Vulnerability in encrypted loop device for Linux

Jerome Etienne jme@off.net

Abstract

This text describes a security hole i found in encrypted loop device for Linux. An attacker is able to modify the content of the encrypted device without being detected (see section 2). This text proposes to fix the hole by authenticating the device (see section 3).

1 Threat model

Encrypting a disk device aims to protect against a off-line attacker who would be able to access the disk between 2 legitimate mounts.

It isn't against an attacker who has access to the running computer when the encrypted device is mounted as either (i) the attacker is root and it can access the encrypted device anyway or (ii) he is an unprivileged user and can be stopped with Unix's right management (i.e. user/group).

2 Attack description

The vulnerability of encrypted loop device is due to its lack of authentication. The aim of encryption is to make the data unreadable for anybody who doesn't know the key. It doesn't prevent an attacker from modifying the data. People assume that an attacker won't do it because the attacker wouldn't be able to choose the resulting clear text. But this section shows that the attacker can choose the resulting clear text to some extends and that modifying the cypher text data may be interesting even if the attacker ignores the result.

This attack is only applicable to device storing data which are reused across mounts: most file-system (e.g. ext2, reiserfs, ext3) but not swap. In some systems, encrypted device are stored in the same location than the encrypted disk containing the operating system. For those systems the attacker who can access the encrypted device, can easily modify the OS to gain access (e.g. kernel) independtly of the encrypted device.

2.1 To insert random data

If the attacker modifies the cipher text without choosing the resulting clear text, it will likely produce random data. The legitimate user won't detect the modification and will use them as if they were valid. As they likely appears random, it will result of a Denial of Service (aka DoS).

2.2 To insert chosen data

The encryption mode used by encrypted loop device is CBC[oST81, sec 5.3]. CBC allows cut/past attacks i.e. the attacker can cut encrypted data from one part of the device and paste them in another location. As both data sections have been encrypted by the same key, the clear text won't be completely random data.

This lack of authentication isn't a CBC flaw. Authentication isn't considered a aim of the encryption mode, so most modes (e.g. ECB, CFB, OFB) doesn't authenticate the data. To use another mode would be flawed in the same way except if they explicitly protect against forgery. Recently some modes including authentication popped up to speed up the encryption / authentication couple but as far as i know they are all patented.

In very short, encrypting with CBC is Cn=Enc(Cn-1 xor Pn) where Enc(x) is encrypting x, Pn is the nth block of plain text and Cn the nth block of cipher text. For the first block, Cn-1 is an Initial vector (aka IV) which may be public and must be unique for a given key. The decryption is Pn = Dec(Cn) xor Cn-1. See [oST81, sec 5.3] for a longer description of CBC.

If the attacker copies s blocks from the location m to n (aka [Cn,...,Cn+s-1] == [Cm,...,Cm+s-1]), Pn+1 up to Pn+s-1 will the same as Pm+1 to Pm+s-1 and Pn will likely appears random. Cn (i.e. Cm) will be decrypted as Pn = Dec(Cm) xor Cn-1 but Cm-1 and Cn-1 are different so Pn will likely appears random. Nevertheless Pn+1 = Dec(Cn+1) xor Cn = Dec(Cm+1) xor Cm = Pm+1, so Pn+1=Pm+1. So if the attacker has an idea of the content of a group of blocks in the device, he can copy them to the Nth block, thus it can choose the content of it without being detected.

As an file-system isn't designed to appears ran-

dom, its content may be predictable to some extents (e.g. common directories and files, inode, superblock). The attacker may use such informations to guess the contents and do a knowledgeable cut/past. For example, an attacker knowing the location of a password file may replace a password by another one which is already known.

3 Proposed fixes

We propose 2 types of fixes: one which authenticate at mount time (see section 3.1) and the other which authenticates at the cluster level (see section 3.2). The choice between the two (see section 3.4) is a user matter as it mostly depends on the access pattern on the encrypted device.

In the proposed fixes, the authentication is a MAC computed over the encrypted device. The MAC is HMAC[KBC97] combined with a configured hash function, preferably a well studied one such as SHA1[oST95] or MD5[Riv92]. The MAC secret key is derived from the pass-phrase via PKCS-5 key derivations ([Kal00, sec 5.1]).

3.1 Authenticating at mount time

As we need to authenticate the device across mounts and not while it is mounted (see section 1), it is sufficient to authenticate the whole device during mount operations. It slows down mount operations but they are rather infrequent so we consider the trade-off delay/security acceptable. The MAC is verified during mount operations and generated during unmount operations. It isn't supposed to be valid while the device is mounted.

The MAC generation is done when unmounting the device. The MAC is computed over all the sectors of the device and the result is appended in the device file after all the sectors.

The MAC verification is done when mounting the device. The MAC is computed over all the sectors of the device. If the result is equal to the MAC appended to the block device, the verification succeed, else it failed. The verification may fail (i) if an attacker attempted to modify the device during 2 legitimates mounts or (ii) if the device hasn't been cleanly unmounted (e.g. computer crash). It is impossible to automatically distinguish both cases with certainty. So if the verification fails, the user is notified and the mount operation may be stopped depending on configuration.

3.2 Authenticating at cluster level

To authenticate the whole device at mount time, may be considered prohibitive by some users, so this section describe an alternative which authenticate the device at the cluster level. A cluster is a group of one or more sectors, the exact number depends on configuration. In this case, the MAC is verified each time a cluster is read from the disk and generated at each write.

If the device isn't cleanly unmounted, the authentication of one or more cluster may fail (e.g. the super block). This case will be detected at mount time. But if an attacker forges data in the device, it will be detected only when the user read the modified data. The kernel will read the forged cluster and the authentication will fail. It may report it with a printk with a rate limitor, it isn't clean but i don't see any better way.

3.3 MAC location

Currently the encrypted loop file-system is stored in a regular file of a hosting file-system. Its size is a multiple of a sector size (i.e. 512 byte). The MAC could be stored in a separate file or included in the regular file. To store the MAC in a separate file, generates problems while managing the loop device file (e.g. copy, backup). The administrator must not forget to copy the MAC file when he copies the device file, else the copied device won't be usable anymore. To store the MAC in the same file as the clusters doesn't has this disadvantage.

3.4 Comparison

To authenticate at the cluster level will increase the access time of each cluster but won't affect mount operation. The exact increase depends on the MAC and encryption algorithms. As a rule of thumb, MAC algorithms are typically 3 times faster than encryption ones so the time dedicated to cryptography for each block will increase by around 30%. To authenticate at mount time will largely slow down the mount operations but won't affect every access once mounted.

The authentication at mount time will detect forgery at mount time, whereas the alternative detects it only when the forged cluster is read, possibly a long time after the modification. Users may consider that it is easier to diagnose who forged it if they have a better idea of when the attack occurred.

To authenticate the whole device at mount time requires a single MAC per device, so the space overhead (typically 16 byte) is negligible compared to the device's size. To authenticate at the cluster level requires a MAC per cluster, it is significantly more but

some people may consider it still negligible, especially with cheap disks.

The choice between the two mostly depends on the access pattern on the encrypted device. If the device is used for interactive purpose, the increased access latency may be unsuitable. If the access latency is important or if every block is frequently modified, to authenticate only once at mount time may be more interesting. If the user can't stand long mount operations, to authenticate at cluster level will be more suitable. As only the final user knows the type access made on his encrypted device, he should be the one able to choose between the two.

4 Acknowledgments

Thanks to Andy Kleen and Phil Swan for their useful comments.

5 Conclusion

This text described an vulnerability in encrypted loop device which allows an attacker to modify the encrypted device without being detected (see section 2). We propose a fix which authenticate the whole device during mount operation (see section 3.1). This fix slows down mount operations but we consider the trade-off longer delay vs additional security very reasonable as mount operations are rather infrequent. We propose another fix which authenticate at cluster level for people who can't stand long mount operation. The choice between the two is a final user matter.

The authentication may be optionally disabled thus if an user considers the trade-off delay/security not in favor of security, he may choose to be vulnerable to this attack and disable it. Nevertheless the author thinks encrypted loop device must be secure by default.

References

[Kal00] B. Kaliski. Pkcs 5: Password-based cryptography specification version 2.0. *Request For Comment (Informational) RFC2898*, September 2000.

[KBC97] H. Krawczyk, M. Bellare, and R. Canetti. Hmac: Keyed-hashing for message authentication. *Request For Comment (Informational) RFC2104*, February 1997.

[oST81] National Institute of Standards and Technology. implementing and using the nbs data encryption standard. *Federal information processing standards fips74*, April 1981.

[oST95] National Institute of Standards and Technology. Secure hash standard. *Federal information processing standards fips180-1*, April 1995.

[Riv92] R. Rivest. The md5 message-digest algorithm. *Request For Comment (Informational) RFC1321*, April 1992.