HP OpenVMS technical journal V19

User space heap corruption

Table of contents

Introduction

User space applications cause heap corruption when they overwrite the heap data structures created by C runtime library (CRTL) routines. This paper tries to explain how this is related to coding error and how to identify these issues by analyzing process dumps and troubleshooting these issues using HP OpenVMS debugger and other tools.

What is heap?

Heap is an area of virtual address space in user space. This is the virtual address space allocated dynamically. Following runtime library (RTL) routines are called to allocate/deallocate heap.

```
LIB$GET_VM/LIB$FREE_VM (LIB$ RTL)

LIB$GET_VM_64/LIB$FREE_VM_64 (LIB$ RTL)

LIB$GET_VM_PAGE/LIB$FREE_VM_PAGE (LIB$ RTL)

LIB$GET_VM_PAGE_64/LIB$FREE_VM_PAGE_64 (LIB$ RTL)

MALLOC/FREE (DECC RTL or famously known as CRTL)
```

These are the routines most commonly called by applications in user mode and thus address space allocated by these routines are always user mode writable. These routines allocate memory using SYS$EXPREG/ SYS$EXPREG_64 system service. RTL calls manage deallocated memory themselves upon calling an appropriate deallocation routine. Due to this, using RTL calls are preferred than calling system calls for better performance and to avoid the burden of managing the heap by applications.

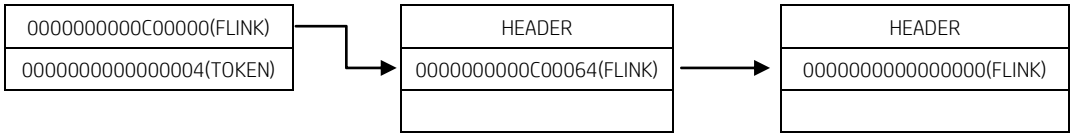
This paper discusses heap corruptions caused when using MALLOC/FREE routines.

Heap internals

MALLOC routine eventually calls SYS$EXPREG/SYS$EXPREG_64 system services to allocate/deallocate virtual address space (VAS). RTL maintains a free list for different sizes. Selection of list of sizes is chosen to make free list maintenance easy. When MALLOC is called by an application, requesting a size of VAS, RTL rounds up the requested size to the nearest size it maintains internally. When FREE routine is called, instead of returning it back to the OS, freed VAS is linked to the free list maintained by RTL. So when the next MALLOC call is called VAS from the free list may be allocated if the free list has a packet matching the requested size.

Figure 1 shows one of the entries in the free list. Free list is a quadword vector having two quadwords for each entry. First quadword contains the first free address in the linked list for a particular size, and the second quadword contains the token used to synchronize the free list access by RTLs in a multithreaded environment.

Figure 1. MALLOC free list



Header for a MALLOC VAS contains 16 bytes. Second quadword of the header contains all zeros.

Figure 2. MALLOC header

PATTERN (2 B)	INDEX (1 B)	NOT USED (1 B)	SIZE (4 B)
NOT USED (4B)			
This QUADWORD address is returned by MALLOC to caller			

Header for a FREE VAS contains 16 bytes. First quadword pointed by the address returned to the user by MALLOC is used to store the next free address in the free list. When a VAS is freed using FREE call, first quadword of the VAS is used by RTL to store the address of the next free pointer. Each allocated/deallocated VAS has a header added by RTL.

Figure 3. FREE Header

PATTERN (2 B)	INDEX (1 B)	NOT USED (1 B)	SIZE (4 B)
NOT USED (4 B)			
FLINK (maintained by free list)			

Pattern field in MALLOC/FREE header stores a pattern, which indicates whether the packet is allocated or deallocated. For an allocated packet pattern is 0xF00D and for a deallocated packet pattern is 0x7777.

What is heap corruption?

Heap management metadata

RTL manages heap by maintaining free lists for different sizes and adding a header to each allocated or deallocated packet as shown in **figure 1, 2, and 3**. This header should be intact for RTL to maintain the heap. Since these data structures are writable from user mode these can be corrupted by applications.

Heap corruption by applications and troubleshooting

Overwriting the buffer

This is a common problem by which applications corrupt RTL heap management data and also can corrupt application data structures or buffer. Attached text file demonstrates this behavior.

overflow.txt

When running OVERFLOW.EXE, MALLOC returns same addresses for ptr[0] and ptr[1] twice. But third time ptr[1] is different even though it was freed before calling MALLOC. This is because memset corrupted the heap metadata due to which RTL cannot determine the actual size of ptr[1] when free CRTL is called. Free CRTL uses the corrupted size to add the packet to a wrong free list. This leads to an issue when this packet gets allocated for a different sized allocation request. Application may overwrite into this allocation assuming that it got an allocation what it requested for.

One of the helpful tools to find the root cause of these kind of heap corruption issues is eFence library. Relink the application with eFence library so that your application uses the heap allocation/deallocation routines provided by eFence library. Linking with this library increases the amount of dynamic memory allocated by an application.

Here is a demonstration of how eFence library detects heap corruption issues observed while running OVERFLOW.EXE

```
$ cc/list/machine_code overflow.c
$
$ link/map/full/cross_reference overflow,libefence.a/libr
%ILINK-W-MULDEF, symbol DECC$REALLOC multiply defined
file: SYS$COMMON:[SYSLIB]DECC$SHR.EXE;2
%ILINK-W-MULDEF, symbol DECC$MALLOC multiply defined
file: SYS$COMMON:[SYSLIB]DECC$SHR.EXE;2
%ILINK-W-MULDEF, symbol DECC$CALLOC multiply defined
file: SYS$COMMON:[SYSLIB]DECC$SHR.EXE;2
%ILINK-W-MULDEF, symbol DECC$FREE multiply defined
file: SYS$COMMON:[SYSLIB]DECC$SHR.EXE;2
$
```

Note: %ILINK-W-MULDEF warnings can be ignored. Ensure that in the MAP file created for your image CRTL allocation/reallocation routines (MALLOC/FREE) are resolved from libefence.a library.

```
$ run overflow.exe

Electric Fence 2.1 Copyright (C) 1987-1998 Bruce Perens.

ptr[0]=EFFE0    ptr[1]=f3fe0

ptr[0]=EFFE0    ptr[1]=f3fe0

%SYSTEM-F-ACCVIO, access violation, reason mask=04, virtual address=00000000000F0000,
PC=FFFFFFFF84236050, PS=0000001B

%TRACE-F-TRACEBACK, symbolic stack dump follows

image     module    routine              line      rel PC           abs PC

LIBOTS                                    0 0000000000012050 FFFFFFFF84236050

OVERFLOW  OVERFLOW  main                 4175 0000000000000282 0000000000030282

OVERFLOW  OVERFLOW  __main               4157 00000000000000B2 00000000000300B2

PTHREAD$RTL  THD_THREAD  thdBase       245262 0000000000005BF2 FFFFFFFF844D4E72

PTHREAD$RTL  THD_INIT  pthread_main    245041 00000000000006B2 FFFFFFFF8448A6B2

0 FFFFFFFF80A9BBD2 FFFFFFFF80A9BBD2

DCL                                       0 000000000007D032 000000007AE29032

%TRACE-I-END, end of TRACE stack dump
```

Mapping the PC reported by ACCVIO when running OVERFLOW.EXE linked with eFence library would map to the exact instruction that is corrupting the heap.

Using the freed pointer

Another heap corruption issue occurs when applications continue using the pointers that are already freed using free call. Attached text file demonstrates this behavior.

usefree.txt

Running program USERFREE.EXE results in ACCVIO because of the statement *ptr[0]=0x00BADBAD. This statement corrupted the free list maintained by heap management routines. Even though the ACCVIO reported by LIBRTL, application continued using the freed pointer is responsible for it.

Let's look at the process dump created while running USERFREE.EXE and figure out how to identify whether the RTL free list is corrupted.

```
$ analyze/process  USEFREE.DMP

OpenVMS I64 Debug64 Version V8.4-001

%DEBUG-I-NODSTS, no Debugger Symbol Table: no DSF file found and

-DEBUG-I-NODSTIMG, no symbols in DISK$DEBUG-
DISK1:[DKA1300.PRAMOD.TEST]USEFREE.EXE;5

%DEBUG-I-DYNIMGSET, setting image LIBRTL

%DEBUG-I-DYNMODSET, setting module LIB$MALLOC

%SYSTEM-F-ACCVIO, access violation, reason mask=00, virtual
address=0000000000BADBAD, PC=FFFFFFFF8406E2C0, PS=0000001B

break on unhandled exception at LIB$MALLOC\LIB$VM_MALLOC\%LINE 24936+224

%DEBUG-W-UNAOPNSRC, unable to open source file
$1$DGA7312:[LIBRTL_2.SRC]LIBMALLOC.C;1

-RMS-F-DEV, error in device name or inappropriate device type for operation

24936: Source line not available
```

```
DBG> set image librtl
DBG> set module lib$malloc
DBG> set radix hex
DBG> examine StaticZone.freeList
LIB$MALLOC\StaticZone..freeList[0:255]
[0]
flink:  0000000000000000
token:  0000000000000000
.....
[51]
flink:  0003000000BADBAD            → RTl Free list is corrupted
token:  0000000000000004
.....
```

Inside RTL, bits 48-63 of FLINK is used for some internal operation. Rest of the bits in the FLINK matches with that of the VA reported by ACCVIO. This symptom indicates that this problem is caused either because application is overwritten the earlier buffer or a pointer was used even after it was freed using the free call. If the problem is reproducible HP OpenVMS debugger can be used.

After it is determined that RTL free list is corrupted use the pattern that corrupted the free list for further analysis.
If it's plain text as the case in this dump, mostly likely corruption happened because of application overwriting an earlier MALLOC packet.

Now, let's relink USEFREE.OBJ with eFence library and see whether eFence library is able to catch the heap corruption.

```
$ cc/ list/machine_code usefree.c
$
$ link/map/full/cross_reference usefree,libefence.a/libr
%ILINK-W-MULDEF, symbol DECC$REALLOC multiply defined
file: SYS$COMMON:[SYSLIB]DECC$SHR.EXE;2
%ILINK-W-MULDEF, symbol DECC$MALLOC multiply defined
file: SYS$COMMON:[SYSLIB]DECC$SHR.EXE;2
%ILINK-W-MULDEF, symbol DECC$CALLOC multiply defined
file: SYS$COMMON:[SYSLIB]DECC$SHR.EXE;2
%ILINK-W-MULDEF, symbol DECC$FREE multiply defined
file: SYS$COMMON:[SYSLIB]DECC$SHR.EXE;2
$
$ run usefree

Electric Fence 2.1 Copyright (C) 1987-1998 Bruce Perens.
%SYSTEM-F-ACCVIO, access violation, reason mask=04, virtual
address=00000000000EFFE0, PC=0000000000030170, PS=0000001B
%TRACE-F-TRACEBACK, symbolic stack dump follows
image     module    routine              line      rel PC           abs PC
USEFREE  USEFREE  main                  3717 0000000000000170 0000000000030170
USEFREE  USEFREE  __main                3705 00000000000000B2 00000000000300B2
```

```
PTHREAD$RTL  THD_THREAD  thdBase        245262 0000000000005BF2 FFFFFFFF844D4E72
PTHREAD$RTL  THD_INIT  pthread_main     245041 00000000000006B2 FFFFFFFF8448A6B2
0 FFFFFFFF80A9BBD2 FFFFFFFF80A9BBD2
DCL                                          0 000000000007D032 000000007AE29032
%TRACE-I-END, end of TRACE stack dump
$
```

Yes, eFence library, in this case, is able to detect the heap corruption as soon it happens. If the PC reported by ACCVIO is mapped it should map to the exact instruction which causes the heap corruption.

Now let's look at how much HP OpenVMS debugger can help to root cause this problem. Since this problem is reproducible OpenVMS debugger can be used to get more closer to the heap corruption. First figure out the address which contains the pattern using the process dump. Use the OpenVMS debugger to set watch on that address and figure out when that pattern is written. For example ACCVIO reported while running USEFREE.EXE lead us to StaticZone..freeList[51] as shown earlier.

```
$ run USEFREE
OpenVMS I64 Debug64 Version V8.4-001
%DEBUG-I-INITIAL, Language: C, Module: USEFREE
%DEBUG-I-NOTATMAIN, Type GO to reach MAIN program
DBG> Go
break at routine USEFREE\main
3709: ptr[0]=(int *)malloc(size);
DBG> set image librtl                  →     Set the LIBRTL image and set a watch
on the free list which is going to be corrupted
DBG> set module lib$malloc             Since the problem is reproducible we are
expecting the same free list to get corrupted
DBG> set radix hex
DBG> set watch StaticZone.freeList[51]
%DEBUG-I-WPTTRACE, non-static watchpoint, tracing every instruction
DBG> Go
%DEBUG-I-DYNIMGSET, setting image USEFREE
watch of LIB$MALLOC\StaticZone..freeList[51].flink at USEFREE\main\%LINE 3711
3711: free(ptr[0]);
old value: 0000000000000000
new value: 00000000002DCC40
watch of LIB$MALLOC\StaticZone..freeList[51].token at USEFREE\main\%LINE 3711
3711: free(ptr[0]);
old value: 0000000000000000
new value: 0000000000000002
break at USEFREE\main\%LINE 3718
3718: ptr[0]=(int *)malloc(size);
DBG> Go
watch of LIB$MALLOC\StaticZone..freeList[51].flink at USEFREE\main\%LINE 3718
3718: ptr[0]=(int *)malloc(size);
old value: 00000000002DCC40
```

```
new value: 0000000000BADBAD →Debugger showing the corrupted value overwriting the
free list FLINK maintained
by RTL

watch of LIB$MALLOC\StaticZone..freeList[51].token at USEFREE\main\%LINE 3718

3718: ptr[0]=(int *)malloc(size);

old value: 0000000000000002

new value: 0000000000000003

break at USEFREE\main\%LINE 3719

3719: ptr[1]=(int *)malloc(size);

DBG> ex 00000000002DCC40

00000000002DCC40:       0000000000BADBAD

DBG>
```

Using this technique address of the corrupted packet can be identified. Looking at the data in this packet, application developers may be able to identify the packet and then relate to the code which works on this packet. With this technique root cause of corruption cannot be identified but would provide valuable inputs to find the root cause of the problem.

Incorrect usage of system calls

OpenVMS system calls that can create new address space, by default, deletes the existing address space. Some of the system calls allow an application to override this behavior. For example SYS$CRMPSC system service provides SEC$M_NO_OVERMAP flag that can be set when calling SYS$CRMPSC to override this behavior. So developers writing applications that uses RTLs to allocate from heap and uses system services that can allocate process virtual address space should read the documentation of those system services carefully. Application that uses both RTL and system calls may lead to issues as demonstrated in the text file attached below.

malloc crmpsc.txt

In program MALLOC_CRMPSC.C, SYS$CRMPSC system calls was not provided with SEC$M_NO_OVERMAP flag, so it can overwrite an existing address space when inadr specifies a nonzero address space. SYS$CRMPSC is passed with an inadr argument that specifies the address space to which SYS$CRMPSC created global section or process section needs to be mapped. In MALLOC_CRMPSC.C inadr argument specifies an address space that was freed using CRTL free call. Because SEC$M_NO_OVERMAP flag was not set, SYS$CRMPSC recreated the address space. Later when MALLOC was called, ACCVIO exception reported while RTL tried to update the heap management data. ACCVIO is reported in LIBRTL, but, this occurred because heap management data was corrupted when SYS$CRMPSC overlapped with an existing address space managed by heap management routines.

Now let's look at the process dump created while running MALLOC_CRMPSC.EXE. This program creates a global section and maps a file named IMP_SRV.DMP to that global section. So ensure that such a file is there in the current directory before running the program.

```
$ anal/proc MALLOC_CRMPSC.DMP

OpenVMS I64 Debug64 Version V8.4-001

%DEBUG-I-NODSTS, no Debugger Symbol Table: no DSF file found and

-DEBUG-I-NODSTIMG, no symbols in DISK$DEBUG-
DISK1:[DKA1300.PRAMOD.QXCM1000907583]MALLOC_CRMPSC.EXE;1

%DEBUG-I-DYNIMGSET, setting image LIBRTL

%DEBUG-I-DYNMODSET, setting module LIB$MALLOC

%SYSTEM-F-ACCVIO, access violation, reason mask=04, virtual
address=000000000007A796, PC=FFFFFFFF8406E6F0, PS=0000001B

break on unhandled exception at LIB$MALLOC\LIB$VM_MALLOC\%LINE 25028+15

%DEBUG-W-UNAOPNSRC, unable to open source file
$1$DGA7312:[LIBRTL_2.SRC]LIBMALLOC.C;1
```

```
-RMS-F-DEV, error in device name or inappropriate device type for operation

25028: Source line not available

DBG> set image librtl

DBG> set module lib$malloc

DBG> set rad hex

DBG> ex StaticZone.freeList
```

Here the **virtual address (VA)(0000000000007A796)** reported by ACCVIO **does not match** with any of the FLINK in the RTL maintained free list. So more likely free list is not corrupted here. So let's look at the process dump using system dump analyzer (SDA) and look at why accessing this VA caused an ACCVIO

```
$ analyze/crash MALLOC_CRMPSC.DMP

OpenVMS system dump analyzer

...analyzing an I64 compressed process dump...

Dump taken on 17-NOV-2012 13:23:18.93 using version V8.4


SDA> show page 000000000007A796

Page table

----------

Mapped Address      PTE Address           PTE        Type  AR/PL KESU MLO GH PgTyp
Loc          Bak       RefCnt  WSLX

----------------- ----------------- ----------------- ----- ----- ---- --- - -----
-- ------ ----------------- ---- --------

00000000.0007A000 000007FE.000001E8 00000B5B.C3060389 VALID 1 U RRRR --U 0 GLOBAL
ACTIVE 00000520.00010000 0001 00000000

SDA>
```

SDA show page information shows that VA is part of read only global page and is valid. As pages maintained by RTL should be writable from usermode, looks like something gone wrong. Clue here is that, page that caused ACCVIO is global, which tells us that this is part of a global section. So most likely a system call that creates the global section has overwritten the heap pages maintained by RTL. It could be possible that if the global section created had write permissions, ACCVIO reported would have been different.

Now how to identify that this global section is overwritten the heap managed by RTL. Here is where pattern maintained in the RTL header comes in handy. RTL puts a 0xFOOD pattern for allocated packets and a 0x7777 pattern for deallocated packets. Now search dump for the allocated/deallocated packet, and then check the size of the packet to see whether the VA reported in the ACCVIO is within the packet. Starting address of the search is the address after all the images are mapped. Use SDA "show proc/image=all" to see what is last P0 address mapped to an image.

```
SDA> SEARCH/MASK=FFFF0000 00052000:00094000  F00D0000

SDA> SEARCH/MASK=FFFF0000 00052000:00094000  77770000

Closest we get is

Match at 00000000.00078394    F00D0070

Looking at the header we figure out that the packet size is 0x2400.

SDA> show stack 00078390:00078394

Specified Stack Range

---------------------
```

```
00000000.00078390    F00D0070.00002400  CTL$C_CLIDATASZ+00698
-------------------------------------
-----------------------------Size of the MALLOC packet
---------------MALLOC pattern

SDA>

Adding the size to beginning of the packet leads us into the read only pages that
caused the ACCVIO.

SDA> evaluate 00078390 + 00002400

Hex = 00000000.0007A790   Decimal = 501648

SDA>
```

If the GLOBAL section is writable ACCVIO would have reported mostly due to free list corruption. But still this technique can be employed to identify whether a MALLOC packet is overwritten by a global section.

Using LIB$ calls and MALLOC

It's not an issue if an application decides to use both LIB$ calls and MALLOC for different allocations. Only problem with this approach is that heap management for LIB$ calls and MALLOC call is done separately. Free list maintained by LIB$ calls is not available for MALLOC and vice versa. This may cause LIB$GET_VM or MALLOC to fail even though there is free memory available in their free lists.

How to avoid heap corruption and troubleshooting

Coding practices

Following good coding practices help to avoid the issues causing heap corruption.

Don't overwrite the buffers allocated for the applications.

Don't continue using the buffers/pointers deallocated using free call.

In cases when a RTL routine and a system call that alters the process address space, needs to be used together, read the documentation carefully to avoid the address space overlap.

Process dump analysis

As shown already process dump analysis helps to understand the corruption and provides valuable inputs for further analysis. If customer application crashes in production process dump analysis would be the way to figure out what has caused corruption. There may be cases where analyzing process dump alone may not root cause the heap corruption, but dump analysis take us closer to the problem.

Using HP OpenVMS debugger

In cases where problem is reproducible OpenVMS debugger can be used to some extent as shown already. OpenVMS debugger also has a feature using which heap can be monitored. Refer section "Using the Heap Analyzer" in HP OpenVMS Debugger Manual (http://h71000.www7.hp.com/doc/84final/4538/BA554_90016.pdf) for more information.

Using eFence

One of the helpful tools to find the case of heap corruption issues is eFence (http://en.wikipedia.org/wiki/Electric_Fence). Ported version of this library for OpenVMS is available here (ftp://ftp.hp.com/pub/openvms/freeware/electricfence). To use this library, relink your application with eFence library. Linking with this library has a side effect of increase in the amount of dynamic memory allocated by an application.

Conclusion

Troubleshooting heap corruption issues are very difficult. Difficulty mainly arises because corruption is detected not when the corruption occurs but after some time or a long time after corruption occurs. Troubleshooting mechanisms mainly relies on looking at the pattern that caused ACCVIO and searching for those patterns in the dump. Then application developers can use their knowledge about the application to detect what could have caused corruption.

For more information

We advise the customer to go through this document whenever they face issues due to heap corruption. This document contains the necessary troubleshooting techniques required to analyze such issues. For further help contact HP support.

4AA4-4796ENW, February 2013