OpenH264 Memory Scan

Table of Contents

guangwwa@cisco.com; huili2@cisco.com; Huashi@cisco.com

1. UBSan check

Overall memory check:

https://clang.llvm.org/docs/index.html

UBSan:

https://clang.llvm.org/docs/UndefinedBehaviorSanitizer.html

Jenkins job:

http://10.140.198.27:8080/view/Robustness_test/job/Openh264_MemCheck_UBSAN/configure

UBSan

Introduction

UndefinedBehaviorSanitizer (UBSan) is a fast undefined behavior detector. UBSan modifies the program at compile-time to catch various kinds of undefined behavior during program execution, for example:

- Using misaligned or null pointer
- Signed integer overflow
- Conversion to, from, or between floating-point types which would overflow the destination

See the full list of available **checks** below.

UBSan has an optional run-time library which provides better error reporting. The checks have small runtime cost and no impact on address space layout or ABI.

Error output:

```
./codec/common/inc/golomb_common.h:103:3: runtime error: shift exponent 32 is too large for 32-bit type 'uint32_t' (aka 'unsigned int')
./codec/common/inc/golomb_common.h:103:3: runtime error: shift exponent 32 is too large for 32-bit type 'uint32_t' (aka 'unsigned int')
./codec/common/inc/golomb_common.h:103:3: runtime error: shift exponent 32 is too large for 32-bit type 'uint32_t' (aka 'unsigned int')
./codec/common/inc/golomb_common.h:103:3: runtime error: shift exponent 32 is too large for 32-bit type 'uint32_t' (aka 'unsigned int')
test/encoder/EncUT_EncoderExt.cpp:732:52: runtime error: signed integer overflow: 1703459111 * 2 cannot be represented in type 'int'
/home/jenkins/Jenkins_Home/workspace/Openh264_MemCheck_UBSAN/codec_unittest:0x91ae35: runtime error: value -inf is outside the range of
./codec/common/inc/golomb_common.h:103:3: runtime error: shift exponent 32 is too large for 32-bit type 'uint32_t' (aka 'unsigned int')
./codec/common/inc/golomb_common.h:103:3: runtime error: shift exponent 32 is too large for 32-bit type 'uint32_t' (aka 'unsigned int')
./codec/common/inc/golomb_common.h:103:3: runtime error: shift exponent 32 is too large for 32-bit type 'uint32_t' (aka 'unsigned int')
./codec/common/inc/golomb_common.h:103:3: runtime error: shift exponent 32 is too large for 32-bit type 'uint32_t' (aka 'unsigned int')
codec/encoder/core/src/svc_encode_slice.cpp:1057:40: runtime error: member access within misaligned address 0x99047c1cd3bd16f4 for type
./codec/common/inc/golomb_common.h:103:3: runtime error: shift exponent 32 is too large for 32-bit type 'uint32_t' (aka 'unsigned int')
./codec/common/inc/golomb_common.h:103:3: runtime error: shift exponent 32 is too large for 32-bit type 'uint32_t' (aka 'unsigned int')
./codec/common/inc/golomb_common.h:103:3: runtime error: shift exponent 32 is too large for 32-bit type 'uint32_t' (aka 'unsigned int')
./codec/common/inc/golomb_common.h:103:3: runtime error: shift exponent 32 is too large for 32-bit type 'uint32_t' (aka 'unsigned int')
test/encoder/EncUT_EncoderExt.cpp:731:51: runtime error: signed integer overflow: 1330426953 * 2 cannot be represented in type 'int'
/home/jenkins/Jenkins_Home/workspace/Openh264_MemCheck_UBSAN/codec_unittest:0x91ae35: runtime error: value -inf is outside the range of
./codec/common/inc/golomb_common.h:103:3: runtime error: shift exponent 32 is too large for 32-bit type 'uint32_t' (aka 'unsigned int')
./codec/common/inc/golomb_common.h:103:3: runtime error: shift exponent 32 is too large for 32-bit type 'uint32_t' (aka 'unsigned int')
./codec/common/inc/golomb_common.h:103:3: runtime error: shift exponent 32 is too large for 32-bit type 'uint32_t' (aka 'unsigned int')
./codec/common/inc/golomb_common.h:103:3: runtime error: shift exponent 32 is too large for 32-bit type 'uint32_t' (aka 'unsigned int')
codec/encoder/core/src/svc_encode_slice.cpp:1057:40: runtime error: member access within misaligned address 0x2edda7b253b96464 for type
```

Scripts:

```
#!/bin/bash
set -x

#vBitPath="/home/jenkins/avc_bits/from_Github"

vOut="OutputUBSAN.txt"
vKey="anitizer|runtime error|egmentation|== Warning|== Error"
vErr="Error.txt"
vMail="Mail.txt"

aECMode=(0 7) # Error Concealment mode, please refer to openh264 api code
vNum=0

function GenerateExe() {
  echo "Current EC mode is $1"
  sed -i "/int32_t iErrorConMethod/c   int32_t iErrorConMethod = $1;" codec/console/dec/src/h264dec.cpp
  make clean >> /dev/null
  make 1>>/dev/null
}

function RunAllTest() {
for vFile in `find ${vBitPath} -name "*.*264"`
do
  echo "Testing Sequence is: "$vFile >>$vOut
  ./h264dec $vFile ./test.yuv 1>>$vOut 2>>$vOut
  rm -f ./test.yuv
  vNum=`expr $vNum + 1`
done
}

echo "Clean the status files:"
rm -f $vOut  && rm -f $vErr && rm -f $vMail

echo "Build Name: ${parent_project}#${parent_build_number}-${NODE_NAME}" >> $vMail
export CC="clang -fsanitize=undefined -fno-sanitize=vptr"
export CXX="clang++ -fsanitize=undefined -fno-sanitize=vptr"
export LD="clang++"
export LDFLAGS="-fsanitize=undefined"

if [ ! -d "./gtest" ]; then
  echo "Prepare the gtest code:" >> $vOut
  make gtest-bootstrap
fi

sed -i 's/\/\/#define NO_DELAY_DECODING/#define NO_DELAY_DECODING/' codec/console/dec/src/h264dec.cpp
for vECMode in ${aECMode[@]}
do
  echo "EC Mode == $vECMode" >> $vOut
  vNum=0
  GenerateExe $vECMode
  ./codec_unittest 1>>$vOut 2>>$vOut
  RunAllTest $vBitPath
done

grep -E "$vKey" $vOut >> $vErr

if [ -s $vErr ]
then
  echo "The listed error occur. Please check it." >> $vMail
  cat $vErr >> $vMail
else
  echo "No error found in this Test. Good luck." >> $vMail
fi

exit 0
```

2. ASan check

ASan:

https://clang.llvm.org/docs/AddressSanitizer.html

Jenkins job:

http://10.140.198.27:8080/view/Robustness_test/job/Openh264_MemCheck_ASAN/

Introduction

AddressSanitizer is a fast memory error detector. It consists of a compiler instrumentation module and a run-time library. The tool can detect the following types of bugs:

- Out-of-bounds accesses to heap, stack and globals
- Use-after-free
- Use-after-return (runtime flag *ASAN_OPTIONS=detect_stack_use_after_return=1*)
- Use-after-scope (clang flag *-fsanitize-address-use-after-scope*)
- Double-free, invalid free
- Memory leaks (experimental)

Typical slowdown introduced by AddressSanitizer is **2x**.

Scripts:

```
#!/bin/bash
set -x

vOut="OutputASAN.txt"
vKey="anitizer|runtime error|egmentation|== Waring|== Error"
vErr="Error.txt"  && vMail="Mail.txt"

aECMode=(0 7) # Error Concealment mode, please refer to openh264 api code
vNum=0

function GenerateExe() {
  echo "Current EC mode is $1"
  sed -i "/int32_t iErrorConMethod/c   int32_t iErrorConMethod = $1;" codec/console/dec/src/h264dec.cpp
  make clean >> /dev/null
  make USE_ASAN=Yes 1>>/dev/null
}

function RunAllTest() {
for vFile in `find ${vBitPath} -name "*.*264"`
do
  echo "Testing Sequence is: "$vFile >>$vOut
  ./h264dec $vFile ./test.yuv 1>>$vOut 2>>$vOut
  rm -f ./test.yuv
  vNum=`expr $vNum + 1`
done
}

echo "Clean the status files:"
rm -f $vOut  && rm -f $vErr  && rm -f $vMail

export CC="clang"
export CXX="clang++"

if [ ! -d "./gtest" ]; then
  echo "Prepare the gtest code:" >> $vOut
  make gtest-bootstrap
fi

#echo "First test the delay mode:" >> $vOut
#for vECMode in ${aECMode[@]}
#do
#  echo "EC Mode == $vECMode" >> $vOut
#  vNum=0
#  GenerateExe $vECMode
#  ./codec_unittest 1>>$vOut 2>>$vOut
#  RunAllTest $vBitPath
#done

echo "Second test the no-delay mode:" >> $vOut
sed -i 's/\/\/#define NO_DELAY_DECODING/#define NO_DELAY_DECODING/' codec/console/dec/src/h264dec.cpp
for vECMode in ${aECMode[@]}
do
  echo "EC Mode == $vECMode" >> $vOut
  vNum=0
  GenerateExe $vECMode
  ./codec_unittest 1>>$vOut 2>>$vOut
  RunAllTest $vBitPath
done

grep -E "$vKey" $vOut >> $vErr

if [ -s $vErr ] ; then
  echo "The listed error occur. Please check it." >> $vMail
  cat $vErr >> $vMail
else
  echo "No error found in this Test. Good luck." >> $vMail
fi
exit 0
```

3. Low memory check

http://10.140.198.27:8080/view/Robustness_test/job/Openh264_MemCheck_LowMemory/

Tools:

ulimit: User limits - limit the use of system-wide resources.

https://ss64.com/bash/ulimit.html

Syntax

ulimit [-acdfHlmnpsStuv] [limit]

Options

-S Change and report the soft limit associated with a resource.

-H Change and report the hard limit associated with a resource.

-a All current limits are reported.

-c The maximum size of core files created.

-d The maximum size of a process's data segment.

-f The maximum size of files created by the shell(default option)

-l The maximum size that can be locked into memory.

-m The maximum resident set size.

-n The maximum number of open file descriptors.

-p The pipe buffer size.

-s The maximum stack size.

-t The maximum amount of cpu time in seconds.

-u The maximum number of processes available to a single user.

-v The maximum amount of virtual memory available to the process.

Run on:

ubuntu-64bits-hf-CodeNomicon

ErrorOutput (Example):

```
[----------] 6 tests from CSliceBufferReallocatTest
[ RUN      ] CSliceBufferReallocatTest.Reallocate_in_one_partition
test/encoder/EncUT_SliceBufferReallocate.cpp:193: Failure
Value of: cmResultSuccess == iRet
  Actual: false
Expected: true
[  FAILED  ] CSliceBufferReallocatTest.Reallocate_in_one_partition (12 ms)
[ RUN      ] CSliceBufferReallocatTest.Reallocate_in_one_thread
test/encoder/EncUT_SliceBufferReallocate.cpp:193: Failure
Value of: cmResultSuccess == iRet
  Actual: false
Expected: true
test/encoder/EncUT_SliceBufferReallocate.cpp:495: Failure
Value of: cmResultSuccess == iRet
  Actual: false
Expected: true
test/encoder/EncUT_SliceBufferReallocate.cpp:497: Failure
Value of: iSlcBufferNum < pCtx->pCurDqLayer->sSliceBufferInfo[iThreadIndex].iMaxSliceNum
  Actual: false
Expected: true
```

Script:

```
#!/bin/bash
set -x
# Print some enviornment information
echo ${WORKSPACE}
echo `ifconfig | grep "inet addr"`
echo `nasm -v`

# The test script body

vErr="Error.txt"

aMemorySize=(163840 81920 40960 10240)

export CC="clang"
export CXX="clang++"

if [ ! -d "./gtest" ]
then
  echo "Prepare the gtest code:" >> $vOut
  make gtest-bootstrap
fi

make clean >>/dev/null
make 1>>/dev/null

rm -f $vErr

# The common unlimited case
echo "The common case: " >> $vErr
ulimit -m unlimited -v unlimited
./codec_unittest args >> $vErr
if [ $? -eq 139 ]; then
    echo "It crashed!"
    exit 1
fi

# The lower memory case

for vMemorySize in ${aMemorySize[@]}
do
  rm $vErr #delete the error file for each passed case
  echo "The memory size is: " $vMemorySize >> $vErr
  ulimit -m $vMemorySize -v $vMemorySize
  ./codec_unittest args >> $vErr
  if [ $? -eq 139 ]
  then
    echo "It crashed!"
    exit 1
  fi
done

rm $vErr # delete the error file when success

exit 0
```

4. MSan Check

MSan: https://clang.llvm.org/docs/MemorySanitizer.html

Jenkins job: http://10.140.198.27:8080/view/Robustness_test/job/Openh264_MemCheck_MSAN/

Introduction

MemorySanitizer is a detector of uninitialized reads. It consists of a compiler instrumentation module and a run-time library.

Typical slowdown introduced by MemorySanitizer is **3x**.

Script:

```
########## This job can't work well now, Memory SAN output warning information ############
########## Don't the reasons, the Clang version? or the source code?          ############
########## Some warnings reported from the gtest framework, it sounds bad     ############

#!/bin/bash
set -x
# Print some enviornment information
echo ${WORKSPACE}
echo `ifconfig | grep "inet addr"`
echo `nasm -v`
echo `gcc -v`

# The test script body

#vBitPath="/home/jenkins/avc_bits/from_Github"

vOut="OutputMSAN.txt"
vKey="anitizer|runtime error|egmentation|== Warning|== Error"
vErr="Error.txt"
vMail="Mail.txt"

aECMode=(0 7) # Error Concealment mode, please refer to openh264 api code

vNum=0

function GenerateExe() {
  echo "Current EC mode is $1"
  sed -i "/int32_t iErrorConMethod/c   int32_t iErrorConMethod = $1;" codec/console/dec/src/h264dec.cpp
  make clean >> /dev/null
  make 1>>/dev/null
}

function RunAllTest() {
for vFile in `find ${vBitPath} -name "*.*264"`
do
  echo "Testing Sequence is: "$vFile >>$vOut
  ./h264dec $vFile ./test.yuv 1>>$vOut 2>>$vOut
  rm -f ./test.yuv
  vNum=`expr $vNum + 1`
done
}

echo "Clean the status files:"
rm -f $vOut
rm -f $vErr
rm -f $vMail

CFLAGS +=

export CC="clang -fno-omit-frame-pointer -fsanitize=memory -fno-optimize-sibling-calls -fsanitize-memory-track-origins -g -fstack-protector-all -D_FORTIFY_SOURCE=2"
export CXX="clang++ -fno-omit-frame-pointer -fsanitize=memory -fno-optimize-sibling-calls -fsanitize-memory-track-origins -g -fstack-protector-all -D_FORTIFY_SOURCE=2"
export LD="clang++"
export LDFLAGS="-fsanitize=memory"

if [ ! -d "./gtest" ]
then
  echo "Prepare the gtest code:" >> $vOut
  make gtest-bootstrap
fi
```

```
#echo "First test the delay mode:" >> $vOut
#for vECMode in ${aECMode[@]}
#do
#  echo "EC Mode == $vECMode" >> $vOut
#  vNum=0
#  GenerateExe $vECMode
#  ./codec_unittest 1>>$vOut 2>>$vOut
#  RunAllTest $vBitPath
#done

echo "Second test the no-delay mode:" >> $vOut
sed -i 's/\/\/#define NO_DELAY_DECODING/#define NO_DELAY_DECODING/' codec/console/dec/src/h264dec.cpp
for vECMode in ${aECMode[@]}
do
  echo "EC Mode == $vECMode" >> $vOut
  vNum=0
  GenerateExe $vECMode
  ./codec_unittest 1>>$vOut 2>>$vOut
  RunAllTest $vBitPath
done

grep -E "$vKey" $vOut >> $vErr

if [ -s $vErr ]
then
  echo "The listed error occur. Please check it." >> $vMail
  cat $vErr >> $vMail
else
  echo "No error found in this Test. Good luck." >> $vMail
fi

exit 0
```

5. Valgrind

Valgrind:

http://valgrind.org/

Jenkins job:

http://10.140.198.27:8080/view/Robustness_test/job/Openh264_MemCheck_Valgrind/

About valgrind:

Valgrind is an instrumentation framework for building dynamic analysis tools. There are Valgrind tools that can automatically detect many memory management and threading bugs, and profile your programs in detail. You can also use Valgrind to build new tools.

The Valgrind distribution currently includes six production-quality tools: a memory error detector, two thread error detectors, a cache and branch-prediction profiler, a call-graph generating cache and branch-prediction profiler, and a heap profiler. It also includes three experimental tools: a stack/global array overrun detector, a second heap profiler that examines how heap blocks are used, and a SimPoint basic block vector generator. It runs on the following platforms: X86/Linux, AMD64/Linux, ARM/Linux, ARM64/Linux, PPC32/Linux, PPC64/Linux, PPC64LE/Linux, S390X/Linux, MIPS32/Linux, MIPS64/Linux, TILEGX/Linux, X86/Solaris, AMD64/Solaris, ARM/Android (2.3.x and later), ARM64/Android, X86/Android (4.0 and later), MIPS32/Android, X86/Darwin and AMD64/Darwin (Mac OS X 10.10, with initial support for 10.11).

Valgrind is Open Source / Free Software, and is freely available under the GNU General Public License, version 2.

Script:

```
#!/bin/bash
set -x
vOut="OutputValgrind.txt"
vKey="anitizer|runtime error|egmentation|== Warning|== Error"
vErr="Error.txt"
vMail="Mail.txt"

aECMode=(0 7) # Error Concealment mode, please refer to openh264 api code
vNum=0

function GenerateExe() {
  echo "Current EC mode is $1"
  sed -i "/int32_t iErrorConMethod/c   int32_t iErrorConMethod = $1;" codec/console/dec/src/h264dec.cpp
  make clean >> /dev/null
  make 1>>/dev/null
}

function RunAllTest() {
for vFile in `find ${vBitPath} -name "*.*264"`
do
  echo "Testing Sequence is: "$vFile >>$vOut
  valgrind ./h264dec $vFile ./test.yuv 1>>$vOut 2>>$vOut
  rm -f ./test.yuv
  vNum=`expr $vNum + 1`
done
}

echo "Clean the status files:"
rm -f $vOut && rm -f $vErr && rm -f $vMail

if [ ! -d "./gtest" ]
then
  echo "Prepare the gtest code:" >> $vOut
  make gtest-bootstrap
fi

echo "Second test the no-delay mode:" >> $vOut
sed -i 's/\/\/#define NO_DELAY_DECODING/#define NO_DELAY_DECODING/' codec/console/dec/src/h264dec.cpp
for vECMode in ${aECMode[@]}
do
  echo "EC Mode == $vECMode" >> $vOut
  vNum=0
  GenerateExe $vECMode
  valgrind ./codec_unittest 1>>$vOut 2>>$vOut
  RunAllTest $vBitPath
done

grep -E "$vKey" $vOut >> $vErr

if [ -s $vErr ]
then
  echo "The listed error occur. Please check it." >> $vMail
  cat $vErr >> $vMail
else
  echo "No error found in this Test. Good luck." >> $vMail
fi

exit 0
```