kaspersky BRING ON THE FUTURE

KasperskyOS Technical Data

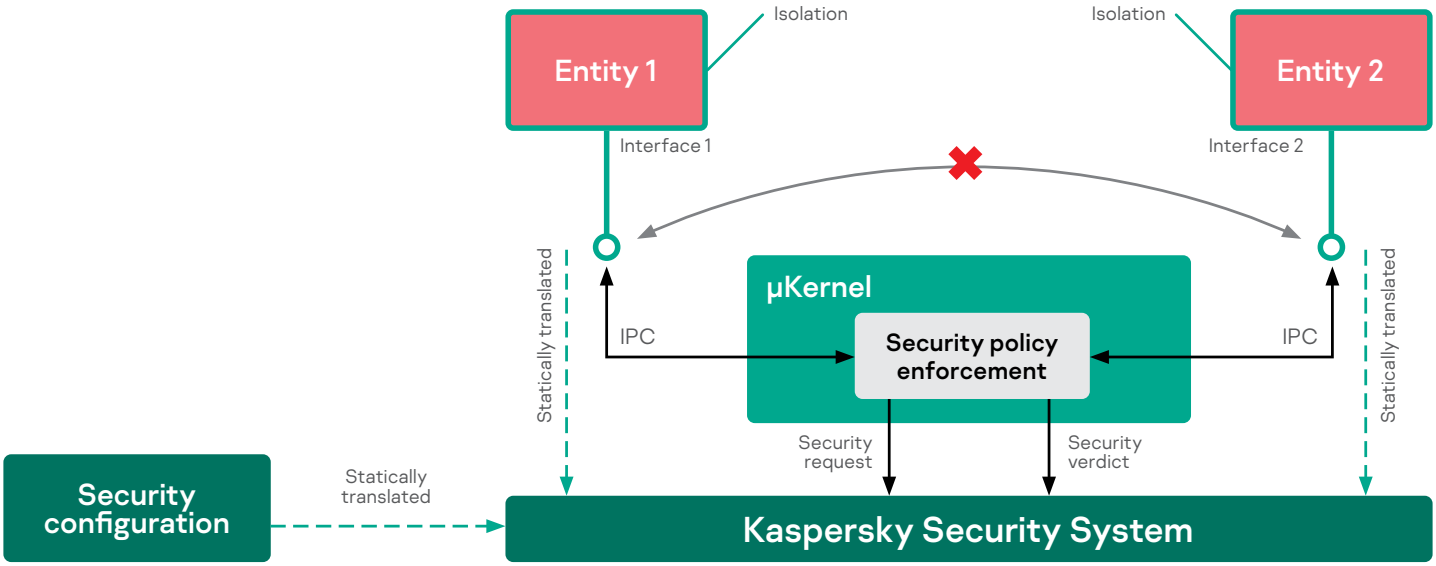
KasperskyOS®

KasperskyOS is a secure operating system for embedded connected devices with specific cybersecurity requirements.

KasperskyOS creates an environment to securely run untrusted and potentially vulnerable code.

Main features of KasperskyOS

- **Microkernel-based secure OS**. Based on an in-house microkernel, not a modification or improvement over any existing OS.
- **Secure by design**. Developed on MILS principles and integrating a flexible access control system (KSS).
- **API for user-space drivers**. SDK can be used with customer drivers and applications.
- **POSIX support**. Support for approximately 98% of POSIX API.
- **Portable**. KasperskyOS runs on Intel x86/x86_64 and ARM platforms. Other hardware platforms with MMU can be supported on request.
- **Hardware virtualization support**. KasperskyOS can be used as the basis for a secure hypervisor with support for Intel VT-x and VT-d.
- **Securing inter-process communications**. Sophisticated approach facilitates control of inter-process communications in accordance with specified security policies.
- **Low attack surface**. Separation of security domains and powerful control over inter-domain communications enables secure execution of vulnerable and/or untrusted applications.



The combination of various security approaches forms the basis of KasperskyOS.

Security principles

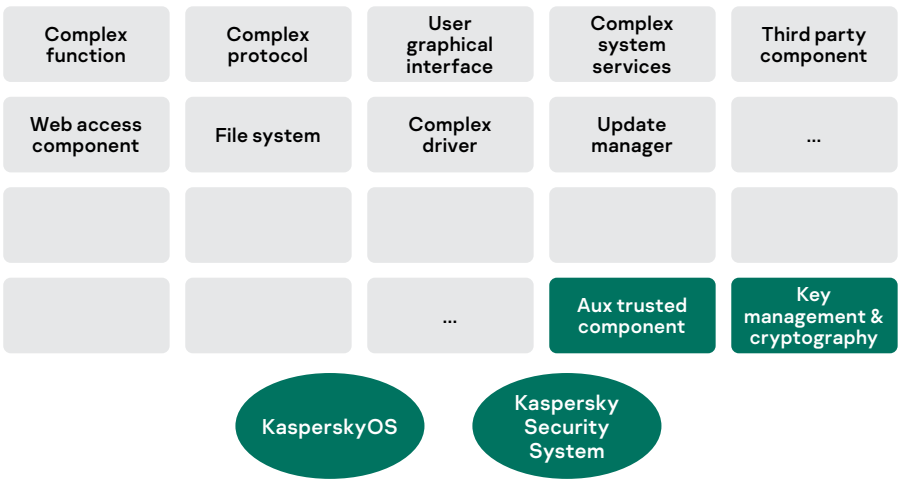
Most operating systems consider security a matter of separating and controlling access to system resources. Unlike those operating systems, KasperskyOS extends this scope with capabilities to specify and enforce solution-specific security properties.

- **µKernel**. Minimal amount of code lines necessary to make kernel mechanisms work, providing more control over the OS code quality.
- **Strong isolation**. The system guarantees isolation of security domains and separation of security features from functional components.
- **Unified inter-process communication (IPC) mechanism**. The microkernel provides a single IPC mechanism.
- **Explicitly defined typed interfaces**. Every service must statically declare all provided interfaces. KSS verifies the correctness of all IPC messages according to interface declaration.
- **Static security configuration**. All processes and their permitted types of communication are preconfigured and checked before functioning.
- **Complete mediation**. The microkernel intercepts all inter-process communications and checks with Kaspersky Security System (KSS). KSS calculates access decisions based on the security configuration.
- **Default deny**. Any action that is not preconfigured in the security policies is denied by default.
- **User-space device drivers**. Drivers are isolated from each other, the microkernel and applications; because drivers run as unprivileged code, an error in one driver will not affect the rest of the system. Possible to restrict driver access to physical devices.

It's true that we don't live in a world of bug-free software, but we do want it to be secure. That's why one of the challenges for KasperskyOS is to keep the whole system secure using a bare minimum of trusted components.

Trusted components in an untrusted environment

One of the main aims of KasperskyOS is to bring security to a complex system using the minimum amount of trusted components. With KasperskyOS, a complex system can be divided into a set of isolated entities or components. Secure critical functions can be placed in separate simple components with a low attack surface that are easy to verify. Only a bare minimum set of functions is considered to be trusted, while other components are not trusted and may contain problems and vulnerabilities of some kind. By means of KasperskyOS and KSS, security properties are defined and enforced for the whole system. Even if a vulnerability is exploited in one of the untrusted components, it doesn't influence the whole solution and doesn't damage critical functions.



With KasperskyOS you can create a flexible and secure solution, but it doesn't mean that every KasperskyOS-based solution is secure.

Methodology

As part of its holistic approach, Kaspersky provides a carefully developed methodology that describes how to create and implement a secure-by-design solution. According to the methodology, it is first of all necessary to understand the functionality definition of the desired solution. Threat analysis then needs to be performed to build a threat model reflecting all the weak points of the solution and providing recommendations and security requirements. After that comes functional design, security separation, TCB definition, design of communication interfaces and security configuration design (specification of security properties). After the design is verified and validated, the security solution can be implemented in accordance with security guidelines. And finally, after implementation, the solution should be thoroughly verified and validated in a process that includes various testing techniques (unit, system, functional, fuzz, penetration, etc.). Following this methodology is essential for the creation of a secure solution.

KasperskyOS-based technologies

There is a list of additional security solutions that can be used together with KasperskyOS:

- **Secure Boot** – hardware-assisted procedure to guarantee the integrity and authenticity of the KasperskyOS boot image.
- **Secure Update** – helps to remotely perform a reliable software update.
- **Secure Audit** – a solution called to recognize, record and store audit logs and provide guarantees that log entries cannot be altered. Adheres to ISO/IEC 15408-2 requirements.
- **Secure Storage** – a key/value registry tightly integrated with KSS to control access to sensitive data.
- **Trusted Channel** – TLS-based framework to establish an encrypted data channel to authenticated and authorized remote parties. Trusted channel framework is configurable to support various backends and communication schemes.
- **Secure Hypervisor (KSH)** – a virtualization solution on top of KasperskyOS to run other operating systems. KSH can run multiple untrusted guest operating environments on a single HW platform, averting any unwanted influence they may have on each other or the host operating system.

Supported languages for native KasperskyOS application development are C and C++. There is also a set of scripting languages: Java, Lua, Qt Script, QML, JavaScript.

Development of KasperskyOS

KasperskyOS, Kaspersky Security Hypervisor, Kaspersky Security System and all the core user-space drivers, services and libraries are developed in C. The toolchain is based on the GNU Compiler Collection.

A set of custom declarative languages (Interface Definition Language, Security Policy Specification Language, etc.) is used to describe the properties of all the security domains as well as the security policies for the deployed system. A compiler suite is provided to translate from these languages into C.

KasperskyOS drivers are user-mode entities that should comply with the KasperskyOS driver model.

System requirements

CPU requirements:
- **Memory Management Unit;**
- **IOMMU (SDMA for ARM) is strongly recommended for reliable isolation of hardware resources.**

Supported architectures:
x86, x86_64, ARMv5, ARMv7, ARMv8 and MIPS32.

Tested hardware platforms:
Intel Generic and Atom CPUs,
NXP i.MX6 (Solo, Duo and Quad),
NXP i.MX27, TI Sitara AM335x,
TI Sitara AM43xx, HiSilicon Kirin620,
MIPS24k.

Minimum RAM size is solution specific.
Recommended RAM size is 128MB.

KasperskyOS supply

KasperskyOS is supplied in SDK form, specially developed and configured for the needs of each customer.

KasperskyOS SDK is a framework that contains all the necessary components required by customers to build their own solution. Kaspersky usually prepares a specific SDK for each customer to meet their requirements, based on HW platform or a set of components.

KasperskyOS SDK includes:

- KasperskyOS µKernel in binary form;
- KSS runtime in binary form;
- Drivers, system libraries and other components from Kaspersky in binary form;
- Third-party open-source software components;
- Examples for developers along with the source code;
- Set of compilers and other tools that include a GCC-based compiler suite and the KasperskyOS compiler suite.

Third-party code license: µKernel, KSS and the core set of SDK components use open-source components with permissive licenses.

KasperskyOS compatibility

KasperskyOS applications can use ISO/IEC 9899:1999 and/or POSIX compatibility layer.

PSE51 and PSE52 POSIX 1003.13 profiles are fully supported. POSIX 1003.1 standard is also partially supported with the most notable limitations being the absence of process control primitives (such as fork() and exec()).

Kaspersky Secure Hypervisor supports unmodified guest Linux and Windows operating systems on top of KasperskyOS on platforms supporting hardware virtualization.

Applications running inside guest operating systems may have access to native KasperskyOS message passing primitives, which allows the business logic between secure native KasperskyOS applications and rich guest OS applications to be decoupled.

www.kaspersky.com

KasperskyOS®-based products: os.kaspersky.com/products/
Kaspersky, Moscow, Russia: www.kaspersky.com
Cyber Threats News: www.securelist.com

#kasperskyos
#bringonthefuture