dm-integrity

The dm-integrity target emulates a block device that has additional per-sector tags that can be used for storing integrity information.

A general problem with storing integrity tags with every sector is that writing the sector and the integrity tag must be atomic - i.e. in case of crash, either both sector and integrity tag or none of them is written.

To guarantee write atomicity, the dm-integrity target uses journal, it writes sector data and integrity tags into a journal, commits the journal and then copies the data and integrity tags to their respective location.

The dm-integrity target can be used with the dm-crypt target - in this situation the dm-crypt target creates the integrity data and passes them to the dm-integrity target via bio_integrity_payload attached to the bio. In this mode, the dm-crypt and dm-integrity targets provide authenticated disk encryption - if the attacker modifies the encrypted device, an I/O error is returned instead of random data.

The dm-integrity target can also be used as a standalone target, in this mode it calculates and verifies the integrity tag internally. In this mode, the dm-integrity target can be used to detect silent data corruption on the disk or in the I/O path.

There's an alternate mode of operation where dm-integrity uses bitmap instead of a journal. If a bit in the bitmap is 1, the corresponding region's data and integrity tags are not synchronized - if the machine crashes, the unsynchronized regions will be recalculated. The bitmap mode is faster than the journal mode, because we don't have to write the data twice, but it is also less reliable, because if data corruption happens when the machine crashes, it may not be detected.

When loading the target for the first time, the kernel driver will format the device. But it will only format the device if the superblock contains zeroes. If the superblock is neither valid nor zeroed, the dm-integrity target can't be loaded.

To use the target for the first time:

1. overwrite the superblock with zeroes
2. load the dm-integrity target with one-sector size, the kernel driver will format the device
3. unload the dm-integrity target
4. read the "provided_data_sectors" value from the superblock
5. load the dm-integrity target with the target size "provided_data_sectors"
6. if you want to use dm-integrity with dm-crypt, load the dm-crypt target with the size "provided_data_sectors"

Target arguments:

1. the underlying block device
2. the number of reserved sector at the beginning of the device - the dm-integrity won't read of write these sectors
3. the size of the integrity tag (if "-" is used, the size is taken from the internal-hash algorithm)
4. mode:

   D - direct writes (without journal)
   : in this mode, journaling is not used and data sectors and integrity tags are written separately. In case of crash, it is possible that the data and integrity tag doesn't match.

   J - journaled writes
   : data and integrity tags are written to the journal and atomicity is guaranteed. In case of crash, either both data and tag or none of them are written. The journaled mode degrades write throughput twice because the data have to be written twice.

   B - bitmap mode - data and metadata are written without any
   : synchronization, the driver maintains a bitmap of dirty regions where data and metadata don't match. This mode can only be used with internal hash.

   R - recovery mode - in this mode, journal is not replayed,
   : checksums are not checked and writes to the device are not allowed. This mode is useful for data recovery if the device cannot be activated in any of the other standard modes.

5. the number of additional arguments

Additional arguments:

journal_sectors:number
: The size of journal, this argument is used only if formatting the device. If the device is already formatted, the value from the superblock is used.

interleave_sectors:number
: The number of interleaved sectors. This values is rounded down to a power of two. If the device is already formatted, the value from the superblock is used.

meta_device:device

Don't interleave the data and metadata on the device. Use a separate device for metadata.

buffer_sectors:number

The number of sectors in one buffer. The value is rounded down to a power of two.

The tag area is accessed using buffers, the buffer size is configurable. The large buffer size means that the I/O size will be larger, but there could be less I/Os issued.

journal_watermark:number

The journal watermark in percents. When the size of the journal exceeds this watermark, the thread that flushes the journal will be started.

commit_time:number

Commit time in milliseconds. When this time passes, the journal is written. The journal is also written immediately if the FLUSH request is received.

internal_hash:algorithm(:key) (the key is optional)

Use internal hash or crc. When this argument is used, the dm-integrity target won't accept integrity tags from the upper target, but it will automatically generate and verify the integrity tags.

You can use a crc algorithm (such as crc32), then integrity target will protect the data against accidental corruption. You can also use a hmac algorithm (for example "hmac(sha256):0123456789abcdef"), in this mode it will provide cryptographic authentication of the data without encryption.

When this argument is not used, the integrity tags are accepted from an upper layer target, such as dm-crypt. The upper layer target should check the validity of the integrity tags.

recalculate

Recalculate the integrity tags automatically. It is only valid when using internal hash.

journal_crypt:algorithm(:key) (the key is optional)

Encrypt the journal using given algorithm to make sure that the attacker can't read the journal. You can use a block cipher here (such as "cbc(aes)") or a stream cipher (for example "chacha20" or "ctr(aes)").

The journal contains history of last writes to the block device, an attacker reading the journal could see the last sector numbers that were written. From the sector numbers, the attacker can infer the size of files that were written. To protect against this situation, you can encrypt the journal.

journal_mac:algorithm(:key) (the key is optional)

Protect sector numbers in the journal from accidental or malicious modification. To protect against accidental modification, use a crc algorithm, to protect against malicious modification, use a hmac algorithm with a key.

This option is not needed when using internal-hash because in this mode, the integrity of journal entries is checked when replaying the journal. Thus, modified sector number would be detected at this stage.

block_size:number

The size of a data block in bytes. The larger the block size the less overhead there is for per-block integrity metadata. Supported values are 512, 1024, 2048 and 4096 bytes. If not specified the default block size is 512 bytes.

sectors_per_bit:number

In the bitmap mode, this parameter specifies the number of 512-byte sectors that corresponds to one bitmap bit.

bitmap_flush_interval:number

The bitmap flush interval in milliseconds. The metadata buffers are synchronized when this interval expires.

allow_discards

Allow block discard requests (a.k.a. TRIM) for the integrity device. Discards are only allowed to devices using internal hash.

fix_padding

Use a smaller padding of the tag area that is more space-efficient. If this option is not present, large padding is used - that is for compatibility with older kernels.

fix_hmac

Improve security of internal_hash and journal_mac:

- the section number is mixed to the mac, so that an attacker can't copy sectors from one journal section to another journal section
- the superblock is protected by journal_mac
- a 16-byte salt stored in the superblock is mixed to the mac, so that the attacker can't detect that two disks have the same hmac key and also to disallow the attacker to move sectors from one disk to another

legacy_recalculate

Allow recalculating of volumes with HMAC keys. This is disabled by default for security reasons - an attacker could modify the volume, set recalc_sector to zero, and the kernel would not detect the modification.

The journal mode (D/J), buffer_sectors, journal_watermark, commit_time and allow_discards can be changed when reloading the target (load an inactive table and swap the tables with suspend and resume). The other arguments should not be changed when reloading the target because the layout of disk data depend on them and the reloaded target would be non-functional.

Status line:

1. the number of integrity mismatches
2. provided data sectors - that is the number of sectors that the user could use
3. the current recalculating position (or '-' if we didn't recalculate)

The layout of the formatted block device:

- reserved sectors
  (they are not used by this target, they can be used for storing LUKS metadata or for other purpose), the size of the reserved area is specified in the target arguments
- superblock (4kiB)
  - magic string - identifies that the device was formatted
  - version
  - log2(interleave sectors)
  - integrity tag size
  - the number of journal sections
  - provided data sectors - the number of sectors that this target provides (i.e. the size of the device minus the size of all metadata and padding). The user of this target should not send bios that access data beyond the "provided data sectors" limit.
  - flags

    SB_FLAG_HAVE_JOURNAL_MAC
    - a flag is set if journal_mac is used

    SB_FLAG_RECALCULATING
    - recalculating is in progress

    SB_FLAG_DIRTY_BITMAP
    - journal area contains the bitmap of dirty blocks
  - log2(sectors per block)
  - a position where recalculating finished
- journal

  The journal is divided into sections, each section contains:
  - metadata area (4kiB), it contains journal entries
    - every journal entry contains:
      - logical sector (specifies where the data and tag should be written)
      - last 8 bytes of data
      - integrity tag (the size is specified in the superblock)
    - every metadata sector ends with
      - mac (8-bytes), all the macs in 8 metadata sectors form a 64-byte value. It is used to store hmac of sector numbers in the journal section, to protect against a possibility that the attacker tampers with sector numbers in the journal.
      - commit id
  - data area (the size is variable; it depends on how many journal entries fit into the metadata area)
    - every sector in the data area contains:
      - data (504 bytes of data, the last 8 bytes are stored in the journal entry)
      - commit id

  To test if the whole journal section was written correctly, every 512-byte sector of the journal ends with 8-byte commit id. If the commit id matches on all sectors in a journal section, then it is assumed that the section was written correctly. If the commit id doesn't match, the section was written partially and it should not be replayed.
- one or more runs of interleaved tags and data.

  Each run contains:

- tag area - it contains integrity tags. There is one tag for each sector in the data area
- data area - it contains data sectors. The number of data sectors in one run must be a power of two. log2 of this value is stored in the superblock.