Design and Implementation of a Secure Linux Device Encryption Architecture

Ahmad-Reza Sadeghi (sadeghi@crypto.rub.de)
Michael Scheibel (m.scheibel@sirrix.com) *
Christian Stüble (stueble@acm.org)
Marcel Winandy (winandy@crypto.rub.de)

 Horst Görtz Institute for IT-Security
Ruhr-University Bochum
Universitätsstr. 150, 44780 Bochum, Germany

* **Sirrix Security Technologies AG**
Zentrum für IT-Sicherheit
Lise-Meitner-Allee 4, 44801 Bochum, Germany

The EMSCB Security Architecture is used as a trustworthy basis for the implementation of secure distributed applications. In this paper we introduce the prototype for a device encryption system based on the EMSCB security kernel. The goal is to provide a strongly isolated hard-disk encryption for Linux, where the secret key information and all related security-critical operations are not under the control of Linux, but under control of an EMSCB application protected and isolated from Linux. We describe the architecture and the prototype implementation of the device encryption system.

Introduction

Today, increasingly more sensitive data is stored on private and business devices such as PC's, Laptops and PDAs. The security critical data include business plans, authorization secrets, and email correspondence. In case the device is stolen or lost this data may be compromised.

An approved security mechanism to mitigate this risk is to encrypt the data. There exist several software-based encryption systems. Some of them are shipped together with the operating system. One example is Linux and its `dm_crypt`, which allows different encryption algorithms to plug in and use them for encrypting file systems.

Unfortunately, most software-based hard-disk encryption products suffer from insecure storage and usage capabilities for security-critical cryptographic keys and operations. The underlying operating systems (OS) that control all data storage mechanisms, i.e. hard-disk, memory, USB, I/O etc., cannot prevent other (potentially malicious) applications from gaining access to the critical key data. This can be seen by the huge number of exploits and continuous security

updates. The reasons are due to various conceptual weaknesses of common computing platforms, in particular due to the monolithic OS kernel architecture and thus increased complexity. This concerns Windows-based operating systems as well as Linux-based ones. A large part of the operating system and supporting processes are executed in a privileged mode, the so called kernel mode, which allows them to directly access the hardware and all other software processes. User applications are usually executed in a non-privileged mode, the so called user mode. Thus, the risk of security weaknesses is higher because of the huge amount of code executed in privileged mode. If such a process can be exploited it is possible to gain access to all kernel data, including the encryption keys used for the hard-disk encryption. An attacker may read out the encryption key from kernel memory or simply deactivate the encryption system by exploiting a common security hole. Runtime protections such as access control and user authentication may be easily circumvented by booting an alternate operating system. Furthermore, an untrusted system administrator usually has full access to all system resources including the cryptographic keys of the users. Countermeasures such as mandatory role-based access control (e.g. SELinux) protect this information from a "root spy" but are much too complicated to maintain and evaluate [1].

We propose a solution to this problem by providing a security architecture that allows secure, reliable and user-friendly device encryption. The security architecture strongly isolates the secret key information and all related security-critical operations from the Linux operating system. This is similar to a hardware based solution but far more cost-effective. Moreover, the architecture is capable of using Trusted Computing (TC) functionalities (based on [2]) to protect the cryptographic keys and to assure software integrity during the booting process of the system.

Threats and Security Requirements

We identified the following threats a device encryption system must deal with: An adversary may try to eavesdrop the cryptographic key used for encryption/decryption. He may try to violate security requirements by maliciously manipulating the system. Moreover, he may try to violate the integrity of the cryptographic keys.

An adversary may try to eavesdrop the user authentication information. He may try to deceive users by a platform providing a faked user interface. Again, he may try to maliciously manipulate the system to gain the authentication information.

Finally, an adversary may gain access to the encrypted data, e.g., on a stolen device, and try to mount offline attacks, i.e., trying to decrypt the data by exploiting weaknesses of the cryptographic algorithm used for encryption.

The main objective of a device encryption system is to protect the confidentiality of data resulting in the following requirements:

- **User authentication:** Only authorized users should be able to access

sensitive data, i.e., authorization is required at start-up time of the system. The authorization should withstand long-term attacks in case of system theft or loss. Moreover, an authorized user should be able to change her authorization data.

- **System integrity:** The encryption system must not be deactivated or tampered with, i.e., the integrity of the encryption system should be protected at runtime and checked at boot time. This requirement pertains to the user interface as well, i.e., the user must be able to trust the input path to the encryption system when entering his authorization data.

- **Confidentiality of encryption keys:** The encryption key should be protected from being read out by unauthorized users or programs.

- **Strength of cryptographic algorithms:** The key generation algorithm should comply with current requirements. The encryption algorithm should be an approved standard. Its operation mode and the key length should provide reasonable security. Besides, the algorithm should be able to be securely updated to meet future requirements.

Related Work

The are a number of software device encryption systems available today. However, most of them either do not offer essential security properties such as isolation (of the encryption keys and operations from the operating system), or they are not open source and not being subject of public analysis.

Commercial Products

Examples of commercial software device encryption systems available at the market are [3],[4],[5],[6]. These products offer variety of features[1]. In this context some products already use the interfaces to a Trusted Platform Module (TPM) to bind encryption keys to hardware and/or software components and for secure random number generation (partially).

A further product is Microsoft's “Secure Startup - Full Volume Encryption” which will be integrated into the upcoming client version release of Microsoft's Windows Operating System (“Windows Vista”) [4]. This encryption feature encrypts the entire Windows volume and uses a Trusted Platform Module (TPM) 1.2 to bind the encryption key to the boot stack, thus ensuring that system files have not been tampered with while the system was offline. However, it does not use TPM authentication mechanisms but relies on conventional OS authentication after the system integrity has been verified.

1 such as AES encryption, centralized user administration and policy enforcement, key recovery mechanisms, pre-boot authentication, multi-user support for sharing resources, etc.

Enforcer Project

The Enforcer [7],[1],[8] is a Linux Security Module (LSM) that binds the cryptographic key for an encrypted file system to long-lived system components, such as the Linux kernel, the boot stack, the Enforcer LSM, and the public key of a so-called “security admin”. The security admin issues and digitally signs a list of file hashes. This security configuration is used by the Enforcer LSM to check the integrity of the applications before execution.

The Enforcer even provides a mechanism to guarantee the freshness of a security configuration. To verify the integrity of the long-lived components the Enforcer enhances the LILO boot loader with TPM support. However, the encryption key information is still located within the Linux kernel since the Enforcer LSM itself is executed in the Linux kernel. Thus, a isolation of encryption keys and operations from the operating system is not supported.

Device Mapper Crypt Target

The Device Mapper is a Linux 2.6 kernel feature that allows to create a virtual block device whose sectors are mapped to sectors on a physical block device, e.g. a hard-disk or USB device. Available mapping types include encryption. Thus data written to the virtual device is transparently encrypted and passed on to the physical device (and vice versa). The crypt target (`dm_crypt`) uses the Linux 2.6 Cryptographic API which provides state-of-the-art symmetric ciphers and hash computation algorithms such as AES and SHA-256.

However, since the crypt target is a kernel feature, the encryption keys and operations are located within the kernel and there is no isolation from the operating system. Furthermore, there are no measures for checking the system integrity before execution.

The EMSCB Project

The European Multilaterally Secure Computing Base (EMSCB) project aims at developing a trustworthy computing platform, based on open standards and open source, that solves many security problems of conventional platforms [9]. The platform deploys

- hardware functionalities provided by Trusted Computing,
- a security kernel, and
- an efficient migration of existing operating systems.

The EMSCB platform allows, in the sense of multilateral security, the enforcement of security policies of different parties, i.e., end-users as well as industry. This is a vital property required for secure execution of a variety of distributed applications. Consequently, the platform enables the realization of various innovative business models, also in the area of Digital Rights

Management, while averting the potential risks of Trusted Computing platforms concerning privacy issues. The source code of the EMSCB platform will be published under an open-source license, e.g., the GPL. The platform can be freely used as basis for application development.

The EMSCB project is partly funded by the German Federal Ministry of Economics and Technology. Project partners include several universities and industry organizations. This consortium is lead by Ruhr-University Bochum (Applied Data Security Group)[1].

Basic System Architecture

One main design goal of EMSCB is the realization of a minimal and therefore manageable, stable and evaluable security kernel for conventional hardware platforms such as PCs, servers, embedded systems, and mobile devices like PDAs and smartphones. This requirement is fulfilled by extracting security-critical operations and data and integrate them into the security kernel [10]. The basic architecture is shown in Figure 1.

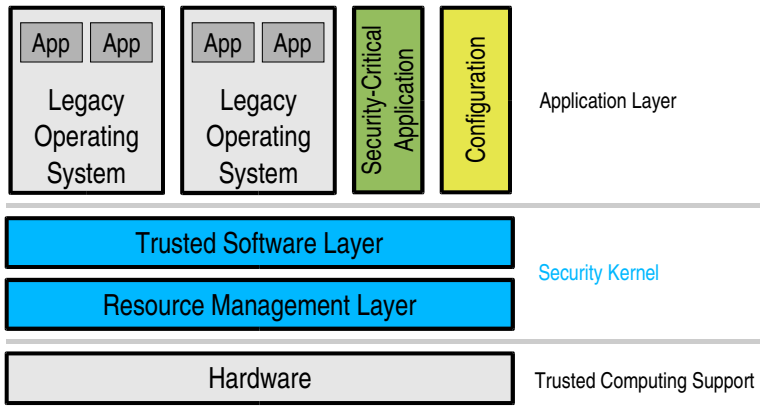

Figure 1: Basic system architecture

The security kernel is composed of a Resource Management Layer, which runs on top of the hardware, and a Trusted Software Layer. The hardware may provide Trusted Computing functionality, e.g., based on TPM. The main task of the Resource Management is the provision of an abstract interface of the underlying hardware resources like interrupts, memory and hard-disk drives. Moreover, this layer allows to share these resources and can realize access control enforcement on the object types known to this layer. This layer can be implemented using a microkernel (e.g. [11]) or a hypervisor virtualization (e.g. [12]) approach.

The Trusted Software Layer combines the services provided by the hardware layer and the resource management. It extends the interfaces of the underlying services with security properties and ensures isolation of the applications executed on top of this layer.

On top of the Trusted Software Layer, security-critical and non-critical applications are executed in parallel. Legacy operating systems can be executed as isolated applications on top of the Trusted Software Layer to

1 http://www.prosec.rub.de

provide end-users a common user interface and a backward-compatible application binary interface (ABI) and allows application providers to reuse existing non-critical applications and components.

Secure Linux Device Encryption

The secure Linux device encryption system is called “Turaya-Crypt” and is based on the microkernel-based EMSCB security kernel. The Linux operating system is executed as a separate EMSCB application. This allows an architecture where the key critical information of a device encryption system is stored and handled in a special EMSCB service outside of Linux. This special service is the HDD-Encrypter as shown in Figure 2.

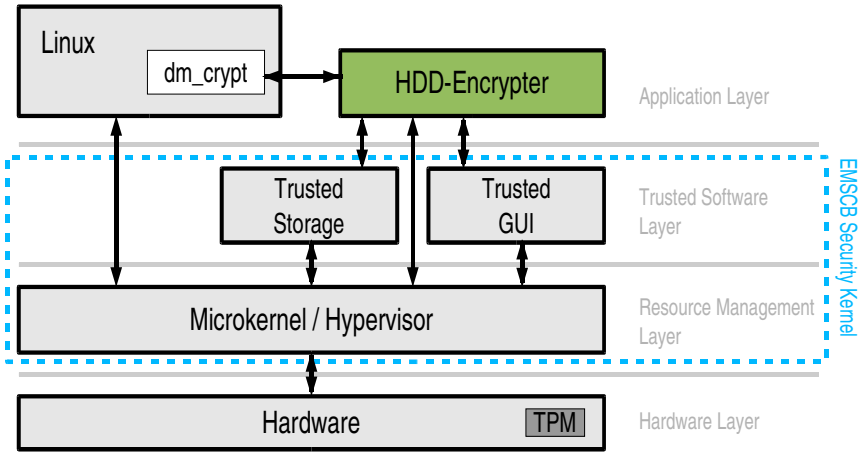

Figure 2: Architecture of the secure device encryption

All key critical information is handled by this service, that itself is fully independent from Linux. After a successful authentication process against the HDD-Encrypter, a Linux function that handles the device encryption just sends the plain text to the HDD-Encrypter service and receives the cipher text afterwards and vice versa without having access to the secret key information. We use the `dm_crypt` interface of Linux so that the device mapper support can be used transparently within Linux.

The authentication process simply authenticates a qualified user, i.e. the data owner, and then provides access to the data to all applications of the respective user. The authentication is performed by providing a password, which is then used to derive an encryption key. Without the correct password the correct encryption key will not be accessible and hence confidentiality is preserved.

We use AES as a fast symmetric encryption algorithm in our implementation. We derive the key from a given password using a cryptographic hash function.

Turaya-Crypt can be run in three operational modes:

- Single-user mode (without Trusted GUI)
- Single-user mode (with Trusted GUI)
- Multi-user mode (with Trusted GUI)

In single-user mode all encrypted devices are encrypted with one single key, which is derived from the single user's password. In multi-user mode every encrypted device has its own individual encryption key. The user's password is used to derive another encryption key, which is used to encrypt/decrypt the encryption key of the device. This allows multiple users to share a common encrypted device but having not to share a common password.

In multi-user mode it is necessary to define keys and user accounts. If users want to have access to certain encrypted devices their access rights to these resources, i.e., the cryptographic keys, must be specified. Thus, there is a need for user management, which is handled by Turaya-Crypt as well.

Note, that all key creation and management is handled outside of Linux. Linux does not see any difference whether Turaya-Crypt runs in single-user or multi-user mode. This is due to the usage of the `dm_crypt` interface of Linux. Within Linux, encrypted devices or partitions are created and used as normal as it would be when using the device mapper crypt target directly.

Trusted GUI

When using the Trusted GUI, Linux runs in an extra window. The password and administration dialogs for accessing the device encryption keys or changing the configuration are displayed in a separated dialog box. On the one hand, the user can recognize that the password or administration dialog does not belong to any potentially malicious application inside the Linux operating system (trusted path to application). On the other hand, Linux is not able to access or manipulate these dialogs, either. Figure 3 shows a screenshot.

For our prototype implementation we used a special GUI system [13] that provides a virtual framebuffer to the Linux system. Linux applications draw

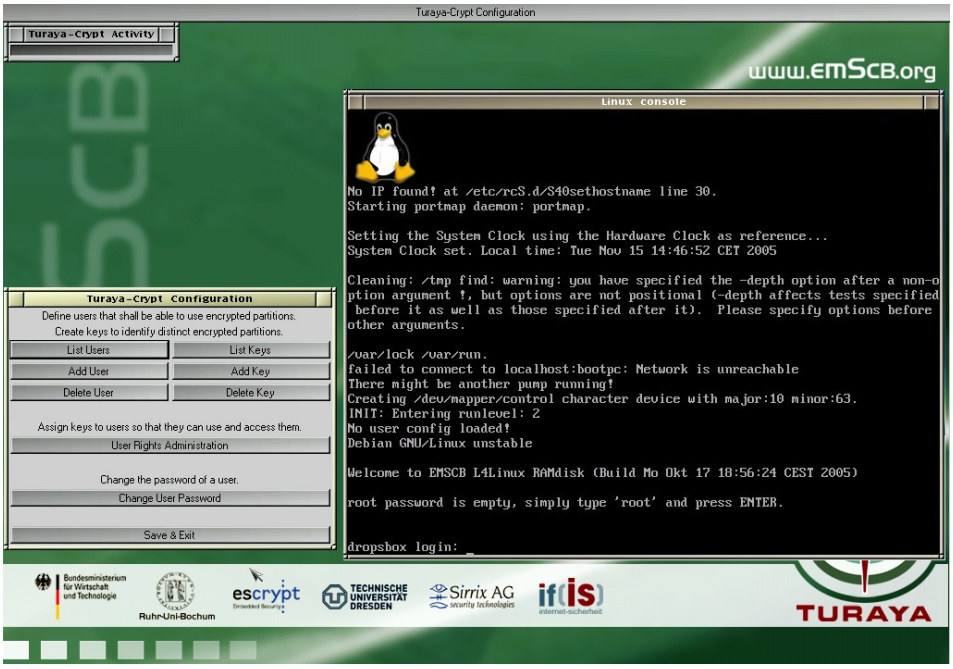

Figure 3: Screenshot of the multi-user mode with Trusted GUI showing the configuration management dialog and Linux in a separate window.

their graphical user interface elements within this framebuffer. Security-critical applications, like the configuration management console of Turaya-Crypt, have separated GUI windows that are isolated from the Linux system. Currently, this system is going to be improved to provide a secure GUI, e.g. [14].

In single-user mode we do not need a special trusted GUI since the bootloader will already ask for the password that is used for key derivation. The bootloader will automatically pass the password to the HDD-Encrypter service. After the system is booted there is no need to ask for the password again. All devices will be encrypted/decrypted with the key derived from the given password. Thus, Linux can be executed in full-screen mode in this case.

Trusted Computing Support

Our proposed system is able to bind device encryption keys to a user authorization secret, hardware components or the trusted software modules. Binding to hardware and/or software components requires a trusted hardware component. Our architecture deploys TPM sealing functionalities for this purpose. However, the architecture is not restricted to using the TPM and can offer the corresponding interfaces of any other hardware platform.

The TPM uses on-chip registers (Platform Configuration Registers, PCRs) to securely store measurements (i.e., hash values) of hardware and software components. The TPM sealing command subsequently binds data to these PCRs. The resulting binary data is then stored persistently.

For our application certain PCRs should reflect the integrity of the trusted components. This can be achieved as follows:

1. A TPM-aware (trusted) BIOS measures the MBR (Mater Boot Record) before execution.

2. The bootloader measures each boot stage before execution.

3. The bootloader is completely loaded. The PCRs now reflect the integrity of the boot process (authenticated boot).

4. The trusted software components are digitally signed. The bootloader checks their signatures before execution. The corresponding public key is hard-coded into the bootloader. If a signature check fails the PCR values are invalidated and the user is requested for interaction (secure boot).

The alternation of authenticated and secure boot allows secure updating of system components without “resealing” of secrets [1],[8].

We use TrustedGRUB[1] as bootloader, which implements the boot mechanism as described.

The Trusted Storage component within the Trusted Software Layer is

1 http://www.prosec.rub.de/trusted_grub.html

responsible for securely and persistently storing the cryptographic keys and user authorization information needed for the device encryption. Therefore, we make use of Trusted Computing functionality to bind the data to a certain platform configuration. That means, the cryptographic keys and the configuration data will only be accessible when the system is in a certain configuration, i.e., running on the defined hardware platform and not being modified in an unauthorized way.

Linux Integration

Currently we use Linux 2.6 as a legacy OS in a modified version that is able to run on a microkernel. It is binary-compatible with the normal Linux kernel and can be used with any PC-based Linux distribution.

To encrypt all data transferred to a physical block device, e.g., a hard disk partition, a virtual block device is created and mapped to this physical device. Whenever the Linux file system driver writes a data block to the virtual device, the block is passed to a wrapper cipher algorithm integrated into the Linux Cryptographic API. We reused the Linux device mapper with `dm_crypt` target for this redirection.

The wrapper cipher transfers the blocks to the HDD-Encrypter service by using an inter-process communication (IPC) call mechanism of the microkernel. The HDD-Encrypter encrypts (or decrypts) the block and returns the result back to the wrapper cipher. Thus the encryption key and associated operations are completely isolated from the legacy OS.

The wrapper cipher is implemented as a Linux kernel module. This allows us to reuse the `dm_crypt` interface and the corresponding Linux commands for configuring and using encrypted devices, i.e., the `cryptsetup` command.

While `cryptsetup` usually requires to specify the encryption algorithm and the password that is used to derive the encryption key, we use this interface to specify our wrapper cipher module. As previously mentioned, the password is entered in a special dialog of the bootloader or the HDD-Encrypter within the Trusted GUI.

Conclusion and Outlook

We have introduced the EMSCB Security Architecture which is used as a trustworthy basis for implementation of secure distributed applications. Within the EMSCB project several application prototypes are being designed and developed.

In this paper we have introduced the prototype for a device encryption system based on the EMSCB security kernel. We have shown that it is possible to build a secure and isolated device encryption system while being interoperable with a legacy OS and its standard applications.

We are currently completing and improving the implementation with respect to system integrity protection and TPM integration. Furthermore, we are working on new improvements of the trusted GUI to provide user-friendly easy-to-use and secure user interfaces.

References

[1] J. Marchesini, S.W. Smith, O. Wild, A. Barsamian, J. Stabiner, "Open-Source Applications of TCPA Hardware", *20th Annual Computer Security Applications Conference (ACSAC 2004)*, ACM, 2004.

[2] Trusted Computing Group, *TPM Main Specification*, Version 1.2 rev. 85, Trusted Computing Group, 2005.

[3] Safeboot N.V., *SafeBoot Device Encryption for PC*, 2005, http://www.safeboot.com/products/device-encryption/pc/

[4] Microsoft Corp., *Secure Startup - Full Volume Encryption: Technical Overview*, 2005, http://www.microsoft.com/whdc/system/platform/pcdesign/secure-start_tech.mspx

[5] PGP Corporation, *PGP Whole Disk Encryption for Enterprises Data Sheet*, 2005, http://download.pgp.com/pdfs/PGP-WDE-Enterprises_DS_050919_F.pdf

[6] Utimaco Safeware, *Security for Mobile PCs and Data Media - SafeGuard Easy Whitepaper*, 2005, http://www.utimaco.com/C12570CF0030C00A/vwContentByKey/W26L6EHK398CCHEEN

[7] Rich MacDonald, Sean Smith, John Marchesini, Omen Wild, *Bear: An Open-Source Virtual Secure Coprocessor based on TCPA*, TR2003-471, Department of Computer Science, Dartmouth College, 2003.

[8] John Marchesini, Sean W. Smith, Omen Wild, Rich MacDonald, *Experimenting with TCPA/TCG Hardware, Or: How I Learned to Stop Worrying and Love The Bear*, TR2003-476, Department of Computer Science, Dartmouth College, 2003.

[9] EMSCB Project Consortium, *The EMSCB project*, 2006, http://www.emscb.org

[10] Ahmad-Reza Sadeghi, Christian Stüble, Norbert Pohlmann, "European Multilateral Secure Computing Base - Open Trusted Computing for You and Me", *Datenschutz und Datensicherheit DuD 28 (9)*, Verlag Friedrich Vieweg & Sohn, 2004.

[11] Jochen Liedke, "On Microkernel Construction", *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP'95)*, ACM, 1995.

[12] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, "Xen and the Art of Virtualization", *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, ACM, 2003.

[13] N. Feske, H. Härtig, "Demonstration of DOpE - a Window Server for Real-time and Embedded Systems", *Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS 2003)*, IEEE, 2003.

[14] N. Feske, C. Helmuth, "A Nitpicker's guide to a minimal-complexity secure GUI", *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC 2005)*, ACM, 2005.