NAME

gbde - Geom Based Disk Encryption

SYNOPSIS

options GEOM_BDE

DESCRIPTION

NOTICE: Please be aware that this code has not yet received much review and analysis by qualified cryptographers and therefore should be considered a slightly suspect experimental facility.

We cannot at this point guarantee that the on-disk format will not change in response to reviews or bug-fixes, so potential users are advised to be prepared that dump(8)/restore(8) based migrations may be called for in the future.

The objective of this facility is to provide a high degree of denial of access to the contents of a "cold" storage device.

Be aware that if the computer is compromised while up and running *and* the storage device is actively attached and opened with a valid pass-phrase, this facility offers no protection or denial of access to the contents of the storage device.

If, on the other hand, the device is "cold", it should present a formidable challenge for an attacker to gain access to the contents in the absence of a valid pass-phrase.

Four cryptographic barriers must be passed to gain access to the data, and only a valid pass-phrase will yield this access.

When the pass-phrase is entered, it is hashed with SHA2 into a 512 bit "key-material". This is a way of producing cryptographic usable keys from a typically all-ASCII pass-phrase of an unpredictable user-selected length.

First barrier: the location of the "lock-sector".

During initialization, up to four independent but mutually aware "lock" sectors are written to the device in randomly chosen locations. These lock-sectors contain the 2048 random bit master-key and a number of parameters of the layout geometry (more on this later). Since the entire device will contain isotropic data, there is no short-cut to rapidly determine which sequence of bytes contain a lock-sector.

To locate a lock-sector, a small piece of data called the "metadata" and the key-material must be available. The key-material decrypts the metadata, which contains the byte offset on the device where the corresponding lock-sector is located. If the metadata is lost or unavailable but the key-material is at hand, it would be feasible to do a brute force scan where each byte offset of the device is checked to see if it contains the lock-sector data.

Second barrier: decryption of the master-key using key-material.

The lock-sector contains an encrypted copy of an architecture neutral byte-sequence which encodes the fields of the lock-structure. The order in which these fields are encoded is determined from the key-material. The encoded byte stream is encrypted with 256bit AES in CBC mode.

Third barrier: decryption of the sector key.

For each sector, an MD5 hash over a "salt" from the lock-sector and the sector number is used to "cherry-pick" a subset of the master key, which hashed together with the sector offset through MD5 produces the "kkey", the key which encrypts the sector key.

Fourth barrier: decryption of the sector data.

The actual payload of the sector is encrypted with 128 bit AES in CBC mode using a single-use random bits key.

Examining the reverse path

Assuming an attacker knows an amount of plaintext and has managed to locate the corresponding encrypted sectors on the device, gaining access to the plaintext context of other sectors is a daunting task:

First he will have to derive from the encrypted sector and the known plain text the sector key(s) used. At the time of writing, it has been speculated that it could maybe be possible to break open AES in only 2^80 operations; even so, that is still a very impossible task.

Armed with one or more sector keys, our patient attacker will then go through essentially the same exercise, using the sector key and the encrypted sector key to find the key used to encrypt the sector key.

Armed with one or more of these "kkeys", our attacker has to run them backwards through MD5. Even though he knows that the input to MD5 was 24 bytes and has the value of 8 of these bytes from the sector number, he is still faced with 2^128 equally likely possibilities.

Having successfully done that, our attacker has successfully discovered up to 16 bytes of the master-key, but is still unaware which 16 bytes, and in which other sectors any of these known bytes contribute to the kkey.

To unravel the last bit, the attacker has to guess the 16 byte random-bits salt stored in the lock-sector to recover the indexes into the masterkey.

Any attacker with access to the necessary machine power to even attempt this attack will be better off attempting to brute-force the pass-phrase.

Positive denial facilities

Considering the infeasibility of the above attack, gaining access to the pass-phrase will be of paramount importance for an attacker, and a number of scenarios can be imagined where undue pressure will be applied to an individual to divulge the pass-phrase.

A "Blackening" feature provides a way for the user, given a moment of opportunity, to destroy the master-key in such a way that the pass-phrase will be acknowledged as good but access to the data will still be denied.

A practical analogy

For persons who think cryptography is only slightly more interesting than watching silicon sublimate the author humbly offers this analogy to the keying scheme for a protected device:

Imagine an installation with a vault with walls of several hundred meters thick solid steel. This vault can only be feasibly accessed using the single key, which has a complexity comparable to a number with 600 digits.

This key exists in four copies, each of which is stored in one of four small safes, each of which can be opened with unique key which has a complexity comparable to an 80 digit number.

In addition to the masterkey, each of the four safes also contains the exact locations of all four key-safes which are located in randomly chosen places on the outside surface of the vault where they are practically impossible to detect when they are closed.

Finally, each safe contains four switches which are wired to a bar of dynamite inside each of the four safes.

In addition to this, a keyholder after opening his key-safe is also able to install a copy of the master-key and re-key any of key-safes (including his own).

In normal use, the user will open the safe for which he has the key, take out the master-key and access the vault. When done, he will lock up the master-key in the safe again.

If a keyholder-X for some reason distrusts keyholder-Y, she has the option of opening her own safe, flipping one of the switches and detonating the bar of dynamite in safe-Y. This will obliterate the master-key in that safe and thereby deny keyholder-Y access to the vault.

Should the facility come under attack, any of the keyholders can detonate all four bars of dynamite and thereby make sure that access to the vault is denied to everybody, keyholders and attackers alike. Should the facility fall to the enemy, and a keyholder be forced to apply his personal key, he can do so in confidence that the contents of his safe will not yield access to the vault, and the enemy will hopefully realize that applying further pressure on the personnel will not give access to the vault.

The final point to make here is that it is perfectly possible to make a detached copy of any one of these keys, including the master key, and deposit or hide it as one sees fit.

Steganography support

When the device is initialized, it is possible to restrict the encrypted data to a single contiguous area of the device. If configured with care, this area could masquerade as some sort of valid data or as random trash left behind by the systems operation.

This can be used to offer a plausible deniability of existence, where it will be impossible to prove that this specific area of the device is in fact used to store encrypted data and not just random junk.

The main obstacle in this is that the output from any encryption algorithm worth its salt is so totally random looking that it stands out like a sore thumb amongst practically any other sort of data which contains at least some kind of structure or identifying byte sequences.

Certain file formats like ELF contain multiple distinct sections, and it would be possible to locate things just right in such a way that a device contains a partition with a file system with a large executable, ("a backup copy of my kernel") where a non-loaded ELF section is laid out consecutively on the device and thereby could be used to contain a **gbde** encrypted device.

Apart from the ability to instruct **gbde** which those sectors are, no support is provided for creating such a setup.

Deployment suggestions

For personal use, it may be wise to make a backup copy of the masterkey or use one of the four keys as a backup. Fitting protection of this key is up to yourself, your local circumstances and your imagination.

For company or institutional use, it is strongly advised to make a copy of the master-key and put it under whatever protection you have at your means. If you fail to do this, a disgruntled employee can deny you access to the data "by accident". (The employee can still intentionally deny access by applying another encryption scheme to the data, but that problem has no technical solution.)

Cryptographic strength

This section lists the specific components which contribute to the cryptographic strength of **gbde**.

The payload is encrypted with AES in CBC mode using a 128 bit random single-use key ("the skey"). AES is well documented.

No IV is used in the encryption of the sectors, the assumption being that since the key is random bits and single-use, an IV adds nothing to the security of AES.

The random key is produced with arc4rand(9) which is believed to do a respectable job at producing unpredictable bytes.

The skey is stored on the device in a location which can be derived from the location of the encrypted payload data. The stored copy is encrypted with AES in CBC mode using a 128 bit key ("the kkey") derived from a subset of the master key chosen by the output of an MD5 hash over a 16 byte random bit static salt and the sector offset. Up to 6.25% of the masterkey (16 bytes out of 2048 bits) will be selected and hashed through MD5 with the sector offset to generate the kkey.

Up to four copies of the master-key and associated geometry information is stored on the device in static randomly chosen sectors. The exact location inside the sector is randomly chosen. The order in which the fields are encoded depends on the key-material. The encoded byte-stream is encrypted with AES in CBC mode using 256 bit key-material.

The key-material is derived from the user-entered pass-phrase using 512 bit SHA2.

No chain is stronger than its weakest link, which usually is poor pass-phrases.

SEE ALSO

gbde(8)

HISTORY

This software was developed for the FreeBSD Project by Poul-Henning Kamp and NAI Labs, the Security Research Division of Network Associates, Inc. under DARPA/SPAWAR contract N66001-01-C-8035 ("CBOSS"), as part of the DARPA CHATS research program.

AUTHORS

Poul-Henning Kamp <*phk@FreeBSD.org*>