DCMTK - Bug #1236

Out of bounds read in dcmqrdb's deleteOldestImages() method

2026-06-22 22:22 - Michael Onken

Status:	Closed	**Start date:**	2026-06-22
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0:00 hour
Target version:			
Module:	dcmqrdb	**Compiler:**	
Operating System:			

Description

As reported by Yuxiao Yan:

File	dcmqrdb/libsrc/dcmqrdbi.cc
Function	DcmQueryRetrieveIndexDatabaseHandle::deleteOldestImages
Key Lines	2462 (malloc), 2505–2518 (delete loop), 2508 (OOB read)
Commit	5708ba6cd446d3e7b2cccc6c3a0bfb94b9aa5db9
Method	ASAN harness — recompiled dcmqrdbi.cc with -fsanitize=address -O1 -g, called deleteOldestImages() directly with controlled divergent state; ASAN fired at dcmqrdbi.cc:2508

ASAN Runtime Evidence

```
==684==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x7ffff698a180
READ of size 4 at 0x7ffff698a180 thread T0
    #0 DcmQueryRetrieveIndexDatabaseHandle::deleteOldestImages(StudyDescRecord*, int, char*, long)
       /src/dcmtk/dcmqrdb/libsrc/dcmqrdbi.cc:2508
    #1 main /tmp/direct_harness.cc:123

0x7ffff698a180 is located 0 bytes after 240000-byte region [0x7ffff694f800,0x7ffff698a180)
allocated by thread T0 here:
    #0 malloc ../../../../src/libsanitizer/asan/asan_malloc_linux.cpp:69
    #1 DcmQueryRetrieveIndexDatabaseHandle::deleteOldestImages(StudyDescRecord*, int, char*, long)
       /src/dcmtk/dcmqrdb/libsrc/dcmqrdbi.cc:2462

SUMMARY: AddressSanitizer: heap-buffer-overflow dcmqrdbi.cc:2508
         in DcmQueryRetrieveIndexDatabaseHandle::deleteOldestImages
Shadow bytes at the faulting address: [fa] — Heap left redzone (past end of allocation)
```

READ of size 4 = Uint32 idxCounter (first field of ImagesofStudyArray) read at StudyArray[10000], which is 0 bytes past the end of the malloc(10000 * sizeof(ImagesofStudyArray)) = malloc(240000) allocation.

Root Cause Analysis

Vulnerable Code — dcmqrdbi.cc:2462–2518

```
StudyArray = (ImagesofStudyArray *)malloc(MAX_NUMBER_OF_IMAGES * sizeof(ImagesofStudyArray));
//                                        ^10000              ^24 bytes = 240000 bytes total

// Fill loop: find all index records whose StudyInstanceUID matches StudyUID
DB_IdxInitLoop(&(handle_->idxCounter));
while (DB_IdxGetNext(&(handle_->idxCounter), &idxRec) == EC_Normal) {
    if (!strncmp(idxRec.StudyInstanceUID, StudyUID, n)) {
        StudyArray[nbimages].idxCounter   = handle_->idxCounter;
        StudyArray[nbimages].RecordedDate = idxRec.RecordedDate;
        StudyArray[nbimages++].ImageSize  = idxRec.ImageSize;
```

```
        if (nbimages == MAX_NUMBER_OF_IMAGES) return QR_EC_IndexDatabaseError;
    }
}

// Delete loop — NO BOUND ON s:
s = 0; DeletedSize = 0;
while (DeletedSize < RequiredSize) {
    IdxRecord idxRemoveRec;
    DB_IdxRead(StudyArray[s].idxCounter, &idxRemoveRec);  // LINE 2508 — OOB when s >= nbimages
    deleteImageFile(idxRemoveRec.filename);
    DB_IdxRemove(StudyArray[s].idxCounter);
    pStudyDesc[StudyNum].NumberofRegistratedImages -= 1;
    pStudyDesc[StudyNum].StudySize -= StudyArray[s].ImageSize;
    DeletedSize += StudyArray[s++].ImageSize;
}
```

Missing invariant: while (DeletedSize < RequiredSize) has no check s < nbimages. When RequiredSize exceeds the sum of ImageSize values for the matched records, s advances past nbimages. If nbimages 0, the very first iteration reads StudyArray[0] which is uninitialized heap. At s MAX_NUMBER_OF_IMAGES 10000, StudyArray[10000] is one element past the end of the 240,000-byte allocation → **heap-buffer-overflow**.

Divergent State Condition

The vulnerability requires pStudyDesc[StudyNum].StudySize (maintained in the study descriptor) to exceed the actual total size of matching index records. This divergence arises from:

1. **Direct path**: StudySize is accumulated across C-STORE requests (line 2624), while index records can diverge if images are removed outside of normal quota paths or if the descriptor is written while the index is in a different state.
2. **nbimages 0**: If no index records match StudyUID (e.g., after index corruption, or if StudySize was inherited from a previous study reusing the same slot) while the descriptor reports non-zero images and a large StudySize, RequiredSize > 0 but nbimages = 0, guaranteeing the OOB at the first loop iteration.
3. **Reachability**: deleteOldestImages is called from checkupinStudyDesc (line 2592) whenever pStudyDesc[s].StudySize + imageSize > maxBytesPerStudy && imageSize <= maxBytesPerStudy. This is the quota-enforcement path triggered by every C-STORE into a study that would exceed its per-study size limit.

Triggering Scenario

1. Configure dcmqrscp with a small per-study quota (maxBytesPerStudy).
2. Pre-populate the study descriptor (via index manipulation or natural drift) so that StudySize is close to or equal to maxBytesPerStudy while the corresponding index has zero or fewer records than implied.
3. Send a C-STORE request for a new image in that study with imageSize ≤ maxBytesPerStudy and StudySize + imageSize > maxBytesPerStudy.
4. checkupinStudyDesc computes RequiredSize > 0 and calls deleteOldestImages.
5. Fill loop finds 0 matching index records → nbimages = 0.
6. Delete loop runs with DeletedSize = 0, incrementing s each iteration (all ImageSize = 0 from uninitialized malloc).
7. At s = 10000: StudyArray[10000] → **heap-buffer-overflow**.

Harness Method

Approach

Since deleteOldestImages is a private method, a thin access patch was used:

- Copied dcmqrdbi.h to /tmp/patched_include/, changed private: → public: (access modifier only; no code changes).
- Recompiled dcmqrdbi.cc from unmodified upstream source (5708ba6) with -fsanitize=address -O1 -g using identical original build flags.
- Replaced dcmqrdbi.cc.o in a copy of libdcmqrdb.a with the ASAN-instrumented object.
- Compiled harness harness_direct.cc against the patched library.
- Harness calls deleteOldestImages(pStudyDesc, 0, studyUID, LONG_MAX) with:
  - pStudyDesc[0].NumberofRegistratedImages = 1, StudySize = 9000
  - Index file has 0 IdxRecord entries for studyUID → nbimages = 0
  - RequiredSize = LONG_MAX — guarantees the loop reaches s = MAX_NUMBER_OF_IMAGES (10000)

Build Steps (inside Docker container reverify_dicom_dcmqrscp)

```
bash pocs/QRDB-002/build_and_run.sh
```

Key Outcome

- **ASAN fired at dcmqrdbi.cc:2508** (StudyArray[s].idxCounter read)
- s = 10000 = MAX_NUMBER_OF_IMAGES
- Address 0x7ffff698a180 is **0 bytes after** the 240000-byte StudyArray allocation
- Shadow byte 0xfa = **Heap left redzone** — past end of allocation

Impact

- **Memory safety**: Heap-buffer-overflow READ, potentially overflowing into the ASAN redzone or adjacent heap metadata in production (without ASAN). On real allocators, StudyArray[10000] through higher indices read arbitrary heap contents.
- **Functional**: The loop also reads garbage idxCounter values and passes them to DB_IdxRead and DB_IdxRemove, potentially corrupting the index file with spurious writes (via DB_IdxRemove writing to garbage offsets).
- **Data loss**: Files corresponding to garbage idxRemoveRec.filename are passed to unlink(). If garbage filenames accidentally match real files, data is deleted.
- **Availability**: Process crash (ASAN) or, in production, the heap corruption may cause a crash in the dcmqrscp daemon, denying DICOM storage service.

Fix

Add an upper-bound check on s in the delete loop:

```
 s = 0; DeletedSize = 0;
 while (DeletedSize < RequiredSize) {
+    if (s >= nbimages) {
+        DCMQRDB_ERROR("deleteOldestImages: exhausted all " << nbimages
+            << " images but DeletedSize=" << DeletedSize
+            << " < RequiredSize=" << RequiredSize);
+        break;  // or return QR_EC_IndexDatabaseError
+    }
     IdxRecord idxRemoveRec;
     DB_IdxRead(StudyArray[s].idxCounter, &idxRemoveRec);
     ...
     DeletedSize += StudyArray[s++].ImageSize;
 }
```

Files

File	Description
pocs/QRDB-002/harness_direct.cc	Direct ASAN harness — calls deleteOldestImages with controlled divergent state
pocs/QRDB-002/harness_storerequest.cc	Alternative harness via storeRequest() public API
pocs/QRDB-002/asan_output.txt	Raw ASAN stderr output from confirmed run
pocs/QRDB-002/build_and_run.sh	Reproducible build + run script for Docker container

History

#1 - 2026-07-02 22:26 - Michael Onken

Closed by commit bd8163.

#2 - 2026-07-02 22:27 - Michael Onken

- Status changed from New to Closed

#3 - 2026-07-02 22:29 - Michael Onken

- Private changed from Yes to No