Security in FileVault

Sakthiyuvaraja Sakthivelmurugan

CIS751 Report, Fall 2007

1 Introduction

Apple introduced FileVault into the Macintosh operating systems to provide a extra layer of security by encrypting the content of users home directory. Only a person with the users password will be able to decrypt and view the content of the users home directory. This report discuss about the implementation details and security in FileVault.

2 Overview of FileVault

FileVault employs Advanced Encryption Standards algorithm (AES also known as Rijndael, is a wildly used block cipher) to encrypt the user data.

When a user selects to secure his/her home directory with FileVault. A *Sparse Image* is created and the contents of the user directory is copied into the sparse image. The older contents are securely deleted so that they cannot be recovered using external software. A sparse image is a disk image which can be encrypted and occupies only the amount a space the data within takes. When a user logs in to the system the sparse image will be mounted in his home directory. The contents and folder structure will be same as it was before, so the user will not see any change in the way his home directory looks before and after encrypting with FileVault. When the user logs out the sparse image will be unmounted. Only a user which the password used for encryption could mount the sparse image.

The sparse image internally has a header and data region. Few fields in the header region is encrypted and whole of data region is encrypted. The users password is used to decrypt the content of the header which are encrypted. These encrypted fields have information about decrypting the data region. The user cans set a *Master Password* using which the password to a sparse image can be reset in the event of the user forgetting his password. If the user forgets both the master and user password the content of the sparse image will be unrecoverable.

3 Sparse Image

3.1 Header Region

The *header* of the sparse image contains details needed for decrypting the data region. The fields of the *header* are:

- Salt for PBKDF2
- Encrypted IV for 3DES-EDE

- Encrypted HMAC-SHA1 KEY
- Encrypted AES-KEY

The key used for encrypting the data region is encrypted (*Key Wrapping)* by again. There are certain advantages of using this mechanism, namely a stronger and difficult to crack encryption technique can be used to encrypt the key. One may argue that this stronger encryption technique can be used for encrypting the data; But it might prove to be detrimental if the time take for encrypting huge data is high.

The *Password Based Key Derivation Function (PBKDF2)* is used to create a key *passKey* from the user password. The key derivation function takes in three arguments user password *user_pass*, salt *s* and iteration count *1000* in this case.

The key wrapping unwrapping is done using the *passKey*, initialization vector IV and applying the 3DES-EDE on it which is shown in Table 2.

- wrapped key = 3DES-EDE(passKey, IV, key_to_be_encrypted)
- passKey = PBKDF2(salt,password, *iteration*)

3.2 Data Region

The *data region* is encrypted/decrypted using AES-128 block cipher. The IV is output of HMAC-SHA1 which takes the chunk number and Hmac-sha1 key read from the header and the key is the 128 bit AES Key stored in the header region.The content of the user home directory stored in the sparse image gets encrypted as 4K byte chunks in AES-128 CBC mode.

- Encrypted Data Chunk = AES-128_{cbc}(K, IV, chunk, ENCRYPT)
- IV = trunc_{128}(HMAC-SHA1(hmac-key || chunkno)
- K - Symmetric Key
- IV - Initialization Vector
- chunk - Data to be encrypted
- HMAC-SHA1 - Hashed Message Authentication Function
- hmac-key - Key used for HMAC-SHA1
- chunkno - Count of the chunk that is to be encrypted/decrypted.

4 Analysis

Most of the time the *user password* will be a simple dictionary based word which are easy to remember. Easy to remember password lacks entropy and are prone to brute force attack. Also the password do not have the sufficient length needed for cryptographic operations. So the password goes through a process called key strengthening which derives a key out of the password which are stronger and is difficult to be cracked.

PBKDF2 is a key strengthening function, it applies a pseudo random function, to the input password or passphrase along with a salt value and repeats the process many times (1000 is a

recommended minimum) to produce a derived key. Having a salt added to the password reduces the ability to use a preset dictionary to attack a password. The iterations increase the work that must be done on the attacker's side to build a brute force attack. If the salt is changed, the entire attack dictionary has to be rebuilt. This overhead makes pre built dictionary attack difficult on FileVault.

The *passKey* derived out of user password is not actually used for encryption of the sparse image data. The role of *passKey* is to wrap another key which is used for encryption of the data. This method is employed because, when user changes his password the encrypted data has to be re-encrypted using the new key. Re-encrypting will be an over head especially when the user data is huge. Now, only the header fields has to be re-encrypted with the new key and has to be replaced in place of the old header.

The unwrapping of the keys is done using 3DES-EDE. The unwrapped keys include the AES-Key and HMAC-SHA1-Key. The former is used for the decryption and encryption of the data and the later is used for the Hashed Message authentication code.

The AES-128 employs CBC *cipher block chaining,* this mode makes sure that no two identical plaintext block encrypt to the same cipher text. This is done by XORing the preceding cipher block with the plaintext.

There are problems with this mode of ciphers. If there is a bit error in the beginning of the block it gets propagated to the rest of the cipher. Also if n^{th} chunk has to be decrypted or encrypted it is dependent on $n\text{-}1^{th}$ chunks. An alternative approch called the counter mode is applied in FileVault. Which is a simple variation of cbc. The IV is not calculated from the preceeding cipher instead the *chunk no* is used for it. Since, IV should not be known by the attacker, an HMAC is created using the chunk no which is difficult to reproduce without knowing the HMAC-Key. HMAC are collision resistant so there will be ε chance that the attacker can find the IV even if the chunk number is know to the attacker.

We have just seen that no two IV can be identical. Consider d_1 and d_2 are two data chunks wich are identical with c_1 and c_2 as chunk numbers. *AES(d_1, IV_1, AES-Key)* $\neq$ *AES(d_2, IV_2, AES-Key).* Where IV_1 = *Hmac-sha1(key, c_1) and* IV_2 = *Hmac-sha1(key, c_2).* This shows that no two cipher in the data region will be identical.

4.1 Possible Attacks

Even after using complex techniques to keep the data safe, there are some attacks possible on FileVault. Watermarking is one of them.

The CBC mode of operation leads to watermarking attacks. Where the attacker will be able to predict the presence of some information with out knowing the key.

4.2 Known Issues

On power failure and on some rare occasion if the sparse image gets corrupted the user may end up losing all data of his home directory since sparse image is a single file inside which all the data of the user home directory is stored.

4.3 Versions

The *Sparse Image* has two versions: version^1 where the header information are at the end of the disk image and version^2 where the header information are stored in beginning of the file. The analysis was done for version^2 which is the current version used in Mac OS X 10.4.

5 Conclusion

FileVault can't possibly be extended with the current design to incorporate a full disk encryption as many people would want to. But its possible to do; to have a full disk encryption the boot process has to be modified to understand the decryption technique and more enhancements so that the encrypted disk image can be mounted from which the OS should start booting. The speed of the system may go down considerably considering the number of encryption and decryption operation that has to occur and a single disk image will be a point point of failure for corruption. Recovering corrupted image will be hurdle that has to be fixed.FileVault can't possibly be extended with the current design to incorporate a full disk encryption.

FileVault was meant to encrypt home directories for which it is perfectly designed and have the security features.

6 Reference

- http://crypto.nsa.org/vilefault/23C3-VileFault.pdf
- http://www.ietf.org/rfc/rfc3826.txt
- http://www.ietf.org/rfc/rfc2898.txt
- New Methods in Hard Disk Encryption, Clemens Fruhwirth

7 Appendix

An overview of the experiments done as a part of the report.

- Apple Open Source Libraries analysis to understand the implementation of FileVault
  - libsecurity_filevault-28631
  - libsecurity_cdsa_utilities-32432
  - libsecurity_apple_csp-32567
- Creation of sparse image using `hdiutil`. The debug mode of hdiutil gave some internals fo the sparse image, but not much could be understood as the disk framework is private.
- A code sinippet which simulates the implementation of the header region.