作者	日期
王胜柯(wangshengke@kedacom.com)	2021-9-27

c++内存溢出、越界排查技能

1、 案例背景介绍

C++实现的负责会议的 cmu 模块，在实验局上每隔一个月就有崩溃产生，尝试了异常堆栈捕获、core文件 等方法，通过core内存分析发现每次崩溃点都不同，第一次是new的管理对象被破坏，第二次是vector的end管理指针被破坏，第三次是 osp 的二元信号量管理对像被破坏，但都没能锁定问题根源（gdb调试core文件技能不在本文展开）；

1.1、 第一次崩溃堆栈及内存值：

```
#14 <signal handler called>
#15 0xf770f430 in __kernel_vsyscall ()
#16 0xf6675227 in raise () from /lib/libc.so.6
#17 0xf6676a63 in abort () from /lib/libc.so.6
#18 0xf66b6fd5 in __libc_message () from /lib/libc.so.6
#19 0xf66bddfc in malloc_printerr () from /lib/libc.so.6
#20 0xf66c0e98 in _int_malloc () from /lib/libc.so.6
#21 0xf66c296a in malloc () from /lib/libc.so.6
#22 0xf68b9c97 in operator new(unsigned int) () from
/lib/libstdc++.so.6

(gdb) x/64x $esp
0xecbd8620:     0x00000000      0x000000ff      0x00000000
0x00000000
0xecbd8630:     0x00000000      0x00000000      0x00000000
0x00000000
0xecbd8640:     0x00000000      0xea900010      0x00000000
0xea973138
0xecbd8650:     0x00000000      0x00000000      0x00000000
0x00000000
0xecbd8660:     0x00000000      0xf7719a37      0x00000000
0x00000000
0xecbd8670:     0x00000000      0x00000000      0x00000000
0xf6953000
0xecbd8680:     0x00000001      0xf680d3c4      0xecbd86d8
0xf68b7aef
0xecbd8690:     0x00000000      0xf771a214      0x00000000
0xecbd873b
0xecbd86a0:     0x00000000      0xdb300010      0xea973148
0x08993748
0xecbd86b0:     0xea973138      0x00000001      0xea973138
0x08f833c8
0xecbd86c0:     0xecbd873b      0xecbd8862      0x08f833c8
0x8519ec00
0xecbd86d0:     0xecbd8862      0xf7719b3d      0xecbd8718
0x08991a65
0xecbd86e0:     0x08f833c8      0xea973138      0x00000001
0x089919be
0xecbd86f0:     0xecbd873b      0x08f833c8      0xea973138
0x08f833c8
```

```
0xecbd8700:     0xf764161c      0x00000001      0xf6953000
0x8519ec00
0xecbd8710:     0xdb30e488      0xf7732fbc      0xecbd8748
0x0898e991
```

1.2、第二次崩溃信息：

```
[0xf775941O]
/opt/mcu/cmu/cmu() [0x83854b3]          <THduInfo::Clear()>:
/opt/mcu/cmu/cmu() [0x8384ecd]          <THduInfo::~THduInfo()>:
/opt/mcu/cmu/cmu() [0x8388ec0]          <void
std::_Destroy<THduInfo>(THduInfo*)>
/opt/mcu/cmu/cmu() [0x8386680]          <void
std::_Destroy<THduInfo*, THduInfo>(THduInfo*, THduInfo*,
std::allocator<THduInfo>&)>:
/opt/mcu/cmu/cmu() [0x8385aa8]          <std::vector<THduInfo,
std::allocator<THduInfo> >::~vector()>:
/opt/mcu/cmu/cmu() [0x83855aa]
<CHduManager::~CHduManager()>:
```

从两次崩溃的排查现象描述，基本可以推断，并非栈溢出、栈变量越界等传统问题导致的，很像堆内存越界访问写入；这个假设成立，那现有的 gdb + core 文件调试定位技术基本无用了，事后内存无法溯源操作用户；

至此，我们需要借助工具了。

2、排查工具选择

2.1、行业备选工具列表

行业内一直存在很多内存检测工具，但都各具特色，主要徘徊在速度和可检测问题范围之间拉锯，需要根据你问题去筛选；

1. D.U.M.A. - Detect Unintended Memory Access. http://duma.sourceforge.net/.
2. Intel. Intel Parallel Inspector. http://software.intel.com/ en-us/intel-parallel-inspector/.
3. Valgrind: A framework for heavyweight dynamic binary instrumentation. In Proc. of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '07), pages 89-100, June 2007.
4. Oracle. Sun Memory Error Discovery Tool (Discover). http://download.oracle.com/docs/cd/E19205-01/ 821-1784/6nmoc18gq/index.html.
5. Parasoft. Insure++. http://www.parasoft.com/jsp/ products/insure.jsp?itemId=63.
6. AddressSanitizer is a fast memory error detector. It consists of a compiler instrumentation module and a run-time library.

2.2、 工具的选择

依据上边问题描述，推断为堆内存操作导致；我们需要一个工具能够实时监测堆栈的内存申请空间、使用长度、是否越界等行为等：

ASan/MSan vs Valgrind (Memcheck)

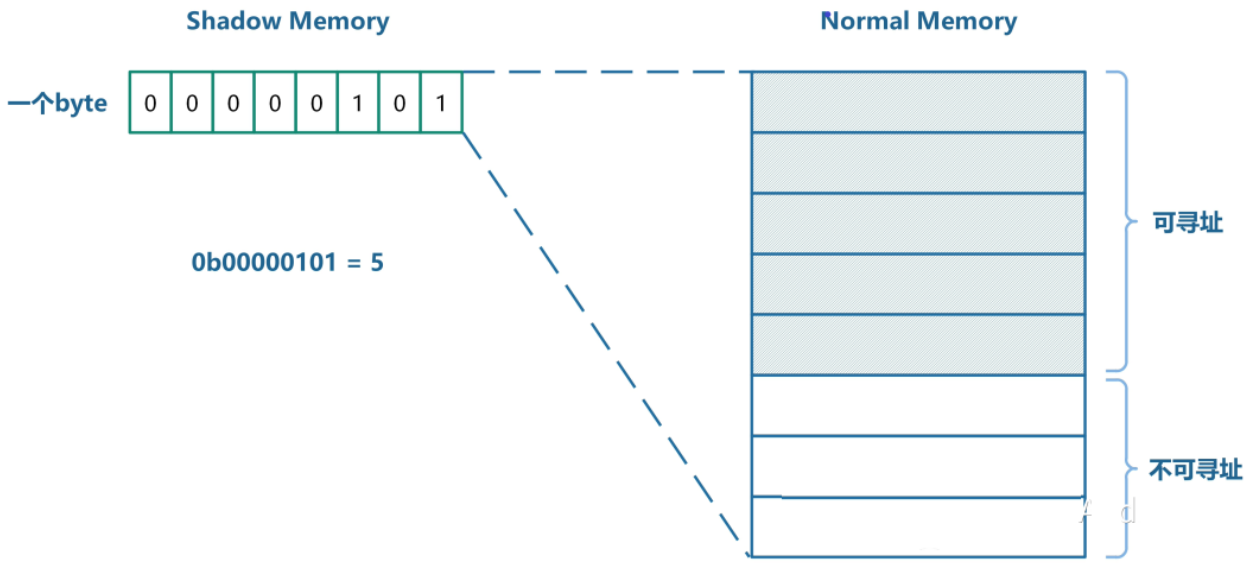
	Valgrind	ASan	MSan
Heap out-of-bounds	YES	YES	NO
Stack out-of-bounds	NO	YES	NO
Global out-of-bounds	NO	YES	NO
Use-after-free	YES	YES	NO
Use-after-return	NO	Sometimes	NO
Uninitialized reads	YES	NO	YES
CPU Overhead	10x-300x	1.5x-3x	3x

2.2.1、valgrind

1. 在行业是名声在外的一款，符合我们监控的要求，但它的缺陷也很明显
2. 它采用的二进制完全映射的影子内存技术，内存需要更多才能监测内存变化；
3. 它会严重降速，调用memcheck工具，基本上是10到30倍的降速，基本上会导致cmu处于不可用状态【引用*valgrind*官网： *The main one is that programs run significantly more slowly under Valgrind. Depending on which tool you use, the slowdown factor can range from 5--100. Memcheck runs programs about 10--30x slower than normal.*】；
4. 实测，降速等各种缺陷导致问题并没通过valgrind检测出来；
5. 工具被排除；

2.2.2、AddressSanitizer

1. 使用人不是那么多，但这仍然阻碍不了它的优越表现
2. 采用了一种取巧的影子内存玩法，将虚拟地址空间的1 / 8分配给它的影子内存，并使用一个带有比例和偏移量的直接映射将一个应用程序地址转换为它相应的影子地址，确保了少量内存就能完成一个程序的监测；
3. 本身降速也很少，监测范围也是我们想要的【引自*USENIX*协会论文：*We present AddressSanitizer, a new tool that combines performance and coverage. AddressSanitizer finds out-of-bounds accesses (for heap, stack, and global objects) and uses of freed heap memory at the relatively low cost of 73% slowdown,1.5x-4x memory overhead， making it a good choice for testing a wide range of C/C++ applications*】；
4. 第三个优势，也是我认为对我最有利的，它是内置于gcc和clang的，只需要增加编译开关既可启用；



5. 我们对每个影子字节使用以下编码: 0表示对应应用程序内存区域的所有8个字节都是可寻址的;K (1 _k _7)表示前K个字节是可寻址的;任何负值都表示整个8字节字是不可寻址的。我们使用不同的负值来区分不同类型的不可寻址内存(堆红区、堆栈红区、全局红区、释放内存)。

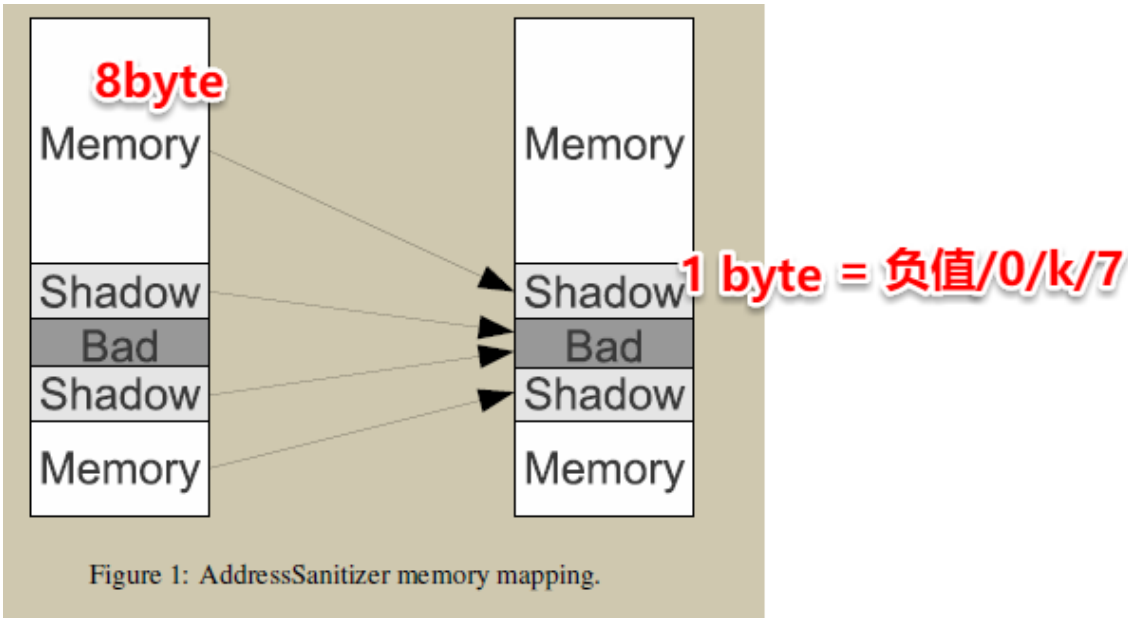


Figure 1: AddressSanitizer memory mapping.

3、 工具解决问题范围

现存C++常见问题包含堆溢出、栈溢出、全局变量越界、已释放内存使用等，这些问题将是本文所能处理的范围。

4、 使用方法、细节

4.1、 强依赖gcc版本

1. AddressSanitizer 第一版本发布于4.8.5的gcc，但是存在缺陷，当监测到任何一个error，它就会强制退出主程序，导致程序无法继续，对于实验局cmu来说，要保障会议这是不能接受的；
2. 所以，使用你可以用的最高版本的gcc，越高版本bug越少，中间还增加了一些功能优化；
3. 我实践的版本：平台编译机用的就是4.8.5无法满足，我重新搭建了一套gcc环境，系统版本和gcc版本如下：

```
[root@wangshengke ~]# cat /etc/centos-release
CentOS Linux release 7.9.2009 (Core)

[root@wangshengke ~]# gcc -v
使用内建 specs。
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/local/libexec/gcc/x86_64-pc-linux-
gnu/7.3.0/lto-wrapper
目标：x86_64-pc-linux-gnu
配置为：../configure --enable-checking=release --enable-
languages=c,c++ --enable-multilib
线程模型：posix
gcc 版本 7.3.0 (GCC)
```

4.2、 编译参数下参

1. 如果没使用makefile，直接命令编译

```
gcc -fsanitize=address -fno-omit-frame-pointer -O1 -g
testmemcheck.c -o testmemcheck
```

2. 使用makefile，需要注意

```
CFLAGS+=-fsanitize=address      #都要追加-fsanitize=address开关
LDFLAGS+=-fsanitize=address     #都要追加-fsanitize=address开关
```

4.3、 解决代码编译失败问题

1. 升级gcc大概率会遇到编译失败，很多是底层libc库版本或c++版本不一致导致；
2. 我只举例其中一个解决方案：-D_GLIBCXX_USE_CXX11_ABI=0追加编译参数，统一API版本；

```
/mnt/20210511_Mustang_V6R1_SP5/80-moservice/mcu/prj_linux/../source/dssfumgrdata.cpp:73: undefined reference to `SFUCLIENT::CRoomNtf
Info::CRoomNtfInfo(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&)'
/mnt/20210511_Mustang_V6R1_SP5/80-moservice/mcu/prj_linux/../source/dssfumgrdata.cpp:77: undefined reference to `SFUCLIENT::CRoomNtf
Info::GetConfE164[abi:cxx11]() const'
/mnt/20210511_Mustang_V6R1_SP5/80-moservice/mcu/prj_linux/../source/dssfumgrdata.cpp:88: undefined reference to `SFUCLIENT::CNtfInfo
::GetJson[abi:cxx11]() const'
/mnt/20210511_Mustang_V6R1_SP5/80-moservice/mcu/prj_linux/../source/dssfumgrdata.cpp:99: undefined reference to `SFUCLIENT::CEndpoin
tNtfInfo::GetConfE164[abi:cxx11]() const'
/mnt/20210511_Mustang_V6R1_SP5/80-moservice/mcu/prj_linux/../source/dssfumgrdata.cpp:100: undefined reference to `SFUCLIENT::CEndpoi
ntNtfInfo::GetEndpointID[abi:cxx11]() const'
/mnt/20210511_Mustang_V6R1_SP5/80-moservice/mcu/prj_linux/../source/dssfumgrdata.cpp:112: undefined reference to `SFUCLIENT::CNtfInf
o::GetJson[abi:cxx11]() const'
/mnt/20210511_Mustang_V6R1_SP5/80-moservice/mcu/prj_linux/../source/dssfumgrdata.cpp:132: undefined reference to `SFUCLIENT::CStream
HasConsumerNtfInfo::GetConfE164[abi:cxx11]() const'
/mnt/20210511_Mustang_V6R1_SP5/80-moservice/mcu/prj_linux/../source/dssfumgrdata.cpp:133: undefined reference to `SFUCLIENT::CStream
```

4.4、 启动

1. 需要声明环境变量

```
export
ASAN_OPTIONS=halt_on_error=0:use_sigaltstack=0:malloc_context_size=
15:log_path=/opt/mcu/cmu/asan.log
```

2. 参数含义

关键字	值	含义
use_sigaltstack	true	If set, uses alternate stack for signal handling.
halt_on_error	true	Crash the program after printing the first error report (WARNING: USE AT YOUR OWN RISK!). The flag has effect only if code was compiled with -fsanitize-recover=address compile option.
malloc_context_size	30	Max number of stack frames kept for each allocation/deallocation.

5、 实际案例

1. cmu崩溃，我们顺利的找到的具体问题；
2. 是由于cmu包含的媒体业务lib中，在对堆内存数组赋值时，用户自定义对象长的总长数组为TChnProperty [25]，使用255下标去寻址写入其中一个u8成员值；
3. 覆盖了堆上部230*sizeof（TChnProperty ）偏移地址对象中的一个字节，导致低概率堆对象被破坏，破坏程度还很小，极难通过gdb + core分析内存内容来推断写入用户；
4. 输出结果也准确的通过红色区域内存保护，感知到了异常访问写入：
    a. 错误类型为（heap-buffer-overflow）
    b. 错误写入了1字节，这一字节属于T18线程（WRITE of size 1 at 0xeb155020 thread T18）
    c. 写入的地址0xeb155020是一个有效地址块的右边区域388 bytes字节处（0xeb155020 is located 388 bytes to the right of 132764-byte region）
    d. 描述存储目标地址的确切影子字节的值为（Heap left redzone: fa）

```
SUMMARY: AddressSanitizer: memcpy-param-overlap
../../../../../libsanitizer/asan/asan_interceptors.cc:456 in
__interceptor_memcpy
==================================================================
```

```
==2992==ERROR: AddressSanitizer: heap-buffer-overflow on address
0xeb15020 at pc 0x09245b86 bp 0xeedd59d8 sp 0xeedd59cc
WRITE of size 1 at 0xeb15020 thread T18 (InterAct)
    #0 0x9245b85 in MediaAPI::TChnProperty::SetChnProperty(unsigned
char, unsigned char) ../../../10-
common/include/platform/mediavmpstruct.h:570
    #1 0x955b6f4 in CMulPic::SetProperty(unsigned char, unsigned
char) ../source/mulpicctrl.cpp:3026
    #2 0x95595e3 in CMulPicCtrl::SetMulPicProperty(unsigned int,
unsigned char, unsigned char) ../source/mulpicctrl.cpp:2443
    #3 0x9555282 in CMulPicCtrl::SetChnnlProperty(unsigned int,
unsigned char, MediaAPI::TChnProperty*)
../source/mulpicctrl.cpp:1195
    #4 0x94e4fb4 in
MediaAPI::CMultiPicMgr::SetChnnlProperty(unsigned int, unsigned
char, MediaAPI::TChnProperty*) ../source/mediaportapi.cpp:1183
    #5 0x8dee222 in CResmgr_Mulpic::SetChnnlProperty(unsigned char,
MediaAPI::TChnProperty*) ../source/resmgr_mulpic.cpp:282
    #6 0x91dd634 in
CInteractionDirector::ProcActorToMulpic(CInteractionObjPair*,
TInteractionVmpParam*, void*)
../source/interactiondirector.cpp:16250
    #7 0x91b1a66 in
CInteractionDirector::ProcMulpicChangeParam(CInteractionObjPair*,
TInteractionVmpParam*, TInteractionResult&)
../source/interactiondirector.cpp:10383
    #8 0x917a291 in
CInteractionDirector::ChangeNToNInteraction(unsigned int, unsigned
char, void*, TInteractionResult&, CViewer*, unsigned int)
../source/interactiondirector.cpp:1485
    #9 0x94654ef in CInteractData::DealChangeNToN(unsigned int,
unsigned int, void*, unsigned char, TInteractionResult&)
../source/interactData.cpp:716
    #10 0x897d98b in CInterActDirInst::ProcChangeNToN(CServMsg&,
TInteractionResult&) ../source/interactdirssn.cpp:941
    #11 0x897aeda in CInterActDirInst::InstanceEntry(CMessage*)
../source/interactdirssn.cpp:462
    #12 0xf6bf4a79 in OspAppEntry
(/opt/mcu/sodir/libosp.so+0x28a79)
    #13 0xf6c0225c  (/opt/mcu/sodir/libosp.so+0x3625c)
    #14 0x80790b0 in asan_thread_start
../../../../../libsanitizer/asan/asan_interceptors.cc:234
    #15 0xf6a1dbbb in start_thread (/lib/libpthread.so.0+0x6bbb)
```

```
    #16 0xf6772Odd in clone (/lib/libc.so.6+0xfeOdd)

0xeb155020 is located 388 bytes to the right of 132764-byte region
[0xeb134800,0xeb154e9c)
allocated by thread T18 (InterAct) here:
    #0 0x812b6d4 in operator new[](unsigned int)
../../../../../libsanitizer/asan/asan_new_delete.cc:82
    #1 0x950b809 in CKdvPlex::Create(CKdvPlex*&, unsigned int,
unsigned int) ../../common/include/kdvplex.h:34
    #2 0x955d001 in CKdvMap<unsigned int,
CMulPic>::NewAssoc(unsigned int) ../../common/include/kdvmap.h:438
    #3 0x955caba in CKdvMap<unsigned int, CMulPic>::AddAt(unsigned
int) ../../common/include/kdvmap.h:203
    #4 0x955855a in CMulPicCtrl::AddMulPicList(unsigned int)
../source/mulpicctrl.cpp:2173
    #5 0x9551174 in CMulPicCtrl::Create(unsigned int&,
MediaAPI::emMulPicType, unsigned char) ../source/mulpicctrl.cpp:100
    #6 0x94e4a6e in MediaAPI::CMultiPicMgr::Create(unsigned int&,
MediaAPI::emMulPicType, unsigned char)
../source/mediaportapi.cpp:905
    #7 0x8d7fbfd in CResManager::ApplyMediaProcEqp(unsigned char,
unsigned char, unsigned int, TEqpParam4Creator*,
MediaAPI::TMediaVidStreamCap*, int, int, unsigned char)
../source/resmanager.cpp:2139
    #8 0x91b4608 in
CInteractionDirector::ProcStartMulpic(CInteractionObjPair*,
TInteractionVmpParam*, unsigned char, TInteractionResult&,
CViewer*, unsigned int) ../source/interactiondirector.cpp:10807
    #9 0x9179d2c in
CInteractionDirector::AddNToNInteraction(unsigned int, void*,
unsigned char, TInteractionResult&, CViewer*, unsigned int)
../source/interactiondirector.cpp:1389
    #10 0x94653ed in CInteractData::DealAddNToN(unsigned int,
unsigned int, void*, unsigned char, TInteractionResult&, CViewer*,
unsigned int) ../source/interactData.cpp:661
    #11 0x897d6f6 in CInterActDirInst::ProcAddNToN(CServMsg&,
TInteractionResult&) ../source/interactdirssn.cpp:909
    #12 0x897ad74 in CInterActDirInst::InstanceEntry(CMessage*)
../source/interactdirssn.cpp:446
    #13 0xf6bf4a79 in OspAppEntry
(/opt/mcu/sodir/libosp.so+0x28a79)
    #14 0xf6c0225c  (/opt/mcu/sodir/libosp.so+0x3625c)
```

```
Thread T18 (InterAct) created by T0 here:
    #0 0x8112a90 in __interceptor_pthread_create
../../../../../libsanitizer/asan/asan_interceptors.cc:243
    #1 0xf6c01db4 in OspTaskCreate
(/opt/mcu/sodir/libosp.so+0x35db4)
    #2 0xf6bf7359 in CApp::CreateApp(char const*, unsigned short,
unsigned char, unsigned short, unsigned int)
(/opt/mcu/sodir/libosp.so+0x2b359)
    #3 0x8e9d930 in userinit ../source/mcu.cpp:545
    #4 0x8eae5cb in main ../source/mcu.cpp:4591
    #5 0xf668e2a2 in __libc_start_main (/lib/libc.so.6+0x1a2a2)

SUMMARY: AddressSanitizer: heap-buffer-overflow ../../../10-
common/include/platform/mediavmpstruct.h:570 in
MediaAPI::TChnProperty::SetChnProperty(unsigned char, unsigned
char)
Shadow bytes around the buggy address:
  0x3d62a9b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x3d62a9c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x3d62a9d0: 00 00 00 04 fa fa fa fa fa fa fa fa fa fa fa fa
  0x3d62a9e0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3d62a9f0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x3d62aa00: fa fa fa fa[fa]fa fa fa fa fa fa fa fa fa fa fa
  0x3d62aa10: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3d62aa20: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3d62aa30: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3d62aa40: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x3d62aa50: 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
==2992==ABORTING
```