January 15, 2014

EncFS Security Audit

This report is the result of a paid 10-hour security audit of EncFS. It has been posted to the EncFS mailing list, so check there for follow-up. I feel that full disclosure is the best approach for disclosing these vulnerabilities, since some of the issues have already been disclosed but haven't been fixed, and by disclosing them, users can immediately re-evaluate their use of EncFS.

Thanks to Igor Sviridov for funding this audit.

Note: This report was updated on February 5, 2014, thanks to feedback from Robert Freudenreich (founder and CTO of boxcryptor), to correct a technical inaccuracy about how initialization vectors are generated and to clarify the conclusion of the report. You can see the old version of the report here.

```
-----------------------------------------------------------------------
                          EncFS Security Audit
                             Taylor Hornby
                            January 14, 2014
                       (Updated: February 5, 2014)
-----------------------------------------------------------------------

1. Introduction

  This document describes the results of a 10-hour security audit of
  EncFS 1.7.4. The audit was performed on January 13th and 14th of 2014.

1.1. What is EncFS?

  EncFS is a user-space encrypted file system. Unlike disk encryption
  software like TrueCrypt, EncFS's ciphertext directory structure
  mirrors the plaintext's directory structure. This introduces unique
  challenges, such as guaranteeing unique IVs for file name and content
  encryption, while maintaining performance.

1.2. Audit Results Summary

  This audit finds that EncFS is not up to speed with modern
  cryptography practices. Several previously known vulnerabilities have
  been reported [1, 2], which have not been completely fixed. New issues
  were also discovered during the audit.

  The next section presents a list of the issues that were discovered.
  Each issue is given a severity rating from 1 to 10. Due to lack of
  time, most issues have not been confirmed with a proof-of-concept.

2. Issues

2.1. Same Key Used for Encryption and Authentication

  Exploitability: Low
  Security Impact: Low

  EncFS uses the same key for encrypting data and computing MACs. This
  is generally considered to be bad practice.
```

EncFS should use separate keys for encrypting data and computing MACs.

2.2. Stream Cipher Used to Encrypt Last File Block

Exploitability: Unknown
Security Impact: High

As reported in [1], EncFS uses a stream cipher mode to encrypt the last file block. The change log says that the ability to add random bytes to a block was added as a workaround for this issue. However, it does not solve the problem, and is not enabled by default.

EncFS needs to use a block mode to encrypt the last block.

EncFS's stream encryption is unorthodox:

1. Run "Shuffle Bytes" on the plaintext.
   N[J+1] = Xor-Sum(i = 0 TO J) { P[i] }
   (N = "shuffled" plaintext value, P = plaintext)
2. Encrypt with (setIVec(IV), key) using CFB mode.
3. Run "Flip Bytes" on the ciphertext.
   This reverses bytes in 64-byte chunks.
4. Run "Shuffle Bytes" on the ciphertext.
5. Encrypt with (setIVec(IV + 1), key) using CFB mode.

Where setIVec(IV) = HMAC(globalIV || (IV), key), and,
- 'globalIV' is an IV shared across the entire filesystem.
- 'key' is the encryption key.

This should be removed and replaced with something more standard. As far as I can see, this provides no useful security benefit, however, it is relied upon to prevent the attacks in [1]. This is security by obscurity.

2.3. Generating Block IV by XORing Block Number

Exploitability: Low
Security Impact: Medium

Given the File IV (an IV unique to a file), EncFS generates per-block IVs by XORing the File IV with the Block Number, then passing the result to setIVec(), which is described in Section 2.2. This is not a good solution, as it leads to IV re-use when combined with the last-block stream cipher issue in Section 2.2:

The stream algorithm (see previous section) adds 1 to the IV, which could *undo* the XOR with the block number, causing the IV to be re-used. Suppose the file consists of one and a half blocks, and that the File IV's least significant bit (LSB) is 1. The first block will be encrypted with the File IV (block number = 0). The second (partial) block will be encrypted with File IV XOR 1 (since block number = 1), making the LSB 0, using the stream algorithm. The stream algorithm adds 1 to the IV, bringing the LSB back to 1, and hence the same IV is

used twice. The IVs are reused with different encryption modes (CBC and CFB), but CFB mode starts out similar to CBC mode, so this is worrisome.

EncFS should use a mode like XTS for random-access block encryption.

Correction 12/05/2014: XTS mode is probably not the ideal option, see Thomas Ptacek's blog post for good reasons why:

http://sockpuppet.org/blog/2014/04/30/you-dont-want-xts/

2.4. File Holes are Not Authenticated

Exploitability: High
Security Impact: Low

File holes allow large files to contain "holes" of all zero bytes, which are not saved to disk. EncFS supports these, but it determines if a file block is part of a file hole by checking if it is all zeroes. If an entire block is zeroes, it passes the zeroes on without decrypting it or verifying a MAC.

This allows an attacker to insert zero blocks inside a file (or append zero blocks to the end of the file), without being detected when MAC headers are enabled.

2.5. MACs Not Compared in Constant Time

Exploitability: Medium
Security Impact: Medium

MACs are not compared in constant time (MACFileIO.cpp, Line 209). This allows an attacker with write access to the ciphertext to use a timing attack to compute the MAC of arbitrary values.

A constant-time string comparison should be used.

2.6. 64-bit MACs

Exploitability: Low
Security Impact: Medium

EncFS uses 64-bit MACs. This is not long enough, as they can be forged in 2^64 time, which is feasible today.

EncFS should use (at least) 128-bit MACs.

2.7. Editing Configuration File Disables MACs

Exploitability: High
Security Impact: Medium

The purpose of MAC headers is to prevent an attacker with read/write access to the ciphertext from being able to make changes without being

detected. Unfortunately, this feature provides little security, since it is controlled by an option in the .encfs6.xml configuration file (part of the ciphertext), so the attacker can just disable it by setting "blockMACBytes" to 0 and adding 8 to "blockMACRandBytes" (so that the MAC is not interpreted as data).

EncFS needs to re-evaluate the purpose of MAC headers and come up with something more robust. As a workaround, EncFS could add a command line option --require-macs that will trigger an error if the configuration file does not have MAC headers enabled.

3. Future Work

There were a few potential problems that I didn't have time to evaluate. This section lists the most important ones. These will be prioritized in future audits.

3.1. Information Leakage Between Decryption and MAC Check

EncFS uses Mac-then-Encrypt. Therefore it is possible for any processing done on the decrypted plaintext before the MAC is checked to leak information about it, in a style similar to a padding oracle vulnerability. EncFS doesn't use padding, but the MAC code does iteratively check if the entire block is zero, so the number of leading zero bytes in the plaintext is leaked by the execution time.

3.2. Chosen Ciphertext Attacks

Since the same key is used to encrypt all files, it may be possible for an attacker with read/write access to the ciphertext and partial read access to the plaintext (e.g. to one directory when --public is used) to perform a chosen ciphertext attack and decrypt ciphertexts for which they have no plaintext access.

EncFS should consider using XTS mode.

Correction 12/05/2014: XTS mode is probably not the ideal option, see Thomas Ptacek's blog post for good reasons why:

    http://sockpuppet.org/blog/2014/04/30/you-dont-want-xts/

3.3. Possible Out of Bounds Write in StreamNameIO and BlockNameIO

There is a possible buffer overflow in the encodeName method of StreamNameIO and BlockNameIO. The methods write to the 'encodedName' argument without checking its length. This may allow an attacker with control over file names to crash EncFS or execute arbitrary code.

3.4. 64-bit Initialization Vectors

Initialization vectors are only 64 bits, even when using AES instead of Blowfish. This may lead to vulnerabilities when encrypting large (or lots of) files.

4. Conclusion

In conclusion, while EncFS is a useful tool, it ignores many standard best-practices in cryptography. This is most likely due to it's old age (originally developed before 2005), however, it is still being used today, and needs to be updated.

The EncFS author says that a 2.0 version is being developed [3]. This would be a good time to fix the old problems.

EncFS is probably safe as long as the adversary only gets one copy of the ciphertext and nothing more. EncFS is not safe if the adversary has the opportunity to see two or more snapshots of the ciphertext at different times. EncFS attempts to protect files from malicious modification, but there are serious problems with this feature.

5. References

[1] http://archives.neohapsis.com/archives/fulldisclosure/2010-08/0316.html

[2] http://code.google.com/p/encfs/issues/detail?id=128

[3] https://code.google.com/p/encfs/issues/detail?id=186