LSM

	UT-RD-SXXX		1.0.0
	2021-9-13		

0.0.1			2021-08-16
0.0.2			2021-08-16
0.0.3			2021-08-22
0.0.4			2021-08-26
0.0.5			2021-09-13

1

LSM : Linux Secrity Module, Linux Linux

DAC: Discretionary Access Control I/O

MA C: Mandatory Access Control ,

VFS Posix

AppArmor :
FileArmor : FileArmor

DF A
eHF A: extened Hybrid Finite Automata, DFA

2

root

root

1.
2.

3

chattr

root

1: chattr

test.c chattr

chattr

1 chattr

2 chattr

chattr

chattr

LSM SELinux AppArmor

SELinux AppArmor

LSM

4

4.1

FileArmor FileArmor

FileArmor

FileArmor

2:

filearmor_parser, libfilear-
mor,

/etc/filearmor.d

libapparmor

filearmor_parser sysfs

filearmor module LSM hook sysfs

eHFA hook

4.2

4.2.1

FileArmor

filearmor_parser flex bison

uid

3

DFA DFA

eHFA table eHFA table sysfs

eHFA table

eHFA eHFA

eHFA table eHFA

3:

```
<filepath> <own app>[ ] <version >[ ] { #
        <action > < uid list > <app list > <permission >, #
        <action > < uid list > <app list > <permission >,
        ...
}
```

<filepath> , hometest.sshrsa_k ey,

<o wn app> FileAr-
mor

<v ersion> 1.0

<action> allow deny allow deny

<uid list> uid * 0, 1000

<app list> * /usr/bin/wps,
/usr/bin/hello

<p ermission> r|w|x|d

#

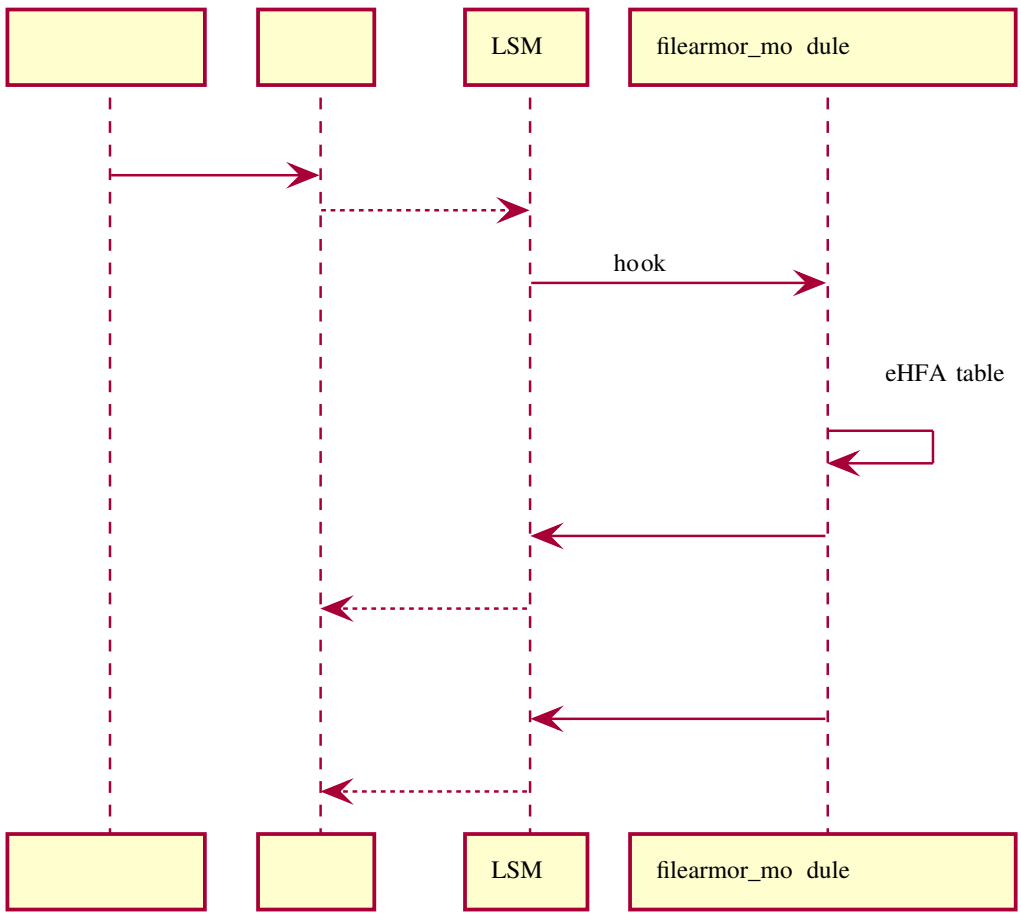
```
/ home/ test / test . doc ă/ usr / bin / vimă1.0 ă{
        allow ă{*}ă{/ usr / bin / cat }ăă r ,
        allow ă{1000 }ă{/ usr / bin / wps, / usr / bin / office }ăă rw,
}

/ home/ test / . ssh / rsa_key ă{
        deny ă{1000 }ă{/ usr / bin / wps, / usr / bin / office }ă w,
}
```

4.2.2 ho ok

LSM

LSM Linux Kernel MAC Hook LSM
Linux Kernel
LSM

inode(socket) file( )
LSM
Hook
filearmor module LSM hook hook
eHFA table
0 uid



4:

1: hook

	Ho ok	
	path_rename	
	path_truncate	
	file_op en	
	path_rmdir/ino de_rmdir	
	path_mkno d/ino de_mkno d	
	path_mkdir/ino de_mkdir	
	file_p ermission	
	path_unlink/ino de_unlink	

hook 0

hook 0 FileArmor

5

5.1

AppArmor SELinux Smack TOMO YO

SELinux AppArmor LSM securit y

```
1 struct file void *f_security ;
2 struct ăcred void *security ;
3 struct ăinode void *i_security ;
4 struct ătask_struct void *security ;
```

5: FileArmor

FileArmor Linux securit y module FileArmor

Linux SECURITY_FILEARMOR 5

1 FileArmor

2 FileArmor boot 0 1

AppArmor FileArmor 6

6: AppArmor

5.2

D rule-exprs expr-stats... compressed-dfa test

```
sudo ./filearmor_parser "-D" "rule-exprs" "-D" "expr-stats" "-D" "expr-tree"
, "-D" "expr-simplified" "-D" "stats" "-D" "progress" "-D" "dfa-states" "-D"
, "dfa-graph" "dfa-minimize" "-D" "dfa-unreachable" "-D" "dfa-node-map" "-D"
, "dfa-uniq-perms" "-D" "dfa-minimize-uniq-perms" "-D"
, "dfa-minimize-partitions" "-D" "compress-progress" "-D" "compressed-dfa"
, test
```

2:

expr-tree	dfa
expr-simplified	dfa
dfa-states	DFA
rule-exprs	
dfa-unreac hable	
dfa-no de-map	dfa
dfa-minimize-partitions	dfa
compressed-dfa	dfa

7:

1-1464/27648*100%=94.7%,

5.3

uid 1000 /usr/bin/rm /home/test/test

```
/home/test/test ă{
        deny ă{1000}ă{/usr/bin/rm}ăd,
}
```

hook

```
//
int check_inode_rmdir(struct inode *dir, struct dentry *dentry)
{
    if (unlikely(dentry == NULL)) {
        pr_err("[%s] [%s] dentry is NULL!", MODULE_NAME,
        __func__);
        return 0;
    }

    char *kbuf = kmalloc(PATH_MAX,GFP_KERNEL);
    if (unlikely(kbuf == NULL)) {
        pr_err("[%s] [%s] kmalloc fail!", MODULE_NAME,
        __func__);
        return -ENOMEM;
    }
    char *raw_path = dentry_path_raw(dentry, kbuf, PATH_MAX);
    if (IS_ERR(raw_path)) {
        kfree(kbuf);
        pr_err("[%s] [%s] failed to get dentry_path_raw",
        MODULE_NAME,__func__);
        return -ENOMEM;
    }

    char *app_path = executable_path(current);
    if (0 != strcmp(app_path,
    "/usr/lib/systemd/systemd")){

        printk("app_path = %s n", app_path);
        int process_owner_uid = (current->cred)->uid.val;
        if (0 == strcmp(raw_path, "/home/wxl/test")){

        //ă id,ă id 1000
        if (ăcurrent_uid().val ă!=ă1000)ăă{
            kfree(kbuf);
            return ă0;
        }
        printk("ă[%s]ăuidă=ă%d", ă__func__, ăcurrent_uid().val);

        if (0 == strcmp(app_path, "/usr/bin/rm")){
            printk("check current uid = %d can't delete
            file n", current_uid().val);
            return -EPERM;
        }
    return 0;
```

```
                }
        }
        return  0;

}
```

8:

5.4

uid 1000 /usr/bin/more /home/test/test.c

```
/ home/ wxl/ test . că{
        allow ă{ 1000 }ă{ / usr / bin / more }ăr ,
}
```

hook

```
//
int  check_file_permission  ( struct  file  *  file  ,  int  mask)
{

        char *ăkbuf ;
        char *ăraw_path ;

        if ă( ăunlikely  ( file  ă==ăNULL )ă) ă{
                pr_err  ( "ă[%s]ădentryăisăNULL!"  , ăă __func__ );
                return  ă0;
                return  ă0
        }

        //ă              0
        kbuf ă=ăkmalloc ( PATH_MAX ,ăGFP_KERNEL );
```

```
16              if ă( ăunlikely   ( kbuf ă==ăNULL)ă) ă{
17                      pr_err  ( "ă[%s]ăkmallocăfailed!"       , ă__func__ );
18                      return  ă- ENOMEM;
19              }
20
21              raw_path ă=ăd_path ( ă&( file   ->f_path  ), ăkbuf , ăPATH_MAX);
22              if ă( ăIS_ERR( raw_path  ) ă) ă{
23                      kfree  ( kbuf );
24                      return  ă0;
25              }
26
27              //ă
28              if  ( ă0ă!= ăstrncmp  ( ăraw_path  , ă"/home/wxl/test"      , ăstrlen   ( "/home/wxl/test"      )) ă) ăă{
29                      kfree  ( kbuf );
30                      return  ă0;
31              }
32
33              //ă              idă         id         1000
34              if  ( ăcurrent_uid    ().  val ă!= ă1000) ăă{
35                      kfree  ( kbuf );
36                      return  ă- EPERM;
37              }
38              printk   ( "ă[%s]ăuidă=ă%d"   , ă__func__ , ăcurrent_uid    ().  val );
39
40              //ătest.c                 /usr/bin/more
41              char *ăkbuf2 ă=ăkmalloc  ( PATH_MAX, ăGFP_KERNEL);
42              char   *app_path   =  executable_path    ( current   );
43              if  ( ă0ă==ăstrcmp  ( ăapp_path  , ăă"/usr/bin/more"      ) ă) ă{
44                      kfree  ( kbuf );
45                      kfree  ( kbuf2 );
46                      printk   ( "check   current    uid   =  %d App =  %s can't   delete    path   =  %s
                        ,     n ",  process_owner_uid     , current   ->comm, raw_path  );
47                      return  ă0;
48              }
49
50              kfree  ( kbuf );
51              kfree  ( kbuf2 );
52              return  ă0;
53      }
```

```
1       cat ătest  . c
2       cat : ătest  . txt     Operation   ănot ăpermitted
```

```
1       $  moreătest  . c
2       hello    world
```

6

LSM hook

preload hook

7

LSM [2] [3] [4] [1]

[1] Alfred V Aho, Monica S Lam, Ravi Sethi, and Jeffrey D Ullman. Compilers: principles, techniques and tools. 2020.

[2] Chris Wright, Crispin Cowan, James Morris, Stephen Smalley, and Greg Kroah-Hartman. Linux security module framework. In Ottawa Linux Symposium, volume 8032, pages 616. Citeseer, 2002.

[3] . Linux . , 2004.

[4] , , , et al. Linux . , 2008.