

openwrt SELinux policy

1. SELinux feature enable

1.1. 编译使能

可通过如下编译命令 enable openwrt SELinux feature;

```
make OPT_SELINUX=y
```

1.2. SELinux 状态

SELinux 有 2 种状态:(目前默认为 enforce 模式)

- **permissive:** 警告模式: 系统不会受到 SELinux 保护, 只是收到警告信息。即使违反了策略的话也可以继续操作, 但是会把违反策略的内容记录下来;
- **enforcing:** 强制模式: 系统受到 SELinux 保护, 开启安全上下文的匹配, 任何违反策略的操作都会被拦截;

在 linux shell 中查看 /etc/selinux/config 文件可确定当前模式:

```
root@OpenWrt:/# cat /etc/selinux/config  
  
SELINUX=enforcing  
SELINUXTYPE=selinux-policy
```

1.2.1. SELinux 状态设置

1) 命令行配置

在 linux shell 中可通过 setenforce 命令临时切换状态:

```
setenforce [Enforcing | Permissive | 1 | 0]
```

此方法生效只是临时的, 重启后配置不会保留.

2) code 中配置

在 `package/system/selinux-policy/files/selinux-config` 文件中也可手动修改默认模式:

```
SELINUX=enforcing  
SELINUXTYPE=selinux-policy
```

2. openwrt SELinux policy

2.1. CIL 语法

openwrt 提供了两种 SELinux 的 policy，1803 项目上使用的是专门为 openwrt 优化的 DefenseSec SELinux Security Policy(CONFIG_SELINUXTYPE_dssp), 其源代码位于 build_dir/target-arm_cortex-a7+neon-vfpv4_musl_eabi/selinux-policy-0.8/src/.

此 selinux-policy 是基于 cil 语法实现的，详细的 CIL 语法描述可参考 <https://github.com/SELinuxProject/cil/wiki>

此外还有部分 CIL 语句语法需参考 secilc(SELinux CIL Compiler) 文档:

<https://github.com/SELinuxProject/selinux/tree/master/secilc/docs>

2.2. SELinux policy

openwrt SELinux policy 里面的 type 是可以由开发者随意定义的，所以导致会有不同的框架出现。但其所实现的 block 继承结构和系统文件/目录的层级结构存在一定的对应关系：

1. av.cil - 定义了 object class;
2. subj.cil - type subj 与 subj_typeattr attribute;
3. dev.cil 及 dev/目录 -

dev/ttydev.cil：在 dev block 中定义 ttydev type 及相关的 allow rule macro，并通过 filecon 语句为 /dev/tty.+ 所有文件生成默认的安全上下文: (u:r:dev.ttydev)

dev/ttydev/consolettydev.cil：console block 继承 dev.tty.obj_template，并为/dev/console 文件生成默认安全上下文: (u.r.console.ttydev)

dev/stordev.cil：在 dev.stor block 定义 stordev type 及相关的 allow rule macro;

dev/stordev/mtdstordev.cil : mtd block 继承 dev.stor.obj_template, 并为 /dev/mtd、/dev/ubi 文件生成默认的安全上下文: (u:r:mtd.stordev)

dev/stordev/sdstordev.cil : sd block 继承 dev.stor.obj_template, 并为 /dev/sd* 文件生成默认安全上下文: (u:r:sd.stordev)

dev/stordev/mmcstordev.cil : mmc block 继承 dev.stor.obj_template, 并为 /dev/mmcblk* 文件生成默认安全上下文: (u:r:mmc.stordev)

dev/stordev/...

dev/nodedev.cil: 在 dev.node block 定义 nodedev type 及相关的 allow rule macro;

dev/nodedev/i2cnodedev.cil: i2c block 继承 dev.node.obj_template, 并为 /dev/i2c-[0-9]+ 文件生成默认安全上下文: (u:r:i2c.nodedev)

dev/nodedev/usbnodedev.cil: usb block 继承 dev.node.obj_template, 并为 /dev/bus/usb/.+、/dev/usb([0-9]+)? 文件生成默认安全上下文: (u:r:usb.nodedev)

dev/nodedev/...

...

dev/.../...

...

综上可发现存在明显的规律:

1)目录名隐含了 block 的继承关系及文件 context type, 例如从 usbnodedev.cil 文件名即可得知此文件定义了 dev > node > usb block 的继承关系(文件名为 3 者倒序排列), 并定义相关安全上下文;

2)目录层级结构与系统文件层级结构匹配;

4. sysfile.cil 及 sysfile/ 目录 -

sysfile.cil: 定义 sysfile type 及相关的 allow rule macros;

sysfile/blocksysfile.cil : sysfile.block block 为 /sys/block 文件生成安全上下文 u:r:block.sysfile;

sysfile/classsysfile.cil : sysfile.class block 为 /sys/class 文件生成安全上下文 u:r:class.sysfile;

sysfile/classsysfile/gpioclasssysfile.cil : 为 /sys/class/gpio 文件生成安全上下文 u:r:gpio.sysfile;

...

文件的层级关系、block 继承关系同上;

5. 其他部分 cil 文件与上述规律略有不符合, 但整体的 block 继承关系与 context type 命名是一致的, 此处不再详述.

2.3. 为文件/目录修改安全上下文

根据上一小节对于文件层级结构与继承关系的理解, 为一个新的文件或目录单独创建 type 及安全上下文就很简单了.

以 /usr/bin/adbd 为例:

当前 policy 在 src/file/execfile.cil 文件中为 /usr/bin 目录及其他所有文件都生成了默认的安全上下文:

u:r:file.execfile;

```

11      (filecon
12      "/usr/bin"
13      dir
14      execfile_file_context)
15      (filecon
16      "/usr/bin/.*"
17      any
18      execfile_file_context)

```

编译产物 build_dir/target-arm_cortex-a7+neon-vfpv4_musl_eabi/selinux-policy-0.8/file_contexts 中可确定默认的安全上下文:

```

52 /usr/bin/.*      u:r:file.execfile
214 /usr/bin      -d      u:r:file.execfile

```

系统启动后可再次确认:

```

root@OpenWrt:/# ls -Z /usr/bin/adbd
u:r:file.execfile      /usr/bin/adbd

```

假设修改 `/usr/bin/adbd` 安全上下文为 `u:r:adbd.execfile`，可在 `src/file/execfile/` 目录下新增 `adbdexecfile.cil` (具体实现可参考同一层级中的其他 `cil` 文件)

或 `src/agent/` 目录下新增 `adbd.cil`:

```
(in .file
  (call .adbd.obj_type_transition_execfile (unconfined.subj_typeattr)))

(block adbd

  ;;
  ;; Contexts
  ;;

  (filecon
    "/usr/bin/adbd"
    file
    execfile_file_context)

  ;;
  ;; Macros
  ;;

  (macro obj_type_transition_execfile ((type ARG1))
    (call .file.execfile_obj_type_transition
      (ARG1 execfile file "adbd")))

  ;;
  ;; Policy
  ;;
  ;; 这里可以按照需要增加与 adbd.execfile type 相关的 rules

  (blockinherit .file.exec.obj_template))
```

linux 启动后检查 `adbd` 的安全上下文:

```
root@OpenWrt:/# ls -Z /usr/bin/adbd
u:r:adbd.execfile /usr/bin/adbd
```

需要注意的时，默认的安全上下文对应的 `file.execfile` 是一个相对 `common` 的类型，默认 `policy` 已经为其添加了一些 `rules`.

在执行上述修改后，可能会突然报出很多与 `adbd.execfile` 相关的 `avc denied log`，需要按照具体需求分析，逐步添加权限或修改应用程序的行为。

3. avc denied log 分析

下面列举几个具体的 case 来描述 avc denied log 分析过程:

要了解被拒绝的各种访问权限的意义及 avc 信息, 可参考 [SELinux by Example Using Security Enhanced Linux.pdf](#).

3.1. case1: 直接添加权限

```
[ 31.876403] audit: type=1400 audit(51798.909:277): avc: denied { execmem } for pid=1684 comm="dnsmasq" scontext=u:r:dnsmasq.subj tcontext=u:r:dnsmasq.subj tclass=process permissive=0
```

此问题为 dnsmasq.subj 类型主体对 self type 的 execmem 权限被拒绝.

execmem 权限解释: make executable an anonymous mapping or private file mapping that is writable;

通过 apol 工具分析 policy 文件可知: dnsmasq.subj 对 self type 的 process class 包含如下权限:

```
allow dnsmasq.subj dnsmasq.subj process fork, getcap, setcap, sigchld, sigkill, signal, signu
ll, sigstop
```

对于此问题, 可尝试直接添加 execmem 权限:

```
src/agent/dnsmasq.cil 文件:

(allow subj self (process (getcap setcap execmem))) //添加 execmem
```

3.2. case2: 新增 type

```
[ 23.729858] audit: type=1400 audit(55672.531:186): avc: denied { execmod } for pid=1334 comm="m
rdloader" path="/usr/lib/libprop2uci.so" dev="mtdblock11" ino=913 scontext=u:r:sys.subj tcontext=u:r:
file.libfile tclass=file permissive=0
```

权限 execmod: Make execute a file mapping that has been modified by copy-on-write.(文件被修改后仍为其添加可执行权限)

此问题为：主体 `sys.subj type` 对于 `/usr/lib/libprop2uci.so` 库文件的 `execmod` 权限被拒绝，进程名：`mrloader`。

进程 `mrloader` 在执行过程中期望执行此 `lib` 文件；

首先应该确定此操作是否合规，如果需要修改 `policy rule` 来赋予此权限，有两种方式：

1. 为所有的 `sys.subj type` 都赋予向 `file.libfile` 的 `execmod` 权限。

在 `sys.cil` 文件中添加：

```
(allow subj .file.libfile (file (execmod)))
```

但 `sys.subj` 是一个比较基础的 `type`，此操作会使得所有 `sys.subj type` 主体对所有 `file.libfile type` 文件都拥有此权限，可能会导致权限释放范围过大。

2. 修改 `mrloader` 的安全上下文 `type`，单独为其赋予对于 `libfile` 的 `execmod` 权限

新增 `src/agent/mrloader.cil` 文件：(实现可参考同级其他 `cil`)

```
1 ;; -*- mode: CIL; fill-column: 79; indent-tabs-mode: nil; -*-
2 ;; SPDX-FileCopyrightText: © 2021 Dominick Grift <dominick.grift@defensec.nl>
3 ;; SPDX-License-Identifier: Unlicense
4
5 (in .sys
6   (call .mrloader.subj_type_transition (subj)))
7
8 (in .file
9   (call .mrloader.obj_type_transition_execfile
10    (unconfined.subj_typeattr)))
11
12 (block mrloader
13
14   ;;
15   ;; Contexts
16   ;;
17
18   (filecon
19    "/usr/bin/mrloader"
20    file
21    execfile_file_context)      ;;配置 /usr/bin/mrloader 新的安全上下文类型为 mrloader.execfile
22
23   ;;
24   ;; Macros
25   ;;
26
27   (macro obj_type_transition_execfile ((type ARG1))
28     (call .file.execfile_obj_type_transition
29      (ARG1 execfile file "mrloader")))
30
31   ;;
32   ;; Policy
33   ;;
```

```

34
35     (blockinherit .agent.base_template)
36
37     (allow subj .file.libfile (file (execmod)))) ;;新增 rule 允许 execmod 权限
38

```

重新编译后的 `file_contexts` 文件中可确认新的 `mrdloader` 安全上下文:

```
/usr/bin/mrdloader    --    u:r:mrdloader.execfile
```

为 `/usr/bin/mrdloader` 重新配置 context 后会新增与 `mrdloader.subj` type 相关的 `avc denied log`, 需要依次处理.

在调试期间可以将 SELinux 配置为 `permissive` 模式, 以便于一次处理更多的 `avc warning`, 避免需要多次修改.

例如:

```

[ 24.259918] audit: type=1400 audit(60682.831:184): avc: denied { write } for pid=1334 comm="mrd
loader" path="pipe:[3438]" dev="pipefs" ino=3438 scontext=u:r:mrdloader.subj tcontext=u:r:sys.subj tc
lass=fifo_file permissive=0
[ 24.406799] audit: type=1400 audit(60682.831:185): avc: denied { write } for pid=1334 comm="mrd
loader" path="pipe:[3438]" dev="pipefs" ino=3438 scontext=u:r:mrdloader.subj tcontext=u:r:sys.subj tc
lass=fifo_file permissive=0
[ 24.526824] audit: type=1400 audit(60682.831:186): avc: denied { write } for pid=1334 comm="mrd
loader" path="pipe:[3438]" dev="pipefs" ino=3438 scontext=u:r:mrdloader.subj tcontext=u:r:sys.subj tc
lass=fifo_file permissive=0

```

同理可在 `src/agent/mrdloader.cil` 中添加:

```
(call .sys.writeinherited_fifo_files (subj))
```

例如:

```

[ 13.086822] audit: type=1400 audit(63598.111:88): avc: denied { execute } for pid=544 comm="sh"
name="mrdloader" dev="overlay" ino=2204 scontext=u:r:rcboot.subj tcontext=u:r:mrdloader.execfile tcl
ass=file permissive=0

```

此问题为 `rcboot.subj` 对 `mrdloader` 的执行操作被 `denied`.

如果需要为 `boot` 赋予对于 `mrdloader` 的执行权限, 可修改:

```

src/initscript/rcboot.cil

(call .mrdloader.execute_execfile_files (subj))

```

3.2.1. 示例补充：新增 security context

根据上文，通过为/usr/bin/mrdloader 新建 type(mrdloader.execfile), 并赋予其对 .file.libfile type 文件的 execmod 权限，可以使得 mrdloader 拥有对库文件 libprop2uci.so 的 execmod 权限。

基于此可进一步拓展，如何使得某一文件(例如此处的/usr/lib/libprop2uci.so) 只能够被 mrdloader 访问？

由于默认的 selinux policy 会为 /usr/lib/ 目录下的文件都配置默认的 security context(libfile_file_context: u.r.libfile), 并且很多 type 都默认拥有对 libfile type 的访问权限, 因此针对这个问题可以尝试为 libprop2uci.so 文件 重新配置默认的 context 并使用 neverallow 语句限制只允许 mrdloader.execfile 类型对此文件的访问。

libprop2uci.so 的 context 默认是在 file/libfile.cil 文件中配置的，因此可直接在此文件中参考原有的 libfile 做如下扩展，更新默认 context：

```
selinux-policy/src/file/libfile.cil

    "/usr/lib/.*"
    any
    libfile_file_context)
+ (filecon
+   "/usr/lib/libprop2uci.so"
+   any
+   mrddlibfile_file_context)

+
+       (context
+       mrddlibfile_file_context
+       (.u
+       .r
+       mrddlibfile
+       (systemlow
+       systemlow)))
+
+
+ (type
+   mrddlibfile)
+
+ (call .file.lib.obj_type (mrddlibfile)))
```

在完成编译后，能够在 file_contexts 文件中看到该文件的 context 配置。

```
file_contexts:
/usr/lib/libprop2uci.so u:r:file.mrddlibfile
```

然后在 `src/agent/mrdloader.cil` 文件中添加 `neverallow` 语句，只允许 `mrdloader.execfile` 类型的访问：

```

37      (typeattribute
38        not_mrdloader_execfile_typeattr)
39
40      (typeattributeset
41        not_mrdloader_execfile_typeattr
42        (not
43          mrdloader.execfile))
44
45      (allow execfile .file.mrdlibfile (file (execmod)))
46      (neverallow not_mrdloader_execfile_typeattr .file.mrdlibfile (file (execmod)))

```

至此即可实现只允许 `mrdloader.execfile` type 对 `.file.mrdlibfile` 类型的文件拥有 `execmod` 访问权限。

3.3. case3: 修改 rules or 修改 app 的行为 1

```

14978 [ 28.149902] audit: type=1400 audit(67109.965:273): avc: denied { getattr } for pid=1718 c
omm="uci" path="/etc/config/cmdline" dev="overlay" ino=2029 scontext=u:r:uci.subj tcontext=u:r:tmp.fs
tclass=file permissive=0
14979 [ 28.192108] audit: type=1400 audit(67110.015:274): avc: denied { getattr } for pid=1722 c
omm="uci" path="/etc/config/cmdline" dev="overlay" ino=2029 scontext=u:r:uci.subj tcontext=u:r:tmp.fs
tclass=file permissive=0
14980 [ 28.243072] audit: type=1400 audit(67110.065:275): avc: denied { getattr } for pid=1726 c
omm="uci" path="/etc/config/cmdline" dev="overlay" ino=2029 scontext=u:r:uci.subj tcontext=u:r:tmp.fs
tclass=file permissive=0
14981 [ 28.294372] audit: type=1400 audit(67110.105:276): avc: denied { getattr } for pid=1730 c
omm="uci" path="/etc/config/cmdline" dev="overlay" ino=2029 scontext=u:r:uci.subj tcontext=u:r:tmp.fs
tclass=file permissive=0
14982 [ 28.345581] audit: type=1400 audit(67110.155:277): avc: denied { getattr } for pid=1734 c
omm="uci" path="/etc/config/cmdline" dev="overlay" ino=2029 scontext=u:r:uci.subj tcontext=u:r:tmp.fs
tclass=file permissive=0

```

此问题为进程 `uci` 对于 `/etc/config/cmdline` 文件的 `getattr` 操作被拒绝。

查看 `file_contexts` 文件, `/etc/config/` 目录下所有文件默认安全上行文类型都是 `uci.conf` type.

```
/etc/config/*.? u:r:uci.conf
```

但在 `linux shell` 中查看此文件的安全上行文，实际却为 `u:r:tmp.fs`.

```

root@OpenWrt:/# ls -Z /etc/config/
u:r:uci.conf          chl_agent
u:r:tmp.fs            cmdline
...

```

uci.subj 对于 uci.conf file 原本是存在 getattr 的权限的, 但 /etc/config/cmdline context 类型变成了 tmp.fs type, 从而导致访问被拒绝.

```
allow uci.subj uci.conf file append, create, getattr, ioctl, link, lock, open, read, rename, setattr, unlink, write
```

关于 /etc/config/cmdline 文件的来源, 追溯到应该是在 sbin/cmdline2uci 文件中生成:

```
4 UCI_TMP_DIR=/var/config
5 UCI_FILE=cmdline
...
14 while [ -n "$CMD" ]; do
15     KEY=`echo $CMD | cut -d "=" -f1`
16     VALUE=`echo $CMD | cut -d "=" -f2`
17     # If KEY contains "." the uci set will fail
18     uci -c $UCI_TMP_DIR set $UCI_FILE.$KEY=$VALUE 2> /dev/null
19     let COUNTER+=1
20     CMD=`echo "$CMDLINE" | cut -d " " -f$COUNTER`
21 done
22
23 uci -c $UCI_TMP_DIR commit $UCI_FILE
24 mv $UCI_TMP_DIR/$UCI_FILE /etc/config/$UCI_FILE
```

此程序在执行过程中, uci 进程将向 /var/config (也就是 /tmp/config) 目录输出目标文件 cmdline, 而 /tmp/config 的 context 是 tmp.fs type.

然后将 /tmp/config/cmdline 移动到 /etc/config/cmdline, 从而导致 /etc/config/cmdline context type 为 tmp.fs.

这类问题可以通过修改 app 行为的方法来 fix: 将 mv 操作改为 cp 操作, 可保证 /etc/config/cmdline context type 为 uci.conf file, 从而避免问题.

```
target/linux/mmp/base-files/sbin/cmdline2uci 修改

24 cp $UCI_TMP_DIR/$UCI_FILE /etc/config/$UCI_FILE
25 rm $UCI_TMP_DIR/$UCI_FILE
```

此时的 /etc/config/cmdline context 为:

```
root@OpenWrt:/# ls -Z /etc/config/cmdline
u:r:uci.conf file /etc/config/cmdline
```

3.4. case4: 修改 rules or 修改 app 的行为 2

```
27880 [ 11.185729] audit: type=1400 audit(138953.858:8): avc: denied { remove_name } for pid=424
comm="uci" name=".cmdline.uci-mCNDIp" dev="tmpfs" ino=2028 scontext=u:r:uci.subj tcontext=u:r:tmp.fs
tclass=dir permissive=0
27881 [ 11.216796] audit: type=1400 audit(138953.888:9): avc: denied { remove_name } for pid=424
comm="uci" name=".cmdline.uci-mCNDIp" dev="tmpfs" ino=2028 scontext=u:r:uci.subj tcontext=u:r:tmp.fs
tclass=dir permissive=0
```

此问题与 case 3 类似，为 uci 进程对 tmp.fs type 的目录 .cmdline.uci-mCNDIp 的 访问权限问题。

Table C-7. dir Permissions

remove_name:

Remove a hard link from the directory (for example, remove or move a file from a directory).

根据 dev="tmpfs"， 可以直接在 /tmp 目录下搜索此目录并根据 ino number 进行匹配：

```
root@OpenWrt:/# find -name *uci-mCNDIp*
./tmp/config/.cmdline.uci-mCNDIp
root@OpenWrt:/# ls -Z ./tmp/config/.cmdline.uci-mCNDIp
u:r:tmp.fs                               ./tmp/config/.cmdline.uci-mCNDIp

root@OpenWrt:/tmp/config# ls -li .cmdline.uci-mCNDIp
2028 .cmdline.uci-mCNDIp
```

进一步确认出错点为：

```
cmdline2uci 程序

uci -c $UCI_TMP_DIR set $UCI_FILE.$KEY=$VALUE 2> /dev/null
```

由于 name 后缀 \$KEY 每次都是变化的，因此无法直接为其新增 type。

可考虑修改 app 的行为，比如 uci 访问目录设置为非 /tmp or /var 目录，或者新增 uci.subj 对 tmp.fs dir 的 remove_name 权限：

```
agent/uci.cil

(allow subj .tmp.fs (dir (remove_name)))

也可尝试直接添加 rule:
(allow subj .tmp.fs manage_dir)
```

3.5. case5: 其他 avc debug 参考

```
19428 [ 9.622772] audit: type=1400 audit(81436.306:8): avc: denied { search } for pid=197 comm="sh" name="netifd" dev="overlay" ino=1839 scontext=u:r:hotplugcall.subj tcontext=u:r:netifd.miscfile tclass=dir permissive=0
19429 [ 9.713531] audit: type=1400 audit(81436.396:9): avc: denied { search } for pid=200 comm="sh" name="netifd" dev="overlay" ino=1839 scontext=u:r:hotplugcall.subj tcontext=u:r:netifd.miscfile tclass=dir permissive=0
19430 [ 9.768737] audit: type=1400 audit(81436.456:10): avc: denied { search } for pid=203 comm="sh" name="netifd" dev="overlay" ino=1839 scontext=u:r:hotplugcall.subj tcontext=u:r:netifd.miscfile tclass=dir permissive=0
19431 [ 9.823974] audit: type=1400 audit(81436.506:11): avc: denied { search } for pid=206 comm="sh" name="netifd" dev="overlay" ino=1839 scontext=u:r:hotplugcall.subj tcontext=u:r:netifd.miscfile tclass=dir permissive=0
19432 [ 9.880249] audit: type=1400 audit(81436.566:12): avc: denied { search } for pid=209 comm="sh" name="netifd" dev="overlay" ino=1839 scontext=u:r:hotplugcall.subj tcontext=u:r:netifd.miscfile tclass=dir permissive=0
19433 [ 9.935913] audit: type=1400 audit(81436.616:13): avc: denied { search } for pid=212 comm="sh" name="netifd" dev="overlay" ino=1839 scontext=u:r:hotplugcall.subj tcontext=u:r:netifd.miscfile tclass=dir permissive=0
19434 [ 12.564605] audit: type=1400 audit(81439.200:68): avc: denied { search } for pid=483 comm="sh" name="netifd" dev="overlay" ino=1839 scontext=u:r:hotplugcall.subj tcontext=u:r:netifd.miscfile tclass=dir permissive=0
```

根据 log 可知, target type 为 netifd.miscfile

在 linux shell 中搜索 netifd 目录, 根据 log 中 ino= 1839 匹配可得目标 dir 为 “/lib/netifd”.

```
root@OpenWrt:/# ls -li /lib/ | grep netifd
1839 netifd

root@OpenWrt:/# ls -Z /lib/ | grep netifd
u:r:netifd.miscfile netifd
```

参考 file_contexts 文件, 安全上行文中 type 处于 hotplugcall namespace 的文件只有 hotplug-call, 实际对应 ./sbin/hotplug-call 文件.

```
hotplug-call 文件:

14 if [ \! -z "$1" -a -d /etc/hotplug.d/$1 ]; then
15     for script in $(ls /etc/hotplug.d/$1/* 2>&-); do (
16         [ -f $script ] && . $script
17     ); done
18 fi
```

此文件遍历 /etc/hotplug.d/ 目录下的某个子目录, 依次执行脚本.

搜索 /etc/hotplug.d/目录下对于 /lib/netifd 目录的访问, 可大致确定被拒绝的操作可能是:

```
ethernet/00-ema:28:          udhcpc -p /var/run/udhcpc-eth0.pid -s /lib/netifd/dhcp.script -f  
-t 0 -i eth0 -C &  
ethernet/00-ema:188:        udhcpc -p /var/run/udhcpc-eth0.1.pid -s /lib/netifd/dhcp.script -  
f -t 0 -i eth0.1 -C &
```

如果判断此操作是期望被允许的，可参考如下方式配置 **rules** 赋予访问权限：

```
src/agent/hotplugcall.cil  
  
(call .netifd.search_miscfile_dirs (subj))
```

上述仅列举了几个 **case** 的处理结果,其他 **avc denied** 问题可按照相同的方法具体分析.