IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

SECURITY IN ANDROID FILE SYSTEM

Miss Kirti . P. Lokhande
ME(CSE)
G.H. Raisoni College of Engineering,Amravati

Dr. Mahip M. Bartere
Asst. Prof. (CSE)
G.H. Raisoni College of Engineering,Amravati

ABSTRACT

Recently smart gadgets including smart phones and tablets are gaining popularity. These devices contain Personal Identifiable Information. An attacker can compromise a smart phone and gain full control of it by connecting another computing device to it using the USB physical link. Moreover, by simply capturing the smart phones physically, adversaries have access to confidential or even classified data if the owners are the government officials or military personnels. In this project, here discuss a secure and encrypted file system on Android operating system and optimize the performance using certified encryption algorithm BlowFish provided in OPENSSL libraries. This project presents EncFS which is a FUSE (File system in USErspace) based file-system offering encryption file system to protect the removable and persistent storage on heterogeneous smart gadget devices running the Android platform. In proposed methodology, the data at rest including physical partition on the device and removable storage card is encrypted using user provided password. The encrypted file system is mounted only after successful password verification with user at system boot up.

Keywords: Smart handheld devices, Full disk encryption, Encrypted filesystem, *Blowfish, Encryption, Decryption.*

1. INTRODUCTION

Technology trends in both hardware and software have driven the hardware industry towards smaller, faster and more capable mobile hand-held devices that can support a wider-range of functionality and open source operating systems. Mobile hand-held devices are popularly called smart gadgets (e.g. smart phones, tablets, e-book readers). The smart gadget life-cycle has evolved drastically in recent years. These new generations of the smart gadget devices such as the iPhone and Google Android devices are powerful enough to accomplish most of the tasks that previously required a personal computer. However, smart gadgets have to come a long way in terms of security. Organizations have come to realize that these commercially available smart gadgets will soon have to serve as an integral part of their operations. This requires a level of security that allows for security of data at-rest and on the move to support secure communications. Major challenges in providing file system encryption in mobile devices are:

1. A major obstacle is that there is a serious lack of National Institute of Standards (NIST) approved encryption algorithms on these commercially available smart gadgets.
2. On smart gadgets where resources, like the battery, are very limited, it is important to keep a low footprint on such solutions.
3. The expectation for each individual application to support encryption runs into the key management problem: other applications in the system can potentially gain access to the key and render the encryption useless.
4. This system has to be ubiquitous and integrate into the ecosystem of smart gadgets with minimal maintenance and installation cost.

Therefore, there is a need for a practical approach to build common security libraries that operate at the operating system level and provide strong encryption. In this project, the focus is on analyzing the performance for persistent storage protection using encryption on smart gadget devices. The project uses EncFS which is a FUSE

(File system in USErspace) based encryption file system and uses certified cryptographic algorithms to store encrypted versions of every file in a source directory. The volume key is decrypted using a password supplied by the user. This is different from full-disk encryption software because the protected data is mounted in memory at a specified mount point in the file system. Since file I/O operations on the mount point eventually hit the encrypted copy of the file on the underlying file system, various performance optimizations can be possible by adjusting the file system parameters such as block size, buffer size. Finally, we discuss the limitations of file system encryption and demonstrate that it is feasible on smart gadget devices with a reasonable performance overhead. Additionally, this project leverages advantages provided by underlying operating system:

- Use of NIST validated cryptographic libraries [8]which are not implemented in kernel-space.
- The proposed implementation can be extended to different hardware with negligible effort.
- This project needs to focus on data encryption without having to deal with other aspects of file system design. Underlying file systems like ext3 and yaffs2 already have strong support for handling data-corruption and journaling.

2. SYSTEM OVERVIEW STUDY

2.1 Android Architecture

Android architecture is composed in layers. These are the application layer, application framework layer, Android runtime and system libraries. Applications are composed of one or more di_erent components. There are four types of components namely activities, services, broadcast receivers and content providers. Activities include a visible interface
of the application. Service components are used for back- ground processing which does not require a visible interface. The broadcast receiver component receives and responds to
messages broadcast by application code. Finally, content providers enable the creation of a custom interface for stor- ing and retrieving data in di_erent types of data stores such as _lesystems or sqlite databases. The application frame- work layer enables the use or reuse of different low-level components. Android also includes a set of system libraries,
which are used by different components of Android. Components of an application can interact with other components { both within the application and outside it { using a specialized inter-component communication mechanism based on Intents. Intent is \an abstract representation of an action to be performed" [4]. Intents can either be sent to a specific component {called explicit intents { or broadcast to the Android framework, which passes it on to the appropriate components. These intents are called implicit intents and are much more commonly used. Both of these types share the same permission mechanism and for the sake of clarity, we only consider implicit intents in this paper.

2.2 *EncFS for Android:* EncFS is selected as the basis for our encryption filesystem. We introduced the filesystem in Section III-A. Three major components are required to make EncFS work on any platform: kernel FUSE library support, user space *libfuse*, and EncFS binaries. To make an encryption file-system work on Android, a modified bootstrapping process and password login was integrated into the operating system framework. EncFS uses standard OpenSSL cryptographic libraries in userspace. This gives us various advantages over using a kernel-based cryptographic library. Some of the features of our solution verses other in-kernel encryption approaches [1], [2] are as follows:

- By using EncFS our system is backward and forward compatible with existing and future Android versions. Since *libfuse* and *libc* are stable across different versions of Android and different devices, only minimal engineering efforts are needed (if any) to make EncFS work on other variations of Android-based smart devices.
- EncFS leverages OpenSSL suite as the crypto engine. The OpenSSL libraries, libcrypto and libssl, implement various cryptographic algorithms that are validated and in compliance with FIPS 140-2 Level:1 standard [3].
- In addition, our approach supports all underlying filesystems, including *yaffs2*, *ext4* and *vfat*.

In order to build EncFS, we created a package with the components described below. It is required to root the phone in order to prepare it with the modified kernel, system binaries and java framework patches discussed below. Once installed, EncFS does not need processes or applications to run as root, in order to encrypt their data. The userspace will function without knowledge of any change in the underlying layers.

2.5 Kernel FUSE support: In general, FUSE module provides a bridge to the actual kernel interfaces. However, the Android Linux kernel does not support FUSE file-systems by default to eliminate redundant functionalities that are not required by Android. Most Android devices, including the Nexus S which we use, do not come with the FUSE modules enabled in the kernel. We obtain the kernel source code from Google's Android Open Source Project (AOSP) website and enabled the kernel FUSE modules necessary for *libfuse* to
run. We then flash our device with this customized kernel.

2.6 Libfuse: As the fundamental supportive library for all FUSE-based file-systems, *libfuse* is not officially supported in the Android system. Furthermore, The Bionic C library in Android is missing glue layer code for interfacing VFS (Virtual FileSystem in Linux) and FUSE. We patched the Bionic C library with missing header files (*statvfs*) and
corresponding data structures that are required for *libfuse* version 2.8.5.

2.3 EncFS: By building the EncFS sources for the ARM architecture, we created the executables that would enable us manage the EncFS filesystem. In addition to *libfuse*, EncFS also depends on the boost library which is a widely adopted C++ library[9], *librlog* for logging[10] and *libcrypto/libssl* for
cryptographic primitives. We patched boost library version 1.45 which is the current-to-date version as of this development and built it against Android Bionic C library. The *librlog* is versioned at 1.4 while the OpenSSL suite included with Android 2.3 is 1.0.0a. EncFS supports two block cipher algorithms.

2.4 User Interface: Normally, the Android framework loads the user interface by unpacking the applications and other files from */system* and */data* partitions. The */data* partition contains all the user-installed applications and all other userspecific data. In our implementation, this */data* partition contains only a skeleton of the required folders which won't be used by the users for actual data. Store the encrypted data in a separate directory and mount it over */data* partition when the user supplies the password. We modified the Launcher application in Android framework to accept this password, which is the key for the encrypted version of the */data* partition. If the password provided by the user is valid, EncFS mounts the encrypted data partition on */data* mountpoint using FUSE. If the mount is performed successfully, the Launcher will call a dedicated native program installed by us to *soft* reboot Android Dalvik environment and the user is presented with his encrypted
userdata partition, decrypted into the memory transparently. The user has limited number of login attempts. If the failure attempts accumulates to a predefined threshold value
(10 in our case), the Launcher program will erase all the data.

2.7 Android: Android is an mobile phone platform developed by the Google-led Open Handset Alliance (OHA).2 The platform quickly became popular amongst the developer community for its open source nature and adoption by telecommunications providers world-wide. While Android is based on Linux, the middleware presented to application developers hides traditional OS abstractions. The platform itself focuses on applications, and much of the core phone functionality is implemented as applications in the same fashion used by third-party developers.

Android applications are primarily written in Java and compiled into a custom byte-code (DEX). Each application executes in a separate Dalvik virtual machine interpreter instance running as a unique user identity. From the perspective of the underlying Linux system, applications are ostensibly isolated. This design minimizes the effects of a compromise, e.g., an exploited buffer overflow is restricted to the application and its data [6].

2.8 Crytography

Cryptography is the science that is widely used for the network security. Key aspects of cryptography are privacy, authentication, identification, trust and verification [2]. There are several ways of classifying cryptographic algorithms. They can be classified based on the number of keys that are employed for encryption and decryption, and further defined by their application and use The cryptographic algorithms can be broadly divided into three types namely Secret Key Cryptography (SKC), Public Key cryptography (PKC) and Hash Functions Some of the secret key algorithms are Data encryption standard (DES), Advanced encryption standard (AES), CAST, International data encryption algorithm(IDEA), Blowfish, Twofish, and
Secure and fast encryption routine(SAFER). In these algorithms AES and Blowfish are the tw o

algorithms proved to be strong in the modern world. RSA, Diffie- Hellman, Digital signature algorithm (DSA), Elgamal and Elliptic curve cryptography are some of the Public key cryptographic algorithms [3].

A. Cryptography Goals:

There are five main goals of cryptography.Every security system must provide a bundle of security functions that can assure the secrecy of the system. These functions are usually referred to as the goals of the security system. These goals can be listed under the following five main categories (Earle, 2005):

- Authentication: The process of proving one's identity. This means that before sending and receiving data using the system, the receiver and sender identity should be verified.
- Privacy/confidentiality: Ensuring that no one can read the message except the intended receiver. Usually this function is how most people identify a secure system.
- It means that only the authenticated people are able to interpret the message content and no one else.
- Integrity: Assuring the receiver that the received message has not been altered in any way from the original. The basic form of integrity is packet check sum in IPv4 packets.
- Non-repudiation: A mechanism to prove that the sender really sent this message. Means that neither the sender nor the receiver can falsely deny that they have sent a certain message.
- Service Reliability and Availability: Since secure systems usually get attacked by intruders, which may affect their availability and type of service to their users. Such systems provide a way to grant their users the quality of service they expect.

3. Conclusion

In this paper, presented an implementation of a portable filesystem encryption engine that uses NIST certified cryptographic algorithms for Android mobile devices. We offer a comparative performance analysis of our encryption engine under different operating conditions and for different loads including file and database (DB) operations. By optimizing the filesystem block-size and I/O mode, we were able to gain 20% to 57% performance. There are various encryption algorithm available like AES, DES, 3DES, blowfish. Therefore, we conclude that
our encryption engine is easily portable to any Android device and the overhead due to the encryption scheme is an acceptable trade-off for achieving the confidentiality requirement.

REFERENCES

[1] "Android honeycomb encryption," http://source.android.com/tech/encryption/android crypto implementation.html.
[2] "Whispercore android device encryption," http://whispersys. com/whispercore.html.
[3] "Openssl fips 1402 security policy, version 1.2."

[4] Independent Security Evaluators, "Exploiting android," http://securityevaluators.com/content/case-studies/android/ index.jsp.

[5] S.Pavithra, Mrs. E. Ramadevi "STUDY AND PERFORMANCE ANALYSIS OF CRYPTOGRAPHY ALGORITHMS " International Journal of Advanced Research in Computer Engineering & Technology Volume 1, Issue 5, July 2012 14, pp.82-86

[6] Independent Security Evaluators, "Exploiting android," http://securityevaluators.com/content/case-studies/android/ index.jsp.
[7] J. P. Anderson, "Computer security technology planning study, volume II," Deputy for Command and Management Systems, HQ Electronics Systems Division (AFSC), L. G. Hanscom Field, Bedford, MA, Tech. Rep. ESD-TR-73-51, October 1972.
[8] "Fips pub 1402, security requirements for cryptographic modules." [Online]. Available: http://csrc.nist.gov/publications/ fips/fips1402/fips1402.pdf
[9] "Boost c++ library," http://www.boost.org/. [Online]. Available: http://www.boost.org/.
[10] "Librlog," http://www.arg0.net/rlog.