Kaspersky Security System for Linux

Introduction

Linux-based operating systems are widely used in embedded systems. You can find Embedded Linux in consumer electronics (e.g. set-top boxes, smart TVs, personal video recorders, in-vehicle infotainment, and networking equipment such as routers, switches, wireless access points, and wireless routers), machine control, industrial automation, navigation equipment, spacecraft flight software, medical equipment or even on mobile devices such as smartphones or tablets. There are dozens of vendors who develop special versions of Linux designed for embedded devices.

Existing security extensions like SELinux or AppArmor are normally designed for very general use and not always suited for embedded solutions due to being:

- difficult to configure
- too general for an embedded system
- insufficiently flexible to model system specifics

Normally, an embedded system has functionality known in advance and a relatively short list of security properties, simply formulated in domain-specific terms. An example of such a property is "any communication of application X with sensor Y must be prohibited until service Z is fully initialized".

However, security extensions like SELinux are built around UNIX concepts, so it is up to the developer how to reshape security properties from their problem domain into the "standard" notions and means.

In the case of an embedded system, this is especially disappointing, because it leads to two types of problems:

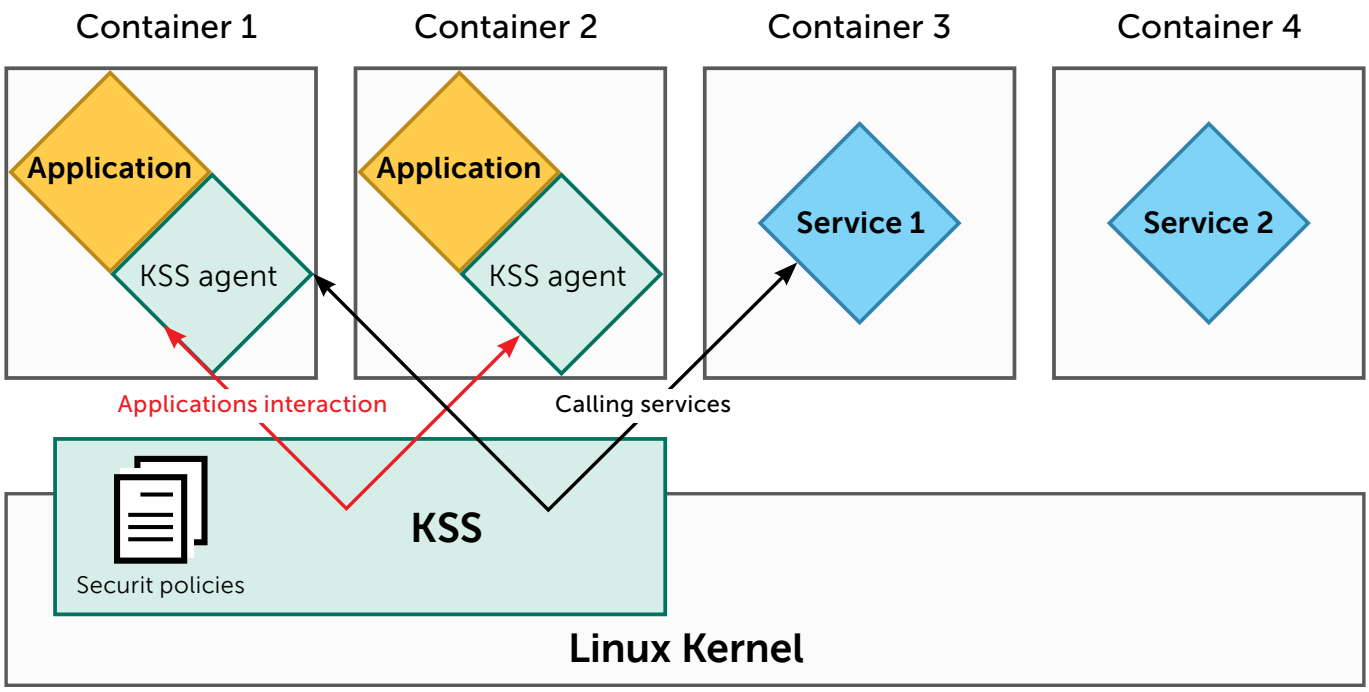
1. Reformulating security objectives is often a non-trivial task that results in unnecessarily complex configurations.
2. Sometimes developers give up and build security policies directly into their business applications.

The former makes it difficult to review and maintain security configurations. Having said that, the latter is much worse – it violates the principle of separating business functionality from security mechanisms. As a result, when elaborating functionality, developers may neglect or forget security- related aspects of the code. Inconsistencies in security and functionality may allow an attacker to take control of the system by exploiting a minor vulnerability in the code. At the same time, a dedicated component for security monitoring that requires specific attention during every change of functionality makes the whole system more dependable.

Other concerns regarding the use of Linux as a base for embedded solutions:

- Large amount of legacy code does not help ensure software dependability.
- Development process usually involves several parties that may change from version to version, raising trust issues with the components provided by those parties.
- Threats in the system boot procedure may undermine any protection implemented by an OS.
- Software updates in a secure manner without any impact on system continuity.
- Irrelevance of conventional security policies to the needs of embedded solutions and a lack of mechanisms allowing customization of security polices according to a particular process.
- Ad-hoc security-related checks implemented incoherently by different components may cause unexpected failures.
- Excessive privileges for some processes as a result of neglect or negligent development (e.g. assigning root access to simplify further maintenance).

Kaspersky Security System for Linux (KSS/Linux) helps address the concerns listed above. It is intended to foster the fast and dependable development of secure embedded solutions.



What Does KSS/Linux Do?

- Provides the means to specify a security policy relevant to an application area.
- Encloses applications into Linux containers.
- Provides communication channels between those containers.
- Manages containers, secures communication channels and enforces configured security policies.
- Provides set of ready components, such as secure service to remotely manage the system, audit/logging, secure storage.
- Can be extended with a custom security policies component.
- Provides the means to securely update core KSS/Linux components: crypto libraries, certificates, keys and other security-related data. (It also can be integrated with a full device firmware update.)

Kaspersky Security System can be integrated with Linux-based solutions in one of two ways: shallow or deeper integration.

Shallow Integration

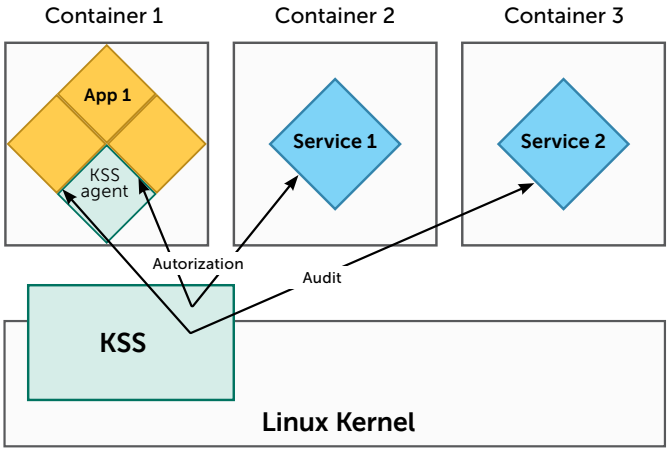
The simplest form of integration requires minimal changes in the existing solution. Put the application in a container and make use of components provided by KSS/Linux:

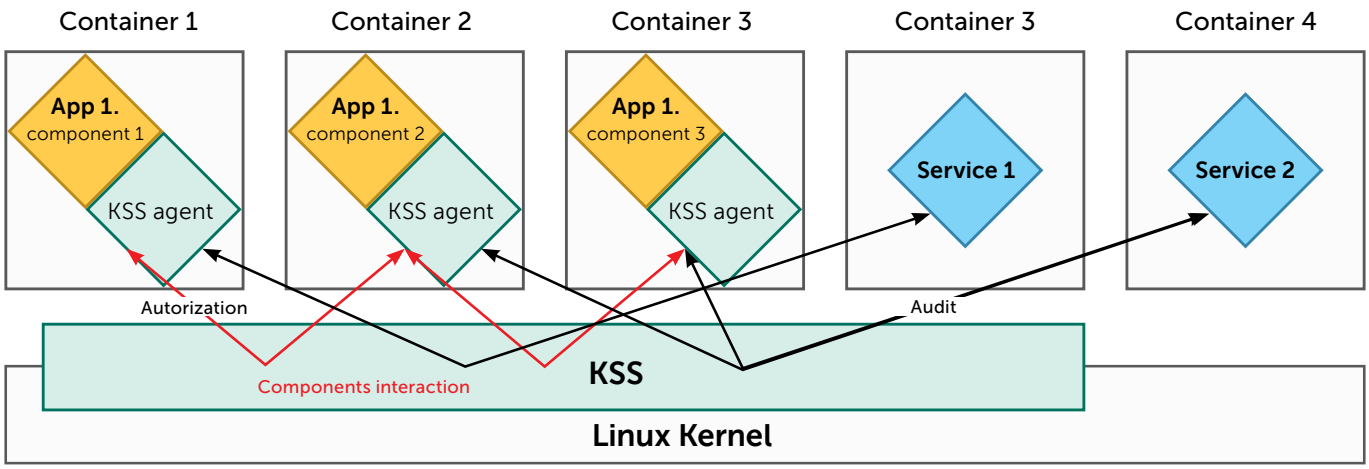
The simplest form of integration is to put existing applications into Linux containers, managed by KSS and deleg.

The most security-critical functions are delegated to the dedicated components, such as:

- secure remote access
- secure audit/logging capabilities
- secure storage for sensitive data (application configuration, keys, certificates, policy settings, etc.)

All these components can be restricted by the application-specific security policy, configured and enforced with KSS/Linux.





Deeper Integration

Deeper integration of Kaspersky Security System with Linux- based solutions implies reworking the architecture of existing applications in order to make these applications inherently secure (with the help of KSS).

For example, the web-based service used for safe and secure remote access to an industrial application can be easily decomposed into several components: input handler, request safety checker, and request processing engine.

As the most exposed and usually most vulnerable component, the input handler runs with minimal privileges. All requests from the input handler to the processing engine are sanitized according to a strict policy. The request safety checker can either act as a mediator between the input handler and request processing engine, or be called as a service by the processing engine. An access authenticator and audit service also act as services with explicitly defined interfaces.

Decomposition may vary from that described above, while retaining the main idea – making security-critical components as small as possible and controlling all component interactions with KSS/Linux.

This gives the following advantages in addition to the features provided by shallow integration:

- Fine-grained monitoring and control of application behaviour due to access to communications between its components
- De-privileging of the most vulnerable parts of the application
- Ability to create specific services such as a secure remote update service using components provided by KSS

Use Cases

Most embedded systems have similar requirements regarding their maintenance and support. For instance, they may require regular software updates, remote configuration, or the ability to install software supplied by third parties. This similarity often leads to the creation of solutions that already exist, reinventing the wheel, using any available means, from low-level libraries to entire solutions supplied by untrusted developers.

This approach usually improves speed to market – and negatively impacts on both the quality and cyber security of the final product. Kaspersky Lab offers a method, the means, and a set of practices intended for the reuse and composition of existing software components, resulting in a secure embedded solution that fulfils various needs.

Kaspersky Security System integrated with Linux is used as a base for:

- Industrial embedded solutions (PLCs, RTUs, HMIs)
- Internet of things gadgets and appliances
- Telecommunication equipment

Secure Remote Device Updates

Some embedded systems may require particular attention to aspects that rarely arise in pure IT applications, like continuous execution or aligning procedures with safety restrictions. For the procedure of remotely updating a device or application software, these aspects should be considered together with the integrity and authenticity requirements of the remotely provided updates. A possible design solution is the implementation of a designated service for remote update that encapsulates all the necessary security mechanisms while also complying with external regulations. A solution of that kind requires flexible and configurable security policies and a mechanism for proper isolation of the service – and both aspects are provided by KSS/Linux.

Secure Remote Device Reconfiguration

The need for remote maintenance and reconfiguration of the embedded solution may force the developer to give extensive privileges to processes that are intended for the appropriate changing of system settings. In the worst case scenario, the reconfiguration functionality is built into the application itself. This sort of design may result in the total compromise of the application and the system itself due to either misuse of reconfiguration functions or exploitation of vulnerabilities in the code that is running with excessive privileges.

There is a design solution for this problem that does not require significant efforts. It is better to implement remote device reconfiguration using a special isolated agent in the system environment. Any adjustment of the rights for all processes in this environment should be in line with the principle of least privilege. Enforcement of an explicitly defined reconfiguration policy should be provided by mechanisms that are independent of the configured process itself, and this policy should be based on the default-deny principle.

KSS/Linux naturally implements this design solution.

Separation Of Duties

Sometimes it is essential that applications running in different predefined modes do not interfere with each other. For example, diagnostic procedures with a physically connected tester must not interfere with remote requests to the equipment under diagnosis. Diagnostic information must not be shared with the remotely connected party.

This non-interference can be achieved by implementing security policies. The enforcement engine may block execution of the application in a particular mode if it does not satisfy the conditions set by the policy. Usually, this policy is specific to a system. The support for flexible security policies and configurations provided by KSS/Linux is very useful for this purpose.

Sandboxing Untrusted Components

Vendors of embedded systems are continuously improving the quality of code and strengthening the design of newly introduced software solutions, while at the same time legacy applications are still widely used in the areas of industrial automation, transportation, energy supply, and other critical domains. Important components that cannot be replaced in the near future and that may threaten the system due to their insecurity, must be isolated and supplied with external hardening measures.

For instance, these measures may include authentication of users and requests, encryption of external connections, filtering of requests, checking of digital signatures for downloaded binaries, and other mechanisms not initially implemented.

Kaspersky Security System facilitates the proper integration of legacy components and introduced security services, working as the reference monitor for their interconnection.

Necessity Of In-Depth System Hardening

The use cases listed above are not independent. What is common to all of them is the idea of proper component isolation and control of their communications with a designated mechanism. They may be successfully combined to fit objectives that are more complex. KSS/Linux allows the control of both external and internal communications, which is very important when components of different sensitivity and trust levels intercommunicate.

Technical Requirements

Linux kernel version 2.6.30 or higher
(3.8 or higher recommended)

Architectures: Intel x86, ARM, PowerPC

www.kaspersky.com

© 2017 AO Kaspersky Lab. All rights reserved. Registered trademarks and service marks are the property of their respective owners.

Find out more at **os.kaspersky.com**
All about Internet security: **www.securelist.com**