapparmor/src  # ln -s libraries/libapparmor/src/ /usr/lib/ |

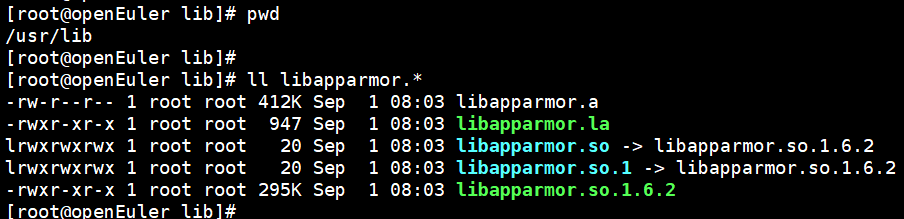
3、在安装apparmor完成后，通过aa-genprof 自动生成配置文件时，报错如下：

（找不到libapparmor.so.1文件）



（1）原因：

libapparmor.so.1文件可以看见是在/usr/lib/目录下：



位于apparmor-2.13.4/libraries/libapparmor/src目录下，而链接库一般在/usr/lib和/lib中，系统自动查找时没有找到。

（2）解决办法：

将apparmor-2.13.4/libraries/libapparmor/src路径添加到环境变量PATH中：

|  |
| --- |
| #### 进入源码目录  # cd /root/kernel\_gkd/ch10/task2/apparmor/apparmor-2.13.4/libraries/libapparmor/src  # ln -s libraries/libapparmor/src/ /usr/lib/ |

参考：

[1] <https://gitlab.com/apparmor/apparmor/-/wikis/Distro_CentOS>