Access Control Systems Lab
Sommersemester 2017

Praktikumsbericht 2

Jean-Marc Hendrikse - 1751591

Prof. Dr. Hannes Hartenstein, Alexander Degitz, Jan Grashöfer, Till Neudecker
Forschungsgruppe Dezentrale Systeme und Netzdienste
Karlsruher Institut für Technologie (KIT)

Introduction

As mentioned in the last lab protocol Linux Security Modules (LSM) is a Framework that allows various implementations of a Reference Monitor for Linux. We dived into one of these implementations - the Security-Enhanced Linux (SELinux) and used it as a Reference Monitor. In this paper we are going to build our own LSM. We will learn how to compile the LSM into the kernel on which we will work.

Preparations

The preparation part for writing our own LSM is the most important part because during development process it will helped you to come back to these steps and rebuild the kernel if you change the krenel code. Because it is so important I would like to summarize what you should do for preparation:

1. First of all we installed necessary dependencies via `sudo dnf install openssl-devel elfutils-libelf-devel`
2. Afterwards we cloned the Linux source tree via `git clone -b acslab -single-branch https://<user>@git.kit.edu/dsn-stud-projects/acs-lab/linux.git`
3. In the next steps we followed the guide at `https://fedoraproject.org/wiki/Building_a_custom_kernel#Building_Vanilla_upstream_kernel`
4. We copied the existing system config file `cp /boot/config-'uname -r'* .config` into our linux directory.
5. After the configuration we started to build the kernel.
   ```
   $ make oldconfig
   $ make bzImage
   $ make modules
   ```
   (become root)
   ```
   # make modules_install
   # make install
   ```

Note: Every single step from above is detailed described in the task sheet of this lab lesson and can be read at `https://fedoraproject.org/wiki/Building_a_custom_kernel#Building_Vanilla_upstream_kernel`

1 Understanding the ACS-Lab LSM template

After checking out the repository from preparation section we get a playground for writing a LSM called "acslab". In order to see the necessary changes between last commit and the kernel sources before Version 4.9 to build a LSM we use the command `git diff v4.9`.

```
[student@DSN-ACSLab-Master linux]$ git diff v4.9
diff --git a/Makefile b/Makefile
index b103777..424aa0e 100644
--- a/Makefile
+++ b/Makefile
@@ -1,7 +1,7 @@
 VERSION = 4
 PATCHLEVEL = 9
 SUBLEVEL = 0
-EXTRAVERSION =
+EXTRAVERSION = jean
 NAME = Roaring Lionus

 # *DOCUMENTATION*
diff --git a/include/linux/lsm_hooks.h b/include/linux/lsm_hooks.h
index 558adfa..5bccd2f 100644
--- a/include/linux/lsm_hooks.h
+++ b/include/linux/lsm_hooks.h
@@ -1933,5 +1933,10 @@ void __init loadpin_add_hooks(void);
 #else
 static inline void loadpin_add_hooks(void) { };
 #endif
+#ifdef CONFIG_SECURITY_ACSLAB
+extern void __init acslab_add_hooks(void);
+#else
+static inline void __init acslab_add_hooks(void) { }
+#endif

 #endif /* ! __LINUX_LSM_HOOKS_H */
diff --git a/security/Kconfig b/security/Kconfig
index 118f454..656ac24 100644
--- a/security/Kconfig
+++ b/security/Kconfig
@@ -164,6 +164,7 @@ source security/tomoyo/Kconfig
 source security/apparmor/Kconfig
 source security/loadpin/Kconfig
 source security/yama/Kconfig
+source security/acslab/Kconfig

 source security/integrity/Kconfig

diff --git a/security/Makefile b/security/Makefile
index f2d71cd..6f9865b 100644
--- a/security/Makefile
+++ b/security/Makefile
@@ -9,6 +9,7 @@ subdir-$(CONFIG_SECURITY_TOMOYO)        += tomoyo
 subdir-$(CONFIG_SECURITY_APPARMOR)      += apparmor
 subdir-$(CONFIG_SECURITY_YAMA)          += yama
 subdir-$(CONFIG_SECURITY_LOADPIN)       += loadpin
+subdir-$(CONFIG_SECURITY_ACSLAB)        += acslab CONFIG_SECURITY_ACSLAB = y
```

```
 # always enable default capabilities
 obj-y                                  += commoncap.o
@@ -24,6 +25,7 @@ obj-$(CONFIG_SECURITY_TOMOYO)          += tomoyo/
 obj-$(CONFIG_SECURITY_APPARMOR)                += apparmor/
 obj-$(CONFIG_SECURITY_YAMA)            += yama/
 obj-$(CONFIG_SECURITY_LOADPIN)         += loadpin/
+obj-$(CONFIG_SECURITY_ACSLAB)          += acslab/
 obj-$(CONFIG_CGROUP_DEVICE)            += device_cgroup.o

 # Object integrity file lists
diff --git a/security/acslab/Kconfig b/security/acslab/Kconfig
new file mode 100644
index 0000000..b217e9a
--- /dev/null
+++ b/security/acslab/Kconfig
@@ -0,0 +1,6 @@
+config SECURITY_ACSLAB
+       bool "ACS-Lab LSM support"
+       depends on SECURITY_PATH
+       default n
+       help
+         This module contains an exemplary LSM for the ACS-Lab.
diff --git a/security/acslab/Makefile b/security/acslab/Makefile
new file mode 100644
index 0000000..9bcfedb
--- /dev/null
+++ b/security/acslab/Makefile
@@ -0,0 +1 @@
+obj-$(CONFIG_SECURITY_ACSLAB) += acslab.o
diff --git a/security/acslab/acslab.c b/security/acslab/acslab.c
new file mode 100644
index 0000000..821673f
--- /dev/null
+++ b/security/acslab/acslab.c
@@ -0,0 +1,72 @@
+/*
@@ -0,0 +1,72 @@
+/*
+ * Playground module for the LSM framework.
+ *
+ */
+
+#define pr_fmt(fmt) "ACS-Lab: " fmt
+
+#include <linux/lsm_hooks.h>
+#include <linux/time64.h>      // timespec64
+#include <linux/time.h>                // timezone
+#include <linux/path.h>                // path
+#include <linux/dcache.h>      // dentry_path
+#include <linux/string.h>      // strnstr
+#include <linux/usb.h>         // USB
+
+/*** Hook handler definitions ***/
+
+/*
+#define VENDOR_ID 0x0000
+#define PRODUCT_ID 0x0000
+
+static int match_usb_dev(struct usb_device *dev, void *unused)
+{
+       return ((dev->descriptor.idVendor == VENDOR_ID) &&
+               (dev->descriptor.idProduct == PRODUCT_ID));
+}
```

```
+
+static int acslab_path_mkdir(const struct path *dir, struct dentry *dentry,
    umode_t mode)
+{
+       char buf_dir[256];
+       char buf_path[256];
+       char *ret_dir;
+       char *ret_path;
+
+       ret_dir = dentry_path(dir->dentry, buf_dir, ARRAY_SIZE(buf_dir));
+       if (IS_ERR(ret_dir)) {
+               pr_info("mkdir hooked: <failed to retrieve directory>\n");
+               return 0;
+       }
+
+       ret_path = dentry_path(dentry, buf_path, ARRAY_SIZE(buf_path));
+       if (IS_ERR(ret_path)) {
+               pr_info("mkdir hooked: <failed to retrieve path>\n");
+               return 0;
+       }
+
+       pr_info("mkdir hooked: %s in %s\n", ret_path, ret_dir);
+
+       // Add your code here //
+
+       return 0;
+}
+*/
+
+static int acslab_settime (const struct timespec64 *ts, const struct timezone
    *tz)
+{
+       pr_info("settime hooked\n");
+       return 0;
+}
+
+/*** Add hook handler ***/
+
+static struct security_hook_list acslab_hooks[] = {
+       //LSM_HOOK_INIT(path_mkdir, acslab_path_mkdir),
+       LSM_HOOK_INIT(settime, acslab_settime),
+};
+
+void __init acslab_add_hooks(void)
+{
+       pr_info("Hooks have been added!");
+       security_add_hooks(acslab_hooks, ARRAY_SIZE(acslab_hooks));
+}
diff --git a/security/security.c b/security/security.c
index f825304..a99b356 100644
--- a/security/security.c
+++ b/security/security.c
@@ -61,6 +61,7 @@ int __init security_init(void)
        capability_add_hooks();
        yama_add_hooks();
        loadpin_add_hooks();
+       acslab_add_hooks();
```

Listing 1: git diff v4.9 output

- As you can see in lisitng 3 we configured `EXTRAVERSION` and set it to `jean`.
- In the LSM hooks *linux/lsm_hooks.h* an if-else definition is appanded which defines that `extern void __init acslab_add_hooks(void);` is called if `CONFIG_SECURITY_ACSLAB`

is set and if not we jump into the `else` clause and execute `static inline void __init acslab_add_hooks(void) { }`

- In the security configuration file with path *security/Kconfig* a source path of our acslab security config file was added *source security/acslab/Kconfig*.
- Subdirectory acslab was created because we set CONFIG_SECURITY_ACSLAB=y: subdir-$(CONFIG_SECURITY_ACSLAB) += acslab in *security/Makefile*.
- The newly created security config is filled with content and definitions that configures the Security ACS-Lab.
- The linked object file is defined in the acslab Makefile.
- The acslab C file for kernel implementation will be edited in the next sections. For now it serves a framework with all necessary definitions and methods we implement later.
- Also in security.c the acslab_add_hooks() was appended

- ACS-Lab LSM is a minor LSM.

2 Exploring the ACS-Lab LSM

After we compiled the kernel successfully and examined all necessary changes to the LSM framework to configure our own LSM, we will start to explore the ACS-Lab LSM. To see what the ACS-Lab LSM does we have to look into the kernel logs. Every access decisions by the LSM is logged right here. To get those messages containing the prefix *ACS-Lab* we just use the command `jorunalctl`

```
$ journalctl -k | grep 'ACS-Lab'
May 23 15:46:35 DSN-ACSLab-Master kernel: ACS-Lab: Hooks have been added!
May 23 15:46:38 DSN-ACSLab-Master kernel: ACS-Lab: settime hooked
```

Listing 2: git diff v4.9 output

Lets change the system time to see what happens in the log messages. For changing the system time we use the command `timedatectl` and set as parameter a date and time, e.g. your birthday:

```
$ sudo timedatectl set-time '1990-09-10 18:00:00'

[student@DSN-ACSLab-Master ~]$ journalctl -k | grep "ACS-Lab"
May 23 15:46:35 DSN-ACSLab-Master kernel: ACS-Lab: Hooks have been added!
May 23 15:46:38 DSN-ACSLab-Master kernel: ACS-Lab: settime hooked
Sep 10 18:00:00 DSN-ACSLab-Master kernel: ACS-Lab: settime hooked
```

Listing 3: git diff v4.9 output

3 Preventing the Creation of Directories

In this section we would like to do some fancy things with the kernel like preventing the user to create a certain directory with name *dsn*. We will only permitt the user with a physical access key such as a 'magic' USB device which should be attached to the system. In order to do so we have to edit the code in `acslab.c`:

```c
/*
 * Playground module for the LSM framework.
 *
 */

#define pr_fmt(fmt) "ACS-Lab: " fmt

#include <linux/lsm_hooks.h>
#include <linux/time64.h> // timespec64
#include <linux/time.h>         // timezone
#include <linux/path.h>         // path
#include <linux/dcache.h> // dentry_path
#include <linux/string.h> // strnstr
#include <linux/usb.h>          // USB
/*** Hook handler definitions ***/


#define VENDOR_ID 0x090c
#define PRODUCT_ID 0x1000

static int match_usb_dev(struct usb_device *dev, void *unused)
{
        return ((dev->descriptor.idVendor == VENDOR_ID) &&
                (dev->descriptor.idProduct == PRODUCT_ID));
}

static int acslab_path_mkdir(const struct path *dir, struct dentry *dentry,
   umode_t mode)
{
        char buf_dir[256];
        char buf_path[256];
        char *ret_dir;
        char *ret_path;

        ret_dir = dentry_path(dir->dentry, buf_dir, ARRAY_SIZE(buf_dir));
        if (IS_ERR(ret_dir)) {
                pr_info("mkdir hooked: <failed to retrieve directory>\n");
                return 0;
        }

        ret_path = dentry_path(dentry, buf_path, ARRAY_SIZE(buf_path));
        if (IS_ERR(ret_path)) {
                pr_info("mkdir hooked: <failed to retrieve path>\n");
                return 0;
        }

        pr_info("mkdir hooked: %s in %s\n", ret_path, ret_dir);

        if (strcmp("dsn", dentry->d_name.name) == 0) {
                if(usb_for_each_dev(NULL,match_usb_dev)) {
                 pr_info("mkdir hooked: <magic usb detected. permission for
   creating dsn granted>\n");
                 return 0;
                }
                pr_info("mkdir hooked: <failed to create directory with
  name dsn>\n");
                return 1;
        }
        pr_info("mkdir hooked: <creating directory permitted>\n");
        return 0;
}

static int acslab_settime (const struct timespec64 *ts, const struct
  timezone *tz)
```

```
61    {
62            pr_info("settime hooked\n");
63            return 0;
64    }
65
66  /*** Add hook handler ***/
67
68  static struct security_hook_list acslab_hooks[] = {
69    LSM_HOOK_INIT(path_mkdir, acslab_path_mkdir),
70    LSM_HOOK_INIT(settime, acslab_settime),
71  };
72
73  void __init acslab_add_hooks(void)
74  {
75    pr_info("Hooks have been added!");
76    security_add_hooks(acslab_hooks, ARRAY_SIZE(acslab_hooks));
77  }
```

Listing 4: Implemented acslab_path_mkdir() acslab.c

In listing 4 we have that one function that controls the permissions to create directories called acslab_path_mkdir(). Because of LSM_HOOK_INIT() in line 69 `acslab_path_mkdir()` implements `path_mkdir()`. We get more information about the `path_mkdir()` function in the hooks file *linux/lsm_hooks.h* that describes the functionality of that function: It is for permission control for creating new directories in an existing directory. The first argument is the path structure of parent of the new directory, the second argument contains the dentry structure of the directory to be created and it returns 0 if the creation is granted and anything different from 0 if denied. Let's jump to the `acslab_path_mkdir()`. We recognize two calls to the `dentry_path` function. The two functions have in common that they return the path of the dentry relative to their mounting point of the filesystem. The `ret_dir` is the existing path and the `ret_path` it the one we would like to create. Only if these dentries are contained in the filesystem this function returns an absolute path. [1] As arguments the function expects the `dentry` of a file to look up, a buffer that stores the path and the length of the buffer. The path is built from the end of the buffer to the beginning.

In line 18, 19 of listing 4 there is the `VENDOR_ID` and `PRODUCT_ID` defined. To get the IDs we should first plug in the USB device. To display information about USB buses in the system and the devices connected to them we use the command `lsusb` [2].

```
[student@DSN-ACSLab-Master acslab]$ lsusb
Bus 001 Device 003: ID 090c:1000 Silicon Motion, Inc. - Taiwan (formerly Feiya
    Technology Corp.) Flash Drive
Bus 001 Device 002: ID 80ee:0021 VirtualBox USB Tablet
Bus 001 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
```

Listing 5: Display USB information by `lsusb`

In the print out of listing 5 the desired USB has the ID `090c:1000`. Everything before the colon is the `Vendor ID` and the number after the colon is `Product ID`. To define the IDs in our code from listing 4 we take the hexadecimal numbers from `Vendor ID` and `Product ID`. From now on the `match_usb_dev()` function will match only our 'magic' device and return false if the IDs dismatch.
Next we will check if the user tries to create a directory named *dsn*. We do this by comparing the String '`dsn`' with the name of the dentry. To get the dentry name we just write `dentry->d_name.name`. If the two strings are identical the string compare function returns 0 and we check whether the 'magic' USB is plugged in or not. To check if any of the USB devices in our system matches our 'magic' USB we use `usb_for_each_dev()` function. That function iterates over USB devices that are connected to the system and takes as argument a callback function.[3] If there is no device connected to the system we return everything different from

0. If our USB device was detected the we return 0. Any directory name other than *dsn* will be permitted to be created.

4 Conclusion

To check if our implementation works we will execute the `mkdir` linux command once without the 'magic' USB:

```
[student@DSN-ACSLab-Master linux]$ mkdir dsn
mkdir: cannot create directory dsn: Operation not permitted
[student@DSN-ACSLab-Master linux]$ journalctl -k | grep "ACS-Lab"
May 30 10:50:31 DSN-ACSLab-Master kernel: ACS-Lab: mkdir hooked: /home/student/
    Desktop/linux/dsn in /home/student/Desktop/linux
May 30 10:50:31 DSN-ACSLab-Master kernel: ACS-Lab: mkdir hooked: <failed to
    create directory with name dsn>
```

After we attached the 'magic' USB device to the system we got no message and the directory was created. Looking at system journal we can see our message for granting permission:

```
May 30 10:51:58 DSN-ACSLab-Master kernel: ACS-Lab: mkdir hooked: /home/student/
    Desktop/linux/dsn in /home/student/Desktop/linux
May 30 10:51:58 DSN-ACSLab-Master kernel: ACS-Lab: mkdir hooked: <magic usb
    detected. permission for creating dsn granted>
```

Listing 6: Display USB information by `lsusb`

With a given frame and added configurations and files it is pretty simple to write a LSM that controls access and permissions. By implementing functions that hooks into system functions it is possible to implement a very strong Reference Monitor.

Literatur

[1] Stackoverflow - How to Use dentry_path_raw(),`https://stackoverflow.com/questions/33249643/how-to-use-dentry-path-raw`, visited on 28/05/2017.

[2] Archlinux-Wiki: lsusb,`https://wiki.archlinux.de/title/Lsusb`, visited on 28/05/2017.

[3] enion.net - API usb for each dev,`http://einon.net/DocBook/usb/API-usb-for-each-dev.html`, visited on 29/05/2017.