Centre for Research on
Cryptography and Security

Storage encryption
... what about data integrity?

Milan Brož
mbroz@redhat.com

DevConf, Brno
January 28, 2017

Agenda

- Data integrity – what it is?
- Encryption / confidentiality / no integrity protection?
- Methods / examples

- Integrity in the Full Disk Encryption context
- Demo (dm-crypt / dm-integrity)

Problem: data corruption

- **Silent data corruption**

  unintentional, random change

  (flaky cables, memory errors, data misplacement)

- **Unauthorized change of data**

  intentional modification (by an attacker)

Both above can propagate further if undetected.

Data integrity (~data-at-rest storage)

- Provides data consistency

- Corruption detection during the entire data life-cycle

Silent data corruption

=> (meta)data checksums

Unauthorized change of data

=> message authentication (MAC), auth. tag

if encryption in place => authenticated encryption

Integrity protection

- **Integrity corruption detection**

  Requires additional storage space

  to store checksum or authentication tag

- **Error correction (detect & repair data)**

  Even more additional space (redundancy)

- **So... why we need it?**

  Isn’t it in hardware already?

  Can we detect unauthorized modification?

  Can it prevent also "reply attacks"?

Non-authenticated integrity protection

- **Parity, checksums**

  example: CRC32

  no problem if collision exists

  (collision = two messages with the same checksum)

- **Cryptographic hash algorithms**

  example: sha256 (sha256sum command)

  finding collision should infeasible in reasonable time

- **Anyone can calculate and verify**

Authenticated integrity protection

- **Keyed cryptographic hash**

  example: HMAC

  only authenticated user can calculate and verify

- **Authenticated encryption (AE) modes**

  example: AES-GCM

  note: CAESAR crypto competition

  provides both confidentiality and integrity

  - **AE with additional data (AEAD)**

    authenticates also additional metadata

Examples for storage integrity (OSS)

- Block device: dm-verity (Android secure boot)
- Filesystems – btrfs, ZFS, bcachefs, ...

  ...

- (Application level – databases, archives, ...)

Our case: LUKS / Full Disk Encryption

- **Length-preserving encryption**

  Plaintext and ciphertext of the same size (sector)

- **Provides confidentiality**

  Data available only to authorized users

- **Provides no data integrity protection**

  Except "poor man authentication" –

  garbage after decryption can be detected

(note) Change propagation in FDE

- **Small (bit) change in ciphertext**

  => pseudo-random change in plaintext

  typically 16 bytes (cipher block) up to whole sector

- **Small (bit) change in plaintext**

  => should propagate to the whole sector (best case)

  in reality, only part of sector changes

- **Error / data corruption is the same case!**

Authenticated encryption with LUKS

- Inspired by "an academic" idea – is it feasible?
- Need metadata per-sector for authentication tag.
- We already have key management – LUKS
- **dm-integrity module**

  provides metadata per-sector
- **dm-crypt module**

  adds authentication encryption (AEAD)

  stores authentication tag and nonce or IV

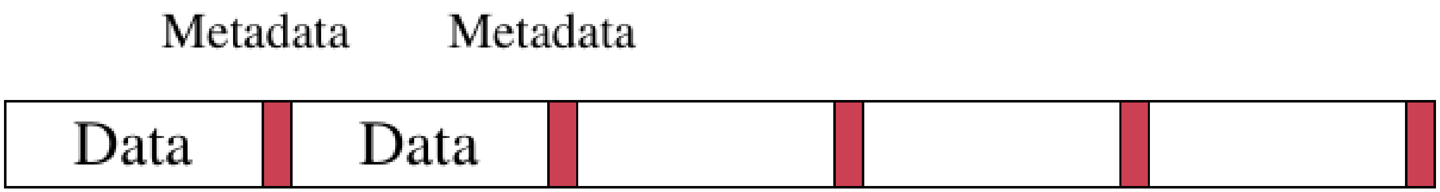
  to dm-integrity additional metadata

dm-integrity

- Separate and new device-mapper target
- Emulates per-sector metadata
  (like DIF in hardware just size is configurable)
- Uses interleaved metadata sectors
- Provides atomic updates of sector+metadata
- Two modes
  1) Provides metadata to another target (dm-crypt)
  2) Standalone mode (built-in checksum)

dm-integrity

- Layout on disk – principle



- Additional metadata space

Metadata [bytes]	**Space** [%] 512B sector	**Space** [%] 4096B sector
16 (128 bits)	3.03	0.39
28 (224 bits)	5.26	0.68
32 (256 bits)	5.88	0.78
48 (284 bits)	9.09	1.16
64 (512 bits)	11.11	1.54
80 (640 bits)	14.29	1.92

dm-crypt extension

- Adds AEAD algorithms for FDE

- Built on top of kernel cryptographic API
  algorithm agnostic – we can use whatever is available!

  - Combination of existing modes (XTS) and HMAC

  - or specifically designed AEAD
    - GCM, GCM-SIV (draft) [just for demo!]
    - Chacha20-poly1305

dm-crypt AEAD extension

- AEAD sector request

```
|----- AAD -------|------ DATA -------|-- AUTH TAG --|
| (authenticated) | (auth+encryption) |              |
| sector_LE |  IV |  sector in/out    |  tag in/out  |
```

<cryptography_corner>

The devil is in the ~~detail~~ nonce (Initialization Vector)!

- Sector inicialization vector / nonce must not be reused.
  (fatal attacks to some modes like GCM)

- Random IV – IV is regenerated after every write
  even with same data => IND-CPA (encryption oracle)
  bonus: no more change localization with XTS-random

</cryptography_corner>

Demonstration

- **LUKS encrypted Fedora**

  - with authenticated encryption

- **cryptsetup extensions**

  - dm-integrity and dm-crypt stack

Conclusion

- We implemented authenticated encryption in LUKS.
- It can be quite easily adopted to existing systems.
- There is a quite high price for it – performance (and decreased storage size).
- Changes need to be merged to upstream kernel.
- Userspace is based on LUKS2.
(Coming one day, really! :)
- Example concept, not a Panacea!

... Integrity protection on other layers ...
if you want it, send a patch, thx :-)

Centre for Research on
Cryptography and Security

Thanks for your attention.

Q & A ?

mbroz@redhat.com
xbroz@fi.muni.cz

DevConf, Brno
January 28, 2017