AN ENCRYPTION KEY MANAGEMENT APPROACH FOR CONFIGURATION FILES

MOUKOUOP NGUENA Ibrahim*, MOUPOJOU MATANGO Emmanuel**‡, ATSA ETOUNDI Roger**

*National Advanced School of Engineering, Faculty of Science, University of Yaounde I
**Department of Computer Science, Faculty of Science, University of Yaounde I

‡MOUPOJOU MATANGO Emmanuel; Yaounde, Cameroon: Tel: +237 673 80 78 64, e-mail: moupojouemma@yahoo.fr

Abstract—One of the major points of interest in software engineering nowadays is how to ensure applications configuration files' security. In fact, they very often contain confidential information as database connection credentials, thereby providing a vulnerability point to these applications, for those files have their information clearly written. Some studies upon 2200 applications revealed that 96% of them were vulnerable, and that 80% of those vulnerable applications contained vulnerabilities exposed by incorrect configuration information management. Encryption is then used to secure these delicate files. The difficulty then lies in the usage and backup of the encryption key so as to guarantee data security. To do this, current approaches are either to hide the encryption key in the application source code or somewhere on disk, it's safety then being compromised; or to protect the key by bounding it to a specific user account, the application can then operate only within this account, obligating that user to be physically present for the availability of the key, which is an unacceptable constraint for critical systems. In this paper, we propose an encryption key management model solving limitations mentioned above. The key lies only in main memory, which is great for its protection; it is submitted to the files security module when starting using a secure and flexible way (directly, through https or SMS).

Keywords—Software engineering, configuration file, security, encryption key.

1. Introduction

A configuration file is a file containing configuration information used by a computer program to adapt or customize its operations. A configuration file may contain information such as connection settings to a database (login, password, port number), communication protocols, language preferences etc. In most cases, since these information are clearly written, they should never fall into the hands of ill-intentioned people, because they give critical information about the design and operation of applications using them: this is a large security hole. For example: Joomla!, Ciel Gestion Commerciale, OpenERP or MDAL [1] applications have their information clearly written in their configuration files.

The report [15] of Cenzic Inc. reveals that 96% of tested applications have vulnerabilities. On the

1. Megasoft Data Access Library: written in Java Framework, facilitating the development, deployment, operation and maintenance of applications accessing databases

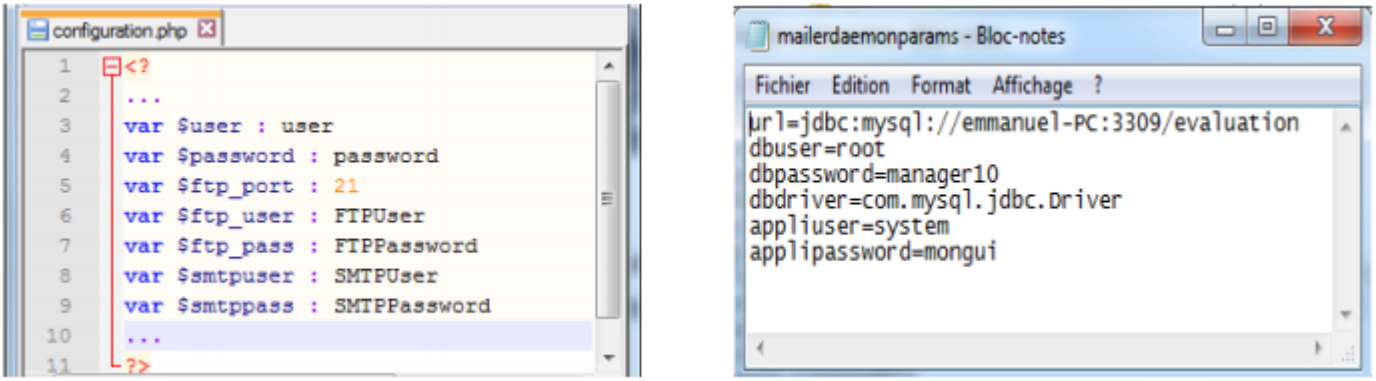

Fig. 1. Joomla! and MDAL configuration files

other hand, the 2013 Cyber risk report of HP [12] states that, upon 2200 tested applications, 80% of vulnerable applications contain vulnerabilities exposed by incorrect configuration, and not by their source code. In fact, attackers exploit misconfigured servers or access to server configuration files, enabling further, more sophisticated attacks, for unsecured configuration files. An incorrectly configured application can be just as dangerous as an incorrectly coded one, assets the authors Savita B. et al.; Ketki and Bryan Sullivan [19][30][7]. For example, Jamie Riden et al. demonstrate how it is possible, by a basic code injection, to download a web application configuration file[25].

One of the biggest malware news stories of these last years took place on 20 March 2013 in South Korea[12]. In a timed and coordinated attack, a malware payload was executed on about 48000 computers belonging to a number of targeted businesses and organizations, effectively crippling them for some time. The attack was made possible because of the vulnerability offered by configuration files which where unsecured or secured by vulnerable tools as mRemote or SecureCRT. In fact, Eric Romang, Raffaele Addesso and Cosine Security demonstrate how is it possible to get passwords protected by mRemote or SecureCRT[26][29][2].

Cryptography[2] is then used to further secure these files, over existing measures [3]. The difficulty is then the choice and saving of the encryption key to guarantee optimal data security, for the application has to know the key used for encryption, in order to read or write information on these configuration files.

Many authors like A. Menezes et al. then agree that the optimal **encryption key management**[4] to ensure data security is the main issue in cryptography [17][18][3][20][11]. Kerckhoffs declares: “The security of a cryptographic protocol must be guaranteed by cryptographic keys used for encryption and decryption, and not by the encryption method”[17] [22]. The security of the system then depends on the confidentiality of the cryptographic keys.

Encryption key management in this document will mainly refer to their storage and usage.

2. Science of transforming a file according to a given algorithm to make it unreadable without Conversely algorithm applied.

3. Physical protection of server stations, definition of access rights for user accounts, implementation of .htacces control for web servers...

4. Activities involving the handling of cryptographic keys and other parameters related to safety throughout the life of the key. Ie their production, storage, use and destruction

2. Materials and Methods

2.1. Existing Encryption Key Management Approaches

2.1.1. Encryption key in the source code of the application or in a file on disk

Some approaches advocate placing the encryption key in the source code of the application[31][28], in a file on disk[3], or else to use as key a feature of the server machine [24](Hard drive id for example); according to Keith Martin **the key security is then compromised** [16]. Indeed, an intruder having the application and encrypted configuration files could decrypt them; or make reverse engineering on the application code to get the key. If the key is stored on disk, he could access them on an easier way, by code injection for example[25].

In addition to their respective disadvantages, the following three approaches have in common the fact that they require the administrator to be **physically** present whenever the system boots so that he submits the key [24], which according to the authors Elaine Barker et al., is an unacceptable constraint for important systems [3].

2.1.2. Encrypting encryption key

This technique consists in encrypting the encryption key itself with a key called KEK [5] [5]. This new key is not stored on disk, but is generated each time we want to make use for encrypting or decrypting the master key. Furthermore, this new key is obtained through the PBE[6] mechanism, comprising inputting a password and a code for outputting a key. According to Keith Martin, the problem is reduced to the **management of that other password and the code** [16], which must be identical to produce the same KEK [6][1].

2.1.3. Backup the encryption key on an external drive

To protect the encryption key, the administrator could make use of a safety equipment[28][34] like a HSM [7]. These are electronic components more or less secured, which are actually digital saves for secured storage of critical elements such as cryptographic keys. According to authors Elaine Barker et al., the disadvantages are that **the external support could pose a security risk[8], fall into evil hands, get lost, or cease to function**[3].

2.1.4. Encryption key protected by the operating system

This approach consists in using the protection mechanisms of user's accounts of the operating system to protect the encryption key [24]. This may be, for example, a user environment variable containing the encryption key [27]. In the case of ASP. Net framework, Microsoft uses a key pair for encrypting and decrypting the "web.config" configuration file [21]. This key pair is in turn encrypted using a key derived from the user's password[3]. The application is therefore forced to operate within a specific user account; if there is not real VPN or internet connection allowing the user session to be started remotely,

5. Key Encryption Key
6. Password Based Encryption
7. Hardware Security Module

then it requires the physical presence of the account holder to start it when necessary, otherwise nothing works. To remedy this, it was proposed to define a machine-level key container, which is not recommended [8]. However, this approach remains platform-specific (windows) and technology-specific (ASP .Net).

2.1.5. Configuration files protected by vulnerable tools

As mentioned in introduction, tools such as mRemote or SecrureCRT are used to protect configuration files. Eric Romang, Raffaele and Addesso Cosine Security unfortunately demonstrate how it is possible to recover passwords protected by these tools [26][29][2]. Some applications in the hacked machines in the introductory example were protected by these tools, but the malware still had access to the configuration information. Note however that mRemote has recently been withdrawn from circulation because of this flaw.

The main limitation of these existing approaches is the violation of the third and fourth Kerchoffs's rule[17] stipulating that: *The key shall be communicated and retained without using written notes, and shall be modifiable at any moment; and that the system must be applicable to telegraphic correspondence.* The six rules established by Auguste Kerchoffs for a cryptographic system will be used later for the validation of the model proposed here.

8. RSA key information may persist even when the Windows user profile is deleted

2.2. Proposed Method: Keeping Encryption Key in Main Memory

To protect configuration files data, the idea is to make them incomprehensible to anyone who even succeeded to get them. Cryptography is then use to protect those data.

The model proposed here is the definition of a new method of configuration files encryption key management. Specifically, it will set the policy of the secured encryption key management and the interactions with the security module[9] proposed here, knowing that this key can be remotely transmitted.

As mentioned previously, this model should:

1. Avoid linking the encryption key to a specific user account.

2. Never store it to disk, but keep it in main memory in order to protect it.

3. Allow the remote submission of the encryption key using a secure and flexible way.

2.2.1. Solution architecture

The objective is to move from the configuration of Figure 2 to the one of Figure 3, below:

In Figure 3, **"Configuration Files Securer" (CFS)** designates the component responsible of encrypting and decrypting configuration files. When the application needs to access a configuration file, it uses that component. The component then decrypts the file and sends the result back to the application. If the application needs to write data in a configuration file, it also works with the component by sending the data to write in the

9. The one in charge of reading/writing data in configuration files

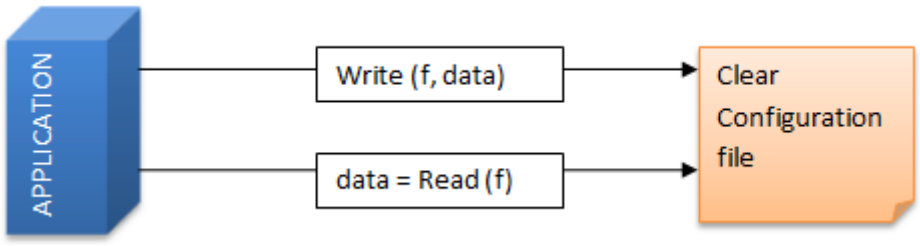

Fig. 2. Classical Access to configuration files

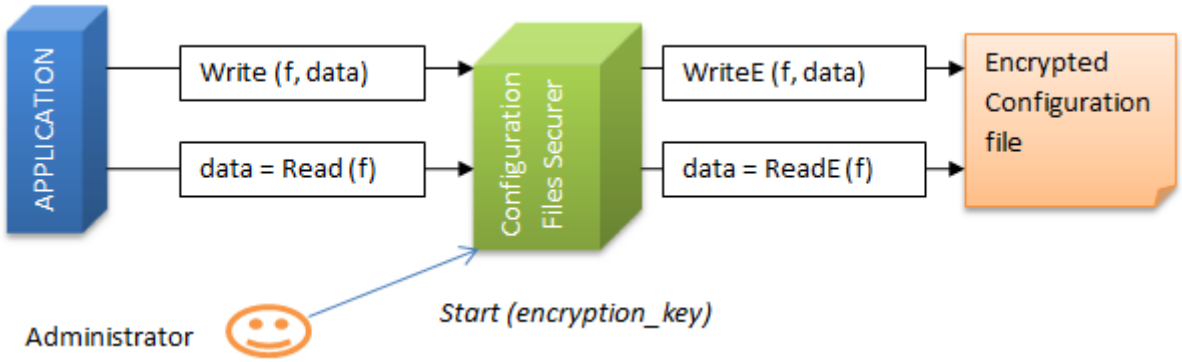

Fig. 3. Secured access to configuration files

file. The CFS writes data in the file and encrypts it.

To perform its task, the CFS takes as input the encryption key sent by the system administrator, at the startup of the module. This key is never stored on disk, but remains exclusively in main memory. When the administrator is not present in front of the server machine at the startup, it's possible to remotely transmit the key in several ways, depending on whether he is connected to the network or not, while ensuring key security.

2.2.2. Risks identification

Data in encrypted configuration files must be decrypted before their use by applications. This fact implies a symmetric encryption (using a unique key). The problem to solve here then is the encryption key protection and accessibility[9]. Indeed, once the configuration files are encrypted, they must be decrypted so that applications can continue to read or to write configuration information in these files. This should be done using the same key used for encryption.

When thinking about a way of managing the encryption key, we must also think about pirate decoding attempts. It must be clearly understood that an intruder will try to decipher the key, he will perhaps have access to the source code of the application. To ensure the reliability of the proposed encryption key security solution, a way to eliminate the risks of hacking that key must be provided. In this context, three enemies have been identified:

1 The one having access to the server machine and the application source code (intruder 1)
2 The intruder who intercepted the encryption key when submitted remotely (intruder 2)
3 The intruder combining 1 and 2 (intruder 3)

2.2.3. Model description

The encryption key is never stored on disk, but remains only in main memory (intruder 1 disabled)[32][28]; it's then secured. An attack attempt could be to suddenly restart the machine, so as to cause a dump of the main memory content to disk, in order to recover the key later from disk. This potential vulnerability is solved by prohibiting dump of memory areas containing sensitive information [32]. Here, however, the intervention of the administrator or RCO [10] is needed to start the configuration files manager. He is notified by a message sent to him by the security module at the startup, when waiting for the key. Doing this reduces the probability for the system to remain in a locked state because of the administrator's absence at the startup. If he is not physically present to start the service, an Https or an SMS channel for remote transmission of the encryption key is made available, thus allowing the start of the service. This GSM solution for the key transmission is very useful for countries where GSM network is always present, rather than internet connection (this is the case for many African countries). More, here, the application can operate under any user account unlike OS solution.

1 **Encryption Key Transmission Interfaces**
The proposed security Component has three interfaces through which the administrator can submit the encryption key. At startup, the security module instantiates each of these interfaces, and remains awaiting the key listening on each of them. Upon receiving the encryption key through one of these interfaces and checking that it is correct, it then closes them all and can start operating. Those interfaces are:

a) **Direct or Window**
This interface is useful when the administrator actually is in front of the machine at the startup. A window is displayed and he can directly enter the encryption key.

b) **Https**
Used when the administrator is not physically present at workstation, but has a network connection. He can then run the web page built and deployed for this purpose, and submits the encryption key . This key is securely transmitted to the security module, which closes all interfaces after receiving it.

c) **SMS**
Useful when the administrator is not present at the workstation, and does not have a network connection. He then sends the key by SMS to a phone connected to the server. The security module communicates with the phone via the serial port and can read or send SMS (to inform about the startup status). The SMS containing the key is immediately deleted after the key has been read.

2 **Using asymmetric encryption**
Asymmetric encryption [33][3] is used for the encryption key transmission. The encryption key is encrypted with the public key and decrypted with the corresponding private key on each side of the entities exchanging it. We therefore use asymmetric encryption for the protection of the symmetric encryption key [10][23].

i At startup, the security module gener-

10. Responsible of Cryptographic Operations

ates a pair of public/private key and publishes its public key; the private key remains in main memory and is not saved on disk.

ii The administrator uses this public key to encrypt the encryption key before submission.

iii The security module then uses its private key to decrypt the transmitted encrypted key, and thus obtains the configuration files encryption key, which also remains in main memory.

If the administrator is absent, has no connection to the network, or tool to perform its encryption, he can make use of an SMS to transmit the encryption key. The fear is that the key sent away may be intercepted (intruder 2): The technique of one-time password [11][33][13][35] is then used, as described below:

3 **Using one-time password**

At the starting of the CFS module, the administrator proceeds as follow:

i He encrypts and submits the current password and the next password through one of the interfaces at his disposal.

ii The CFS then verifies that the current password can perform the decryption, and that the new password has not yet been used.

iii If these two conditions are met, configuration files are decrypted with the current password and re-encrypted with the new one, that remains in main memory. The new key is then encrypted by an irreversible encryption and saved in the list of passwords already used.

Without password encryption, if the message is intercepted by an intruder, he will know the key used to make the last encryption, and if he has access to encrypted files, he could decrypt them. To avoid this, the reverse Vigenere square [12] is used to perform codification of the key before submitting it to the security module. This module has a copy of this square, and uses it to determine the real passwords transmitted. On the other hand this square is encrypted on disk, using the hashed identifier of hard drive as encryption key.

11. Valid password once to connect to a system

3. Results and Discussion

3.1. Results

As result of this work, it was proposed an encryption key management model for configurations files ensuring the protection and the availability of that key.

Concretely, the result consists in a security module (component called "Configuration Files Securer") placed between encrypted configuration files and applications. At the startup, this component notifies the administrator that it needs the encryption key (that is never stored on disk), and receives it in a secured and flexible way[13], and can then start deserving applications requesting access to their configuration files. Notice that this solution respects all the Kerckhoffs rules established for a cryptographic system [17].

Figure 6 describes the internal structure of that security module:

12. Variant of Vigenere square, developed by Dr. MOUKOUOP. It is a matrix of 31 columns and N rows. Each line corresponds to a character of the selected alphabet, and each column corresponds a day of the month. The character to be used on day j to encode the character i is placed at the intersection of row i and column j.

13. directly, through https or SMS

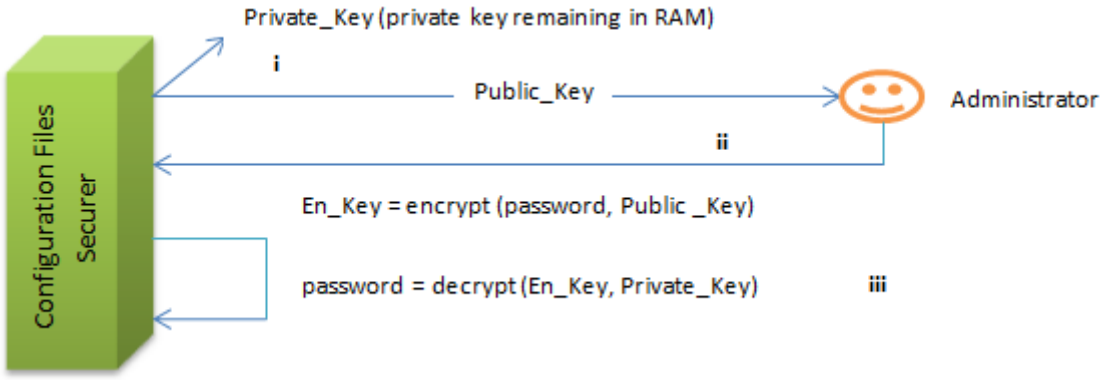

Fig. 4. Starting security module using asymmetric encryption

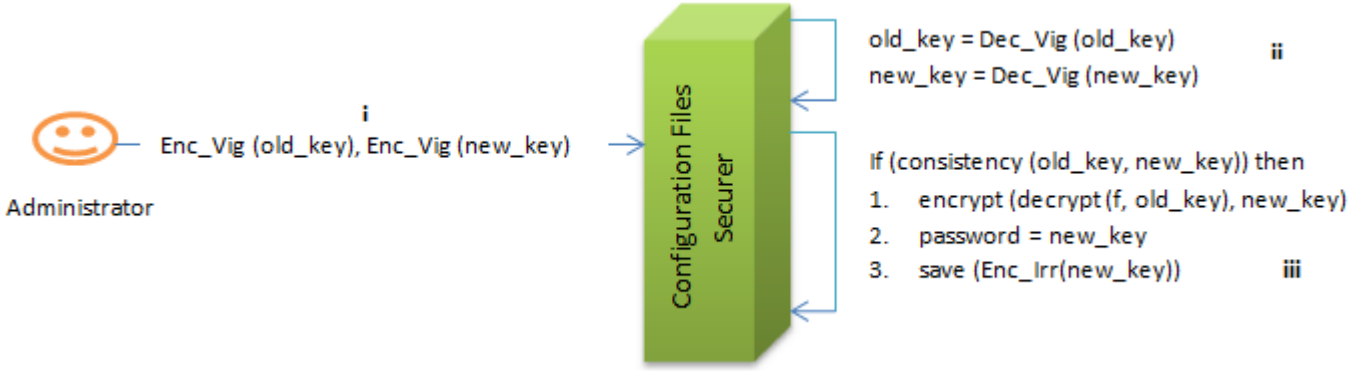

Fig. 5. Starting security module using one-time password

3.2. Discussion

3.2.1. Model validation by Kerckhoffs principles

We now check whether the proposed model meets the six rules established by Kerckhoffs for a cryptographic system [17].

1 **The system must be physically, if not mathematically, indecipherable**
To implement the solution, AES-256 algorithm[14] was chosen because, despite existing theoretical attacks[4], it is currently the safest symmetric encryption algorithm [14].

2 **It must not require secrecy, and it can conveniently fall in the hands of the enemy**
This rule, also called Kerckhoffs' principle means that: Any encryption method is known by the enemy and the security of the system only depends on the key management. The main purpose of the proposed model was precisely to protect the encryption key. It is clearly written nowhere and resides only in memory when used.

3 **The key must be provided and retained without written notes, and be changed or modified at the discretion of the relevant**
As said earlier, the encryption key is clearly written nowhere and resides only in main memory. In addition, the proposed model

14. Advanced Encryption Standard

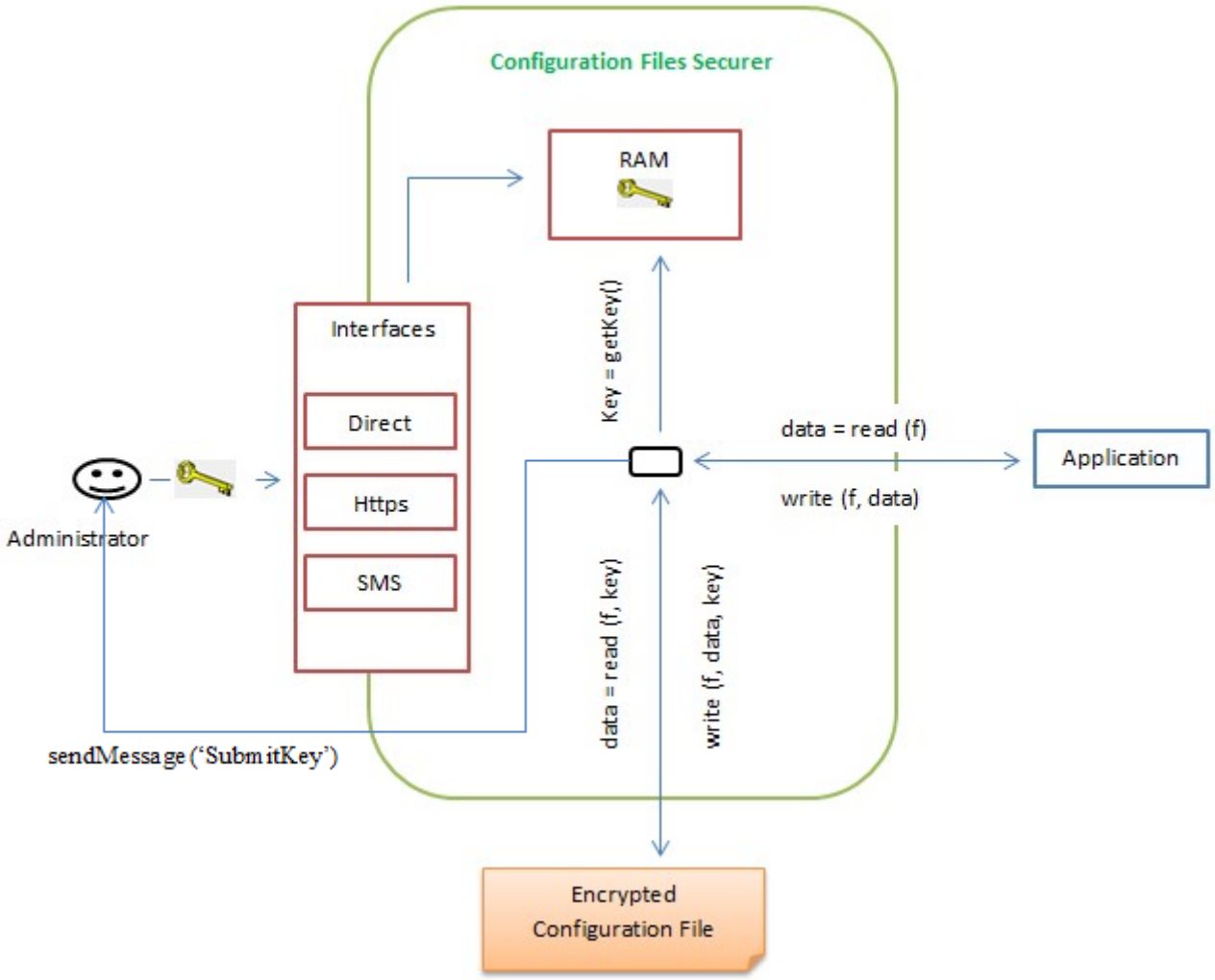

Fig. 6. Internal Schema of the security component

allows the keys to be modified at any time for security purposes.

4 **It must be applicable to telegraphic correspondence**
The model proposed here makes it possible to remotely transmit the encryption key at the startup of the security module. The key can be transmitted via https, SMS, or directly entered into the system, and this is done using a secured way.

5 **It must be portable, and its usage and operations not requiring the assistance of several people**
Only a system administrator is required to manipulate the encryption key.

6 **The system must be easy to use, requiring no spirit strength, nor knowledge of a long series of rules to be observed**
It is true that the usability is a rather relative concept, especially in the case of a security system. However, once taken in hand, the processes described in the proposed model do not prove to be complex.

3.2.2. Complexity of the system hacking problem

Assuming that the variant of the one-time password has been used, to perform an attack other than brute force unto the system to access configuration information, the attacker shall:

- Pick up the transmitted key. This is very difficult, because he does not previously know which of the three interfaces (SMS, https, or window) will be used by the administrator to submit the password, in order to monitor it.
- Have the decrypted reverse Vigenere square (so he shall have access to the application source code and to the server's hard disk identifier).
- Make an intrusion on the server to have currently encrypted configuration files.

However, it is practically impossible to simultaneously meet these three conditions in an application session: **the solution thus proves to be reliable.**

3.2.3. Comparison of the different encryption key management approaches

Table 1 below summarizes and compares the different approaches of encryption key management for configuration files' security.

4. Conclusion

Configuration files contain confidential information which should be preserved from all attacks, it is then necessary to encrypt them. To protect the encryption key, existing approaches consisted firstly in hiding the encryption key in the application source code, or somewhere on disk, its safety being compromised; or to exploit the protection mechanism of users accounts of the operating system, the problem is this case is that the application can operate only under a specific user account, which is quite restrictive for important systems. Moreover, the physical presence of this user is needed at the startup of the system. In this paper, a model was proposed to address these limitations. The encryption key is kept only in main memory, and may be submitted remotely to the configuration files security module when starting using a secure and flexible way. In perspective, this module could be enhanced to be developed and deployed as a real stand alone server, which can serve simultaneously a multiplicity of applications or distributed applications; it should therefore be necessary to define a protocol for client applications authentication.

Acknowledgments

The authors thank the government of Cameroon for premium sought they gave us in funding for our research. We thank the CETIC[15] for funding our research. We thank MEGASOFT SARL Company for allowing us to undertake this work.

15. Centre d'Excellence Africain en Technologies de l'Information et de la Communication

TABLE 1
Summary and comparison of different approaches of encryption key management for configuration files' security

APPROACH	DESCRIPTION	IMPLEMENT	SYSTEM ATTACK	SOLUTION
Key hidden in application source code	Applications have in their source code the key enabling them to encrypt and decrypt configuration files	EASY	EASY	EASY
Key saved on disk	The system reads the key from disk. The key may be read from a file, the hard drive identifier, a user environment variable content...	EASY	MEDIUM	EASY
Using one-time password + reverse Vigenere square	To start the security module, the administrator encodes the current and new passwords using the reverse Vigenere square. The system decodes the passwords, received through one of its three interfaces (window, SMS, https) and starts if the current password is the last to be used, and the new password ever. The files are then decrypted with the current password and re-encrypted with the new one, that remains in main memory.	COMPLEX	COMPLEX	COMPLEX
Using asymmetric encryption	The system generates a pair of public/private keys and publishes its public key, but keeps the private key in main memory. The administrator uses that public key to encrypt the password before submitting it. The security module decrypts the transmitted encrypted key with its private key, and gets the encryption key that also remains in main memory.	MEDIUM	COMPLEX	MEDIUM

References

[1] Symmetric-key management, 2014. http://books.mcgraw-hill.com/downloads/products/.

[2] Raffaele Addesso. Van dyke securecrt discloses password to local users, fevrier 2013.

[3] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. Recommendation for key management. *National Institute of Standards and Technology*, juillet 2012.

[4] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full aes. 2011.

[5] S. Burnett and S. Paine. Rsa security's official guide to cryptography. *RSA Press*, 2001.

[6] David Cane, David Hirschman, Philip Speare, and Lev Vaitzblit. Secure file archive through encryption key management. *United States Patent N. 5940507*, aout 1999.

[7] B. Savita Chavan and B. B. Meshram. Classification of web application vulnerabilities. *International Journal of Engineering Science and Innovative Technology (IJESIT)*, Volume 2(ISSN: 2319-5967):page 241, mars 2013.

[8] Richard Clayton and Mike Bond. Experience using a low-cost fpga design to crack des keys. *Cryptographic Hardware and Embedded Systems*, aout 2002.

[9] Club de la Securité de l'Information Français. *Gestion des secrets cryptographiques : Usages et bonnes pratiques*, mai 2012.

[10] Ian Curry. *An Introduction to Cryptography and Digital Signatures*. Entrust, mars 2001.

[11] Direction centrale de la sécurité des systèmes d'information France. *Gestion des clés cryptographiques*, mars 2006.

[12] Joy Marie Forsythe, Brian Gorenc, Heather Goudey, Abdul-Aziz Harir, Scott Lambert, John Park, Joe Sechman, Nidhi Shah, Jasiel Spelman, Jewel Timpe, and Jewel Timpe. Cyber risk report 2013. Technical Report 4AA5-0858ENW, HP, janvier 2014.

[13] Neil M. Haller. The s/key one time-password system. *CiteSeer*, fevrier 1994.

[14] Romaric Hinault. Le chiffrement aes cracker par des chercheurs francais belges et de microsoft, aout 2013. http://www.developpez.com/actu/36169/Le-chiffrement-AES-cracke-par-des-chercheurs-francais-belges-et-de-Microsoft-la-methode-reste-tres-complexe/.

[15] Cenzic Inc. Application vulnerability trends report. Technical report, Cenzic Inc., 2014.

[16] Martin Keith. *Cryptographic Key Management*. Information Security Summer School, juillet 2006.

[17] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, janvier 1883.

[18] Gary C. Kessler. *An Overview of Cryptography*, Novembre 2006.

[19] Ketki. Common vulnerabilities in configuration files, juillet 2013.

[20] A. Menezes, Van P. Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, juillet 1996.

[21] Microsoft. Chiffrement des informations de configuration à l'aide de la configuration protégée, decembre 2013. http://msdn.microsoft.com/fr-fr/library/53tyfkaw

[22] Nadia EL MRABET, J. C. Bajard, and S. Duquesne. *Cryptographie*. LIRMM-I3M-CNRS Université Montpellier I, octobre 2008.

[23] Christof Paar. *Symmetric Key Management:Key Derivation and Key Wrap*. Ruhr-Universitat Bochum, fevrier 2009.

[24] Radek. Where to store a key for encryption, mars 2012. http://security.stackexchange.com/questions/12332/where-to-store-a-key-for-encryption.

[25] Jamie Riden, Ryan McGeehan, Brian Engert, and Michael Mueter. Know your enemy: Web application threats, juin 2014.

[26] Eric Romang. Eric romang blog : Securecrt, juin 2014.

[27] C. Ross. Standards for encrypting passwords in configuration files, decembre 2013. http://security.stackexchange.com/questions/15040/standards-for-encrypting-passwordss-in-configuration-files.

[28] Karen Scarfone, Murugiah Souppaya, and Matt Sexton. *Guide to Storage Encryption Technologies for End User Devices*, novembre 2007.

[29] Cosine Security. Stealing passwords from mremote, juin 2011.

[30] Bryan Sullivan. *Top 10 Application Security Vulnerabilities in Web.config Files-Part One*. SPI Dynamics, mai 2007.

[31] Nicolas VERNON. *Réalisation d'un outil d'extraction de données et sécurisation du projet DNA*. Université de Nantes, Aout 2008.

[32] John Viega. Protecting sensitive data in memory, février 2001. http://www.ibm.com/developerworks/library/s-data.html?n-s-311.

[33] Nafi Kawser Wazed, Tonny Shekha Kar, Sayed Anisul Hoque, and Dr. M. M. A Hashem. A newer user authentication file encryption and distributed server based cloud computing security architecture. *International Journal of Advanced Computer Science and Applications*, 2012.

[34] Bing Wu, Jie Wu, and Mihaela Cardei. A survey of key management in mobile ad hoc networks. *Handbook of Research on Wireless Security*, 2001.

[35] Ilsou YOU. An one-time password authentication scheme for secure remote access in digital home networks. *Journal of Security Engineering*, novembre 2005.