Embedded Alley Solutions, Inc. SMACK for Digital TV

Revision 1.2 • September 21, 2009

Embedded Alley Solutions, Inc • 2055 Junction Avenue, Suite 125 • San Jose, California 95131 • +1 408 577 1884 • www.embeddedalley.com

Table of Contents

Acknowledgement

This paper was derived from work sponsored by CE Linux Forum. Embedded Alley would like to acknowledge and thank CELF for their continued support of the development and enhancement of Linux-based embedded devices.

Introduction

Modern consumer electronic products (mobile phones, set top boxes, digital TVs, etc) implement complex software solutions that usually include the entire software stack from operating system on a bottom level, to end user applications accessing system resources through middle ware abstraction layers. In embedded systems most of available user space applications are intended to implement a device function, such as capturing, handling and showing video stream on TV screen in case of digital TV device. These applications are normally implemented either by the device vendors or their partners that are considered as trusted sources.

In addition to vendor or partner applications, there is another group called third-party applications. These are developed by third parties (e.g. independent software companies or developers, end users) using a development platform (not necessarily open) provided by vendors. The main goal of third-party applications are to extend existent platforms with more features that are not directly related to main device function. And providing a development platform enables end users to develop or select from a variety of existent application choices customizing their devices for personal needs.

One of the aspects of supporting third-party applications is to create a development platform that exposes all necessary interfaces to interact with underlying hardware and users. However, since third-party applications can not be considered as trusted, there has to be a way to control access to various system resources to prevent them from performing illegal actions, such as accessing protected data, destroying system information, distributing TV stream and so on. In other words, third-party applications should be run in “sandbox” that exactly defines what actions third-party applications are allowed to perform.

The problem of access control of third-party applications is even more serious in embedded products based on embedded Linux where all system and user applications normally run as a root user with UID 0, which by default has all privileges and capabilities to access any resources and to perform any actions.

One of the possible ways to solve this problem in Embedded Linux is to move away from the conventional way of granting user privileges based on user discretion to another mechanism already implemented in a Linux kernel, such as Linux security module (LSM). LSM implements a mandatory control access approach based on a set of rules enforced by the system instead of a particular user. Currently, there are three LSMs available in Linux kernel – Security-Enhanced Linux (SELinux), Simple Mandatory Access Control (SMACK) and TOMOYO Linux, where SELinux is a

desktop/server LSM and others are oriented towards embedded systems.

This paper describes the results of applying SMACK LSM to a Phillips Linux based Digital TV Split Application Architecture (SPACE). It starts from an overview of the platform and how applications interact with it. And process to provide basic use-cases for third-party applications and their access control requirements defined by Phillips SPACE developers. Then a SMACK rule set to address the requirements will be proposed and verification process will be described, supplemented with results of analysis of SMACK impact to system memory consumption and performance.

Also in this paper there is a discussion on how SMACK by itself, is not enough to guarantee that all the requirements are met. The paper will propose a solution consisting of a hybrid of several Linux security and control mechanisms, such as SMACK, Capabilities and C-groups with SMACK playing a major role.

SMACK overview

SMACK stands for Simple Mandatory Access Kernel. It is an alternative Linux Security Module providing simple configuration interface and smaller memory footprint than its predecessors, which makes it suitable for embedded systems based on Linux. This section is a brief overview of SMACK.

Since SMACK is a mandatory control access (MAC) mechanism it operates with standard MAC terms, like:

- Subject
  On SMACK subjects are task running in the system.

- Object
  Objects are passive entities in a system. On SMACK objects are files of any type, native Linux IPC objects and tasks in a sense when they are accessed by other tasks (e.g. one task sends a signal to another)

- Access
  Any attempt by a subject to put on or get any information from an object

SMACK performs access control based on labels assigned to objects and subjects and a set of the following rules:

1. Any access requested by a task labeled "*" is denied.
2. A read or execute access requested by a task labeled "^" is permitted.
3. A read or execute access requested on an object labeled "_" is permitted.

4. Any access requested on an object labeled "*" is permitted.
5. Any access requested by a task on an object with the same label is permitted.
6. Any access requested that is explicitly defined in the loaded rule set is permitted.
7. Any other access is denied.

The sixth rule implies having a special SMACK rule set where subjects and objects relationships are explicitly defined in the following format:

subject-label object-label access

where ***access*** is a combination of 'r' (read), 'w' (write), 'x'(execute) and 'a'(append)

SMACK labels are strings of no more than 23 characters. They can be kept in extended file system attributes for files, SMACK configuration files or, are inherited from the object owner for dynamic objects like shared memory.

SPACE digital TV platform

Overview

Split application architecture (SPACE) is a software platform offered by Philips for their digital TV products. SPACE platform is based on the following open source components:

- Linux kernel (2.6.x)
- DirectFB 1.2
- SaWMan
- FusionDale

DirectFB

Direct FB library provides hardware graphics acceleration, input device handling and abstraction, integrated windowing system with support for translucent windows and multiple display layers. All user applications running in SPACE are DirectFB applications that create DirectFB window and draw into it.

SaWMan

Shared application and window manager (SaWMan) is a custom DirectFB window manager module that allows one process hooked to SaWMaN to be an application and window manager. An application manager controls life cycle of other processes in the system and their appearance on a screen including window layout and focus management.

FusionDale

FusionDale is an application of DirectFB Fuison library that provides high-level inter-process communications (IPC) mechanisms. DirectFB Fusion is based on shared memory where all IPC objects are stored and accessed by different applications. Shared memory is implemented via memory mapped files located in temporary RAM-based file system (tmpfs).

SPACE architecture

SPACE consists of the following building blocks:

- Application Manager (amApp)
- Platform application (pflApp)
- Plarform API (pflAPI)
- Television application (tvApp)

Application manager

The application manager is responsible for the other application's life cycles and their windows layout. It decides in which DirectFB virtual layer a window will be shown and defines window position and size on a screen. All other DirectFB applications create and draw into DriectFB window but do not control how the window will be rendered on a screen. The application manager is hooked into SaWMan.

Platform application and API

Platform application implements a hardware platform abstraction layer for all other applications running in the system. It offers access to different hardware resources, such as TV tuner, audio, video and graphics control, via platform API wrapped around FusionDale. The system module of DirectFB is a part of platform application.

Television application

Television application implements core TV functionality. It allows users to set general TV settings, browse TV channels, read teletext and so on.

There are also other user applications that include media application to play video and audio files, web browser application currently based on Opera, user manual application allowing user to browse through TV manual and home application used by users to start other applications. All of these components are supplied by Philips and, thus, do not require any access control.

Open SPACE and third-party applications

Overview

Open SPACE is a Philips SPACE platform open for external or third-party applications development. It enables third-parties to develop DirectFB applications using platform API. These applications are not intended to implement core TV functionality, which is completely covered by vendor applications. Instead, they extend digital TV with new features, such as entertainment or multimedia capabilities.

However, in current SPACE implementation there is no explicit edge between vendor and third-party applications. Practically it means that third-party applications running in the system get the same set of privileges as vendor applications. It leaves an opportunity for external applications to abuse of privileges and perform undesired actions like damaging TV or accessing any content that require special licensing.

Distributing open SPACE platform to different third-parties does not leave any possibility to control each possible application separately due to a constantly growing number of applications. Thus, the problem of access control has to be solved in general for all external applications by using system-wide control mechanisms. To define a set of privileges available to third-party applications an analysis of external applications use cases is required.

Third-party application types

Third-party applications in SPACE are basically DirectFB applications that create a DirectFB window and draw into it. Even though every application is different, depending on their functionality they can fall to different groups. And each group determines the specific system resources the applications will need to access. Three different examples of third-party application types are described below:

- Content viewers
- Entertainment applications
- Internet services

Content viewers

Content viewers applications display data stored in various formats. The examples include audio/video and flash players, pictures and document viewers.

Specific requirements:

- Access to pluggable media – USB sticks and external hard drives, SD/MMC cards
- Access to data partition shared among all the applications in the system

Entertainment applications

This group of third-party applications includes games of various types. They could be characterized by close and intensive interaction with users.

Specific requirements:

- Access to hardware acceleration resources
- Access to multiple input devices that may not be directly handled by vendor software (a remote control device oriented)

Internet services

In additional to general-purpose web browser supported by SPACE, Internet services applications are dedicated to particular Internet resources providing a digital TV oriented (tuned for display and control specifics) interface to them. Such applications may include weather application, YouTube player, MySpace portal viewer and many other examples.

Specific requirements:

- Network access

Third-party application use-cases

Below are two use-cases for third-party application requirements. First is an example of a simple game controlled using the TV remote control. As an example, it could be a Tetris game. Second application is a simple Internet service displaying current weather.

Simple game application

- Launched by application manager
- Reads local configuration
- Reads saved data. For example last passed game level.

- Creates DirectFB window
- Listens to input events
- Performs necessary computations
- Draws to the DriectFB window
- Saves data and configuration
- Exits per user request

Weather application

- Launched by application manager
- Creates DirectFB window
- Sets up network connection with remote weather server
- Periodically reads data from the server
- Draws weather information to the DirectFB window
- Exits per user request

Third-party applications access control requirements

Below is a list of access control requirements defined by Philips SPACE developers that should be met in order to define a sandbox for third-party applications running in SPACE environment:

- No access to certain device nodes

  There is a list of devices (e.g. mtd, i2c) that external applications should never have an access to. Other devices (e.g. USB input devices) may be accessed by third-party applications because they are not controlled by SPACE or just required for normal application work (e.g. /dev/urandom)

- No ability to create device node

  This is necessary to prevent external applications from creating device nodes in their home directories and accessing prohibited devices.

- No access to certain mounted data partitions

  Various data partitions may contain information that needs to be protected (e.g. HDMI, ECD keys). External applications should never have access to such partitions.

- No ability to mount file system

  This is necessary to prevent external applications to mount a data partition containing protected information. External applications should have access to externally pluggable media. Vendor software should mount external block devices and grant access to mounted data partitions to third-party applications.

- Limited network access

  Only trusted external subnets or hosts should be accessed by external applications. It includes only those network resources required for normal operation of Internet services.

- Limited access to platform API

  Currently the platform API is implemented as a single library providing access to all underlying resources. The API is divided into different groups per functionality. It is necessary to protect certain API subsets from external applications. If not, an external application can get a hold of the TV-tuner and, for example, distribute a TV stream across the network.

- Limited memory consumption

  External applications should be able to allocate and use only limited amounts of memory.

- Limited CPU consumption

  External applications should not consume more than given quantum of CPU time.

How to apply SMACK

Currently Linux kernel offers a number of different mechanisms to perform access control to different system resources and functions:

- Users and groups

  This is a discretion access control (DAC) mechanism used when different resources are controlled by their owners, identified by user and group ID. This implies having a power user (UID 0) that has a full set of privileges to access all resources.

- POSIX Capabilities

  A capability is a privilege to perform a certain action in the system. An example would be the capability to mount a file system or open a raw socket. A full set of capabilities forms the power of super-user but this power is split up in discrete privileges that can be individually granted to different processes not necessarily running as super-user (root). To support legacy UNIX super-user by default all the processes running as root are granted full set of capabilities.

- Linux security modules (SELinux and SMACK)

  System wide mandatory control access (MAC) to control access to different resources on a system level, rather than by different users.
- Control groups (C-groups)

Having different mechanisms, which are intended to accomplish the same goals, leads to undesired results when different mechanisms interfere and sometimes conflict with each other.

For example, when a process attempts to open a file, user/group permissions are checked first and if passed LSM is involved. However, if a user associated with the process does not have required permissions, access is immediately denied and LSM is not called. This breaks an idea of MAC where access to all resources is totally under control of the system.

Another example is interference between LSM and POSIX capabilities. There are capabilities that override LSM/SMACK policy, so if a process, e.g. running as root, has those capabilities then it can bypass SMACK rules.

Additionally, for different operations like opening a file or creating a device node, the order of calling different security mechanisms to perform access checks may be different. This makes it more complicated to configure and implement access control.

To avoid any possible conflicts between different security mechanisms it is preferable to let a single subsystem (SMACK for example) have total control of system resources. However, it will be discussed in the following chapter SMACK is not capable of addressing all the requirements. This calls for a hybrid approach, combining SMACK with other mechanisms like POSIX capabilities and C-groups. And impact of those mechanisms should be minimized.

Addressing the requirements

- No access to certain device nodes

  All the device nodes that needed to be protected can be protected using special SMACK labels and policy preventing external applications running with different labels to access them.

- No ability to create device node

  This is not directly supported by SMACK even though there is a corresponding LSM callback. However, there is a special POSIX capability (CAP_MKNOD) to control this.

- No access to certain mounted data partitions

  This can be accomplished with SMACK. Mount point and all files on a data partition should have special SMACK labels that are not accessible by external applications.

- No ability to mount file system

SMACK allows controlling this privilege basing on a label associated with a process trying to mount file system.

- Limited network access

  SMACK allows assigning labels to external host and subnets. In addition all packets coming from external network and not labeled with SMACK labels (using Netlabel to SMACK labels mapping) are associated with special “ambient” label. It allows controlling network traffic between third-party applications and external hosts.

- Limited access to platform API

  Platform API is based on DirectFB Fusion IPC library. This is a high-level IPC mechanism based on native Linux IPC, namely shared memory and special device driver helper. SMACK works with native Linux objects only, so theoretically it is possible to build in SMACK into Fusion.

  Exact implementation depends on how Fusion is utilized by the platform API (based on Fusion Dale). FusionDale uses a single Fusion “world” associated with a special Fusion kernel device node. Analysis of SPACE running on Philips digital TV shown that different platform API subsets are associated with separate memory mapped files shared across different process entered the Fusion “world”.

  This means it is possible to assign different SMACK labels to different platform API groups. However this approach requires modifications in platform API implementation as well as in Fusion (e.g. to handle a case when a certain shared file can not be mapped because of lack of sufficient permissions).

- Limited memory consumption

  In general, LSM and, thus, SMACK do not implement memory consumption control. To achieve this goal C-groups available in Linux kernel, can be utilized. An external process can be put to a special memory C-group that do not allow to consume more than a given memory quantum.

- Limited CPU consumption

  This type of control is not directly supported neither by SMACK nor other Linux mechanisms. To lower CPU consumption in case of intensive CPU load the application manager can adjust priorities of external applications processes.

Running third-party applications

All third-party applications in SPACE environment should be run following the rules below:

1. All third-party applications are running as super user (UID 0)
2. All third-party applications have all POSIX capabilities disabled

3. All third-party applications are assigned a special SMACK label
4. All third-party applications are put into a special memory C-group
5. All third-party applications run with lower priority than vendor applications

Although the first rule looks inappropriate, it intent is to have all applications running in the system having the same user id to prevent conflict between SMACK and permission control based on user groups. A full super-user power is taken away from third party processes by the second rule. All other processes running in the system are by default granted full set of privileges and, thus, SMACK does not break their operation.

The third rule enforces SMACK policy to prevent access for resources closed to user applications.

The forth and fifth rules allows controlling memory and CPU consumption, respectively.

Proposed solution

The solution proposed to implement necessary access control using SMACK includes the following components:

- SMACK rule set
- Necessary changes to SPACE:
  - Changes to system initialization scripts
  - External application installer
  - External application launcher
  - Platform API changes

SMACK rule set

This is the main rule set that defines access permissions to system resources for third-party applications. Each system resources group should be assigned to a special SMACK label and the rule set has to explicitly define relationships between third-party applications processes and resource groups.

In addition to the SMACK rule set, all additional SMACK configurations should be defined, such as network ambient label, mapping between external hosts/subnets and SMACK labels.

Changes to system initialization scripts

The system initialization scripts need to be changed to perform the labeling of system resources and

enforcing SMACK policy. This includes mounting root file systems with proper SMACK mount options, labeling device nodes, applying SMACK configuration and other necessary actions.

External application installer

There needs to be a way to distinguish third-party and vendor applications. One possible way is to provide a different set of keys to different development groups (vendor, partners or third-parties) and check against a key during installation process.

External applications may be installed from external pluggable media (e.g. USB stick or MMC/SD card) or from special network resource (application store).

Installation steps should include:

- Checking against a key
- Saving application files to a proper location
- Disable all POSIX capabilities associated with applications executable file

External application launcher

SPACE already implements the application manager application that is responsible for starting all other applications in the system. The application manager should be extended to perform the following steps during third-party application launching before executing new process:

- Disable all capabilities – set SECURE_NOROOT security bit for a newly created process so it does not receive all capabilities as a regular root process.
- Assign a SMACK label corresponding to third-party applications
- Put the new process to a proper memory C-group, if C-groups are used
- Set a lower priority to the new process

Platform API changes

Platform API changes are required to integrate SMACK support. These are required to protect different API subsets from being used by third-party applications. In current SPACE implementation, single Fusion 'world' is used for Platform API implementation, which means that all platform APIs are located in a single shared memory pool implemented as a single shared file. Such an implementation prevents SMACK to protect platform API groups. To make this possible, it is necessary to split platform API across different shared files and tag them with different SMACK labels. This work will not be covered in this paper and requires further investigations.

Applied SMACK

The implementation of concepts presented in a previous chapter includes a set of SMACK labels to tag different system resources and tasks, SMACK rule set to define relationships between third party applications and other objects in the system, and a third-party application loader. The main goal of the implementation is to develop a comprehensive list of subjects, objects and rules to meet the access control requirements. In addition to comprehensiveness there are few other characteristics that are just as important. It is also important that the number of rules and labels should be minimal to avoid extra complexity that adds more space for possible security holes. Also the proposed implementation should be a graphic example of SMACK application, easily understandable and maintainable.

Labeling system resources

The following sets of labels represent system resources relevant for third-party applications:

- ***third_party***

  Assigned to all third-party applications running in the system
- ***tv***

  Assigned to Digital TV specific data
- ***ext_media***

  Assigned to mount point and files located on external media (e.g. MMC/SD card)
- ***prot_device***

  Device node that should be protected from third-party applications
- ***open_device***

  Device node open for third-party applications
- ***trusted_net***

  Network resources open for third-party applications
- ***untrusted_net***

  Network resources closed for third-party applications

Smack rule set

The following smack rule set implements the access control requirements to third-party applications:

1. ***third_party*** ***open_device*** ***rwx***
2. ***third_party*** ***ext_media*** ***rwx***

3. ***third_party*** ***trusted_net*** ***w***
4. ***trusted_net*** ***third_party*** ***w***
5. ***_*** ***trusted_net*** ***w***
6. ***trusted_net*** ***_*** ***w***
7. ***untrusted_net*** ***_*** ***w***
8. ***_*** ***untrusted_net*** ***w***

The rule '1' grants third-party applications full access to open device nodes

The rule '2' grants third-party applications full access to external media devices

The rule '3' grants third-party applications write access to trusted networks

The rule '4' allows packets coming from trusted networks to be delivered to third party applications

The rules '5' through '8' are intended to allow other applications (native or partner) to access both trusted and untrusted networks. Since other applications running in the system are not explicitly assigned to any SMACK labels they get default '_' label, which is also default for all unlabeled network resources (default ambient label). Having these rules is necessary because of SMACK specifics in network packets handling where access check is performed against an application, not an object that application is trying to access.

Third-party application loader

The application loader is an instrument to run a third-party application in the system. It is started as a native application with a full set of privileges that allows it to take them out of loaded third-party program.

The loader perform the following steps to run a third-party application:

- Fork a new process
- Set a SMACK label for new process
- Disable all privileges for new process
- Execute third-party program

The application loader is called '***run_app***' and should be run as follows.

```
Usage: run_app [-h] [-n] -l label args
               -n      no-root flag, remove 0-UID priveleges
```

```
-l     label SMACK label, maximal 23 characters
-h     pint this message
args   program to run with parameter
```

The example below shows how to use the application loader to start a '***weather***' applet that connects to '***weather.net***' server (both applet and server mentioned in the example are fake). Note that the weather server should be added into list of trusted network:

```
run_app -n -l third_party wheather_applet -s weather.net
```

SMACK rule set verification

The verification process is intended to check that the proposed SMACK rule set implements the access control requirements for third-party applications running in the system. A special shell script has been implemented to simulate a third-party application trying to access different open and protected system resources checking whether access is granted or not.

Test environment

Hardware platform

The target hardware platform is a NXP TV543 digital TV reference board. However, due to a lack of support for this board in the latest Linux kernel and the fact that SMACK functional behavior does not depend on a hardware platform it has been decided to use a virtual environment to perform the SMACK rule set verification and analysis. The virtual platform used for verification is QEMU emulator version 0.9.1 for MIPS Malta Core LV platform. Three virtual machine instances have been used to simulate digital TV running third-party application, remote server located in trusted network and remote server located in closed network.

Linux kernel

- Linux kernel 2.6.30-rc7

  The latest kernel available at the time of SMACK testing contains SMACK with features not available in earlier kernel versions, e.g. labeling network resources.

Root file system

- glibc-2.9

- Busybox 1.7.2 with a SMACK patch from smack-util-1.0 package applied
- attr-2.4.43 package to manipulate extended file attributes
- libcap-2.16 library to manipulate POSIX capabilities of a process
- SMACK rule set stored in */etc/smack/load*
- SMACK test applications

Test applications

This chapters describes test applications and scripts that have been implemented to verify the SMACK rule set, including system preparation script, simple client-server application to simulate remote servers and test script simulating a third-party application.

Test preparation script

Test preparation script runs on QEMU instance representing digital TV platform and performs the following steps:

- Configure network interfaces
- Label network resources with SMACK labels
- Label protected device nodes with '***prot_dev***' label
  Devices labeled as protected are */dev/mem* and */dev/hdc*
- Label open device nodes with '***open_dev***' label
  Devices labeled as open are */dev/zero* and */dev/null*
- Create */home/tv* directory
  Represents digital TV protected data partition
- Create a file in */home/tv*
  Represents digital TV protected data
- Label */home/tv* and its content with '***tv***' label
- Create */home/third_party*
  Represents third-party applications home directory
- Create a file in */home/third_party*
  Represents third-party applications data
- Label */home/third_party* and its content with '***third_party***' label
- Mount external disk to */mnt/ext* and label it with '***ext_media***' label

Client-Server application

Simple client-server application is intended to test access control to different networks. Server side opens a TCP port and listen for connection. Once connection is established and message is received it sends a response and exit. Client side tries to set up a connection with server, sends a message, and waits for response then exits.

SMACK test script

The test script runs as a third-party application and is intended to verify whether the system meets the access control requirements for third-party applications. The test attempts to perform different operations and access various system resources. On each step a result of performed operation is compared with expected value and if they do not match then failure is reported. The sequence of steps executed by the script is the following:

- Access protected data partition (directory).

  All operations should be prohibited:
  - Read content of protected directory
  - Change current directory to protected directory
  - Read a file located in protected directory
  - Create a file in protected directory

- Access home data partition (directory)

  All operations should be allowed:
  - Read content of home directory
  - Change current directory to home directory
  - Read a file located in home directory
  - Create a file in home directory

- Access external medial mounted to a local directory

  All operations should be allowed:
  - Read external media content
  - Read a file from external media
  - Create a file on external media

- Access protected device nodes

  All operations should be prohibited:
  - Read data from protected device node
  - Write data to protected device node

- Access open device nodes

  All operations should be allowed:
  - Read data from open device node
  - Write data to pen device node

- Create a device node in home directory

  The operation should be prohibited

- Mount external device

  The operation should be prohibited

- Connect to a trusted server

  The operation should be allowed

- Connect to an untrusted server

  The operation should be prohibited

Test procedure

The entire verification process includes the following stages:

1. System startup
   - QEMU instance representing digital TV platform is launched
   - SMACK rule set is loaded during system startup with init script

2. Test preparation
   - The test preparation script is executed
   - QUEMU instances representing trusted and untrusted servers are launched

- Server application is started on the both QEMU nodes

3. Test execution
   - The test script is started with the third-party application loader applications

4. Results verification
   - Check if the test script reported any error message
   - If no errors have been reported then the test is passed
   - If at least one error has been reported then the test failed

SMACK memory foot-print analysis

This chapter discusses SMACK contribution into overall system memory consumption, the subject that is very important for embedded systems having in order of magnitude less memory than desktops and servers.

The memory consumption analysis performed covers both static and dynamic memory consumed by SMACK kernel module and included:

- SMACK source code examination

  This step is intended to identify places where SMACK perform dynamic memory allocation and when.

- SMACK module object file analysis

  This step is intended to determine module size including code, static initialized and uninitialized data, and memory that will be freed after the module is loaded and initialized.

- Run time memory allocation tracing

  This step is intended to assess SMACK impact to dynamic memory allocation and, thus, other applications running in the system.

The entire memory allocation analysis has been performed within the same test environment used for the SMACK rule set verification. Using a virtual environment does not impact memory allocation performed by Linux kernel and user applications and provides mechanisms helpful for dynamic memory allocation analysis. For instance, GDB stub implemented in QEMU virtual environment.

The tolls and mechanisms used for memory analysis include:

- GNU Binutils for MIPS
- Built-in Linux kernel capabilities:
  - /proc/meminfo
  - SLAB allocator tracer (*kmemtrace*)
- GNU Debugger (gdb) for MIPS

GNU Binutils tools have been used to perform static memory analysis and identify all the places where Linux SLAB allocator is called to provide dynamic memory.

Memory information reported via '*procfs*' file system has shown how much system memory is allocated in total to compare kernels with and without SMACK module enabled.

SLAB allocator tracer is a relatively new kernel mechanism available in the latest mainline kernel. It traces calls to SLAB allocator from different kernel modules and reports this information to user space applications. It helped to trace SMACK module memory allocation dynamics. However, it is not possible to trace memory allocations/frees from a point when the system is started because the *kmemtrace* module is loaded during last stage of system initialization where a lot of memory allocations were already performed by SMACK module.

To trace early memory allocations a number of counters have been introduced to the SMACK module updated by routines allocating dynamic memory. The GNU debugger attached to QEMU virtual machine has been used to track those counters in run time.

Static memory foot-print

SMACK module built in part of Linux 2.6.30-rc7 kernel has a size of 24 KBytes including 20772 Bytes of code and 1304 Bytes of static data.

Dynamic memory foot-print

SMACK source code analysis has shown that there are a few places where SLAB allocator is called. Basically, for every object created in the system (e.g. file or socket) SMACK creates a special structure associated with that object. So, the actual size of dynamic memory allocated by SMACK depends on the number of objects in the system, which will vary depending on system load. Amount of memory allocated for a single object is relatively small:

- 44 bytes per SMACK label

- 20 bytes per SMACK rule
- 32 bytes per network label
- 28 bytes per file system inode object presented in memory
- 24 bytes per mounted file system
- 32 bytes per socket

However, due to SLAB allocator overhead, which allocates memory from preallocated pool of fixed size chunks, it allocates 128 bytes per an object.

The difference in dynamic memory consumption between kernels with and without smack enabled is about 644 Kbytes after system is initialized. It includes 587 KBytes allocated for file system inode, super block, and socket objects. Running the SMACK test script does not affect dynamic memory consumption.

SMACK performance analysis

In this chapter it will be discussed how performing run-time access control with SMACK LSM module impacts system performance. SMACK implements mandatory access control on a system level which means that it is involved in most of operations performed by user applications running in the system. It means that it impacts performance of every single operation, which, in turn, impacts overall system performance.

The methodology chosen to assess performance impact is to run the same set of benchmarking tools in the same hardware environment on two kernels, one with SMACK module enabled and another without it. Then results are analyzed and compared. The most interesting areas to watch SMACK overhead are file system operations and networking.

Since system performance depends on hardware platform (amount of CPU speed, I/O device throughput, physical memory amount) it is necessary to run benchmarking tests on a real hardware platform, which is NXP TV543 reference board:

- PNX8543 SoC based on MIPS 4Ke core running at 300Mhz speed
- 128MB RAM
- USB 2.0 storage drive

The kernel used for benchmarking is a Linux 2.6.27.9 port to the TV543 board. The user space part of software stack (tools, applications and libraries) remained the same as for the SMACK rule set verification.

Having an older kernel version limited performance testing to file system operations because SMACK version in Linux 2.6.27 kernel does not support assigning labels to network resources, the feature actively used in the proposed implementation. However, since access control for different operation (e.g. opening a file or establishing network connection) is performed by the same routines assessing impact of file system operation can give us an idea about possible impact to network. Also, networking is not that important for digital TV platform as file system performance as long as SMACK does not drastically reduce network throughput.

File system performance analysis

Methodology

In general, Linux performs access control checks on most of file operations including opening, creating or deleting a file, reading or writing to file, etc. It does it through mechanisms provided by the LSM module enabled in the system, e.g. SMACK or SELinux. However, as soon as a user process opens a file, SMACK no longer verifies access permissions on reading or writing operations and always grants access to the file. It means, that the overhead introduced by SMACK on read/write operations should be minimal and may vary depending on the number of blocks written to a file and the size of a single block. Most of overhead introduced by SMACK should be for operations associated with file manipulation operations, such as creating or deleting.

To verify the expected results discussed in the previous paragraph the following set of file system tests were selected:

- Running '*bonnie*++' test to create a large number of files of different size
  - The test assess SMACK impact on file manipulation operations
  - The test reports the number of files created or deleted per second
  - File sizes used are 0 Bytes, 1 Byte and 10 KBytes
  - The number of files is 10240
  - The test has been run in two modes, with and without write buffering enabled. Disabled write buffering means performing file system buffer flushing after every single file operation

- Running '*bonnie*++' test to write large amounts of data
  - The test assess SMACK impact on read/write operations
  - The test reports a number of bytes read or written per second

- Copying a file located in RAM based file system (*tmpfs*)
  - The test assess SMACK impact on read/write operations
  - The test reports a number of bytes read or written per second
  - Tools used to copy a file are '*dd*' and '*cp*'
  - Block sizes used by '*dd*' tool are 1 KByte, 8 KBytes, 64 KBytes and 512 KBytes
  - RAM based file system allows to minimize I/O waiting time
  - Size of copied file is 10 Mbytes

In addition to the reported numbers, CPU utilization has been tracked for all tests to assess SMACK impact to CPU consumption and I/O wait time.

All of the tests have been run on both kernels with and without SMACK enabled. Then average results have been calculated and the difference is calculated in percentage relative to the numbers obtained from non-SMACK tests.

Results

File creation

The following performance degradation has been observed in SMACK-enabled kernel using *bonnie++* file creation tests in both random and selection order (percentage rounded up):

- Creating files of zero byte length: 5%
- Creating file of 1 byte length: 6%
- Creating files of 10 KBytes length: 12%

The first number shows pure SMACK overhead on file creation operation, because no writes are performed to a file in this case. Running the same tests with disabled write buffers showed performance degradation in range from 2% to 4% depending on file size. In this case, SMACK overhead is compensated by I/O device access time.

File deletion

Bonnie++ test application performs file deletion tests in two ways, in sequential order and random order, showing drastically different results in both cases. In the case of random order, the performance degradation varies in 7% - 10% range, then for sequential order it increases up to 30% in case of zero length file. Such a difference can be explained by the fact that file system driver reads a block of data

from I/O device to memory, so information about the next file in sequence is already precached in memory, so I/O device access is not performed on each operation. That makes SMACK overhead relatively long comparing to delete operation resulting in high performance degradation.

Flushing file system buffers after each operation reduces SMACK overhead to 1%-3% range in both, sequential and random, cases. Similarly to file creation test, SMACK overhead is compensated by I/O device access overhead.

Read operation

Neither '*bonnie++'* test nor '*cp*' tests have shown any performance degradation of file read operation.

Write operation

The results reported by *bonnie*++, *dd* and *cp* tools are in 0% - 5% range of performance degradation depending on block size. The worst result (5%) is reported by 'bonnie++' tool for byte-to-byte writing of 10 MBytes file. SMACK adds a constant absolute overhead to a write operation always granting access to a file. However, total number of write operations depends on block size resulting in different relative overhead that can be totally compensated by I/O device wait time.

Conclusion

The first part of this paper formulates access control requirements for third-party applications based on analysis of existent Digital TV platform offered by Phillips.

Then a solution that addresses the requirements is proposed. However, two problems have been revealed during research performed to develop the solution. First, using SMACK is not enough to meet all the access-control requirements. And secondly, using high-level IPC mechanisms in the platform may complicate the task of applying SMACK depending on the way high-level IPC maps to native Linux IPC mechanisms that are controlled by LSM.

The proposed implementation is presented along with verification process and its results. It proves the feasibility of SMACK application.

The last part of the document provides results of SMACK memory consumption and performance analysis showing that SMACK is suitable for embedded systems and, in particular, to digital TV platforms. Even though some numbers reported by performance tests can be relatively high, they have been derived under unusual system load different from normal TV operations.