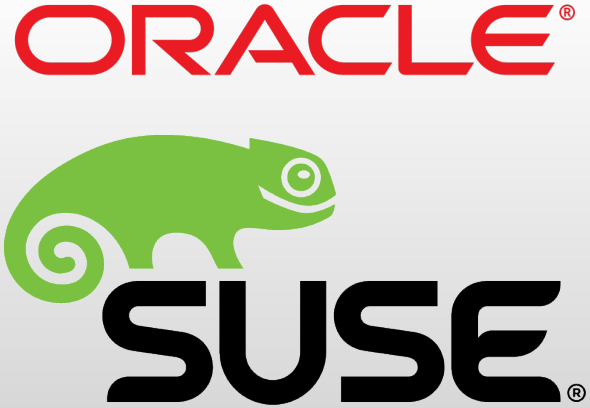
Memory Management Issues

Crash Dump Analysis 2015/2016

CHARLES UNIVERSITY IN PRAGUE

faculty of mathematics and physics

Department of Distributed and Dependable Systems



Overview

- **Slab allocator**
  - Kernel memory allocator
  - Internal debugging features
  - Support in mdb
- **Memory management errors**
- **libumem**

References

- **Jeff Bonwick: The Slab Allocator: *An Object-Caching Kernel Memory Allocator***
  - http://www.usenix.org/publications/library/proceedings/bos94/bonwick.html
- **Jeff Bonwick, Jonathan Adams: *Magazines and Vmem: Extending the Slab Allocator to Many CPUs and Arbitrary Resources***
  - http://www.usenix.org/event/usenix01/bonwick.html
- **Man pages**
  - libumem(3LIB), umem_debug(3MALLOC)

Slab allocator

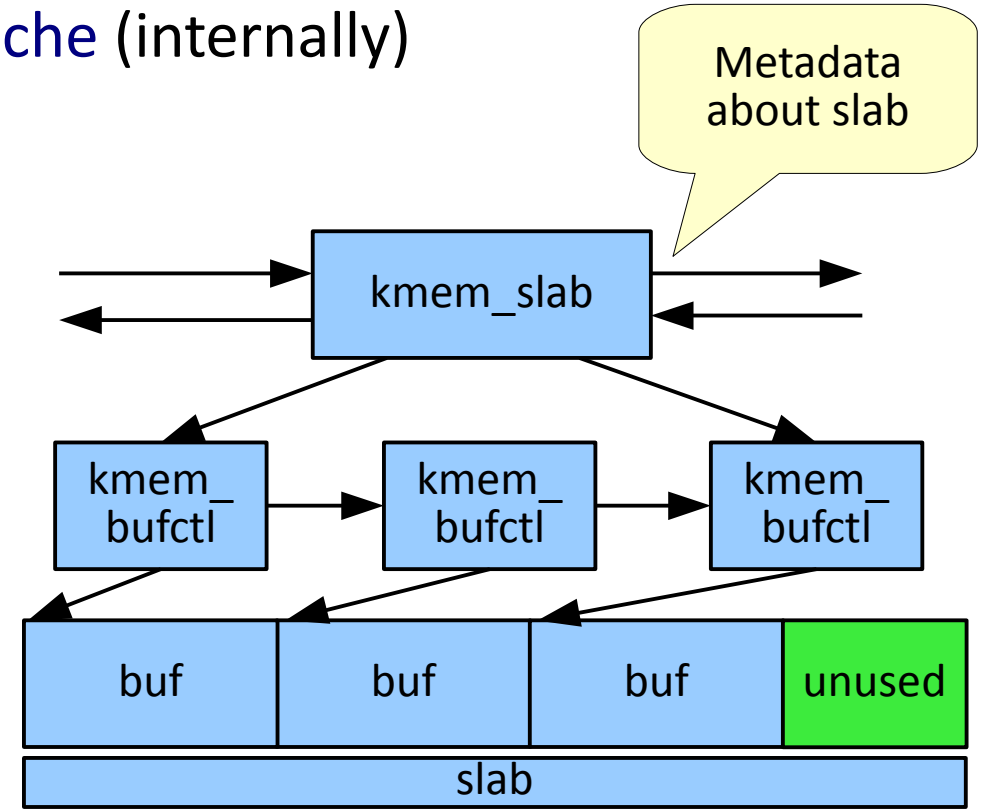
- **Object caching allocator**
  - API for dealing with objects which are frequently allocated / deallocated
    - Avoiding full initialization after each allocation
  - Also for non-caching allocations
    - Kernel version of malloc/free
      - `kmem_alloc(size)` – allocate a buffer
      - `kmem_free(bufp, size)` – release it

Slab allocator (2)

- **Cache**
  - Contains objects of same type/size
  - Several caches for different purposes
    - Object caches: `process_cache`, `thread_cache`, ...
      - Used for types such as `proc_t`, `kthread_t`, ...
    - Non-caching allocations use caches internally: `kmem_alloc_8`, `kmem_alloc_16`, ...
      - Suffix `_8` or `_16` is maximum allocation size
      - Best fit: `kmem_alloc(14)` allocates a buffer from the `kmem_alloc_16` cache (2 bytes wasted)
  - Cache consists of slabs

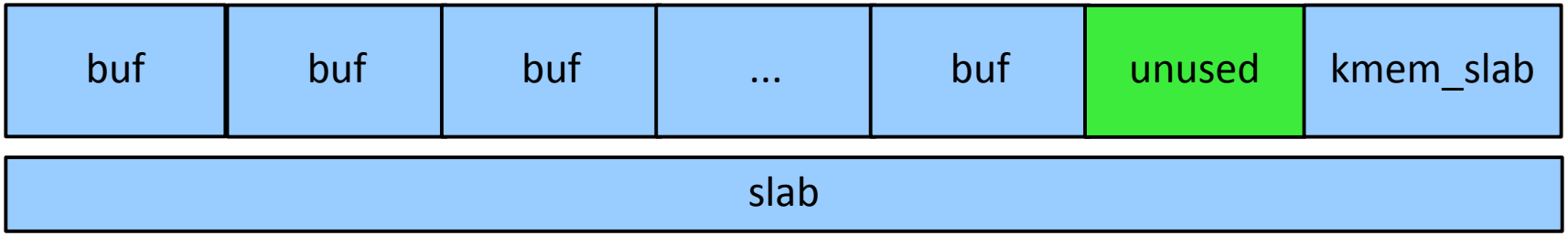
Slab allocator (3)

- **Slab**
  - Allocation unit for a cache (internally)
  - One or more pages
- **Buffer (buf)**
  - Raw memory area
    - Non-cached data
    - Object
- **Bufctl**
  - Buffer's metadata



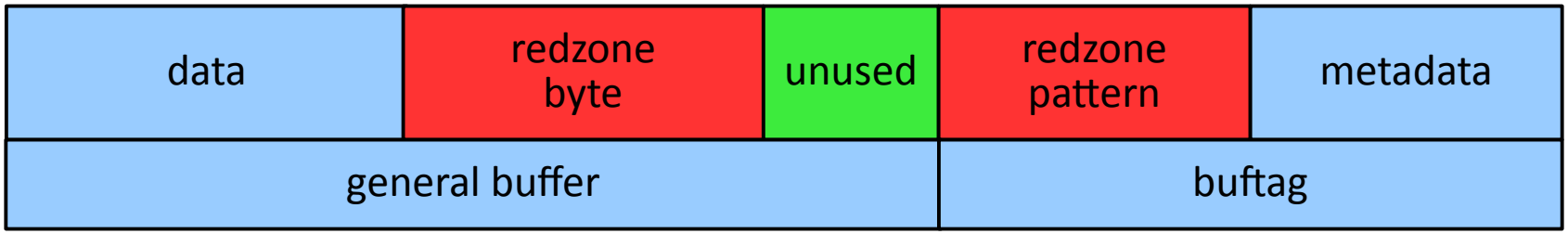
Slab allocator (4)

- **Small objects**
  - Size <= 1/8 page size
    - Slab metadata stored in the slab
    - kmem_bufctl allocated from the slab
      - Except for debugging mode



Buffer layout

- **Buftag**
  - Appended to each buffer depending on value of kmem_flags variable
    - Redzone pattern
    - Metadata (bufctl back pointer, stats)
- **Redzone byte**
  - Marks end of allocated user area
    - Used in case of non-caching allocations



Vmem allocator

- **Backing store for Slab allocator**
  - General-purpose large allocations
    - One or multiple pages

kmem_flags

- **Allocator debugging features**
  - Kernel variable
    - Add

      ```
      set kmem_flags = 0x0f
      ```

      to `/etc/system`
    - Bits for particular features
      - Defines in `/usr/include/sys/kmem_impl.h`

```
#define KMF_AUDIT       0x00000001      /* transaction auditing */
#define KMF_DEADBEEF    0x00000002      /* deadbeef checking */
#define KMF_REDZONE     0x00000004      /* redzone checking */
#define KMF_CONTENTS    0x00000008      /* freed-buffer content logging */
#define KMF_NOMAGAZINE  0x00000020      /* disable per-cpu magazines */
#define KMF_FIREWALL    0x00000040      /* put all bufs before unmapped pages */
```

Debugging patterns

- **Memory patterns as human-readable strings**
  - Free pattern `deadbeef`
  - Uninitialized pattern `baddcafe`
  - Redzone pattern `feedface`
  - Defines in `/usr/include/sys/kmem_impl.h`

```
#define KMEM_FREE_PATTERN              0xdeadbeefdeadbeefULL
#define KMEM_UNINITIALIZED_PATTERN     0xbaddcafebaddcafeULL
#define KMEM_REDZONE_PATTERN           0xfeedfacefeedfaceULL
#define KMEM_REDZONE_BYTE              0xbb
```

Memory management errors

- **Classification**
  - Multiple free
  - Buffer overrun
  - Use of uninitialized memory
  - Use after free
  - Memory leaks
  - Memory allocator specific
    - Freeing buffer to wrong cache
    - Freeing invalid pointer

Multiple free

- **Impact**
  - May corrupt heap
  - May free someone else's buffer

```
kernel memory allocator:
duplicate free: buffer freed twice
buffer=ffffff008ada0388  bufctl=ffffff008acf2710  cache: kmem_alloc_32
previous transaction on buffer ffffff008ada0388:
thread=ffffff00012fac80  time=T-0.000000267  slab=ffffff008ace8898  cache: kmem_alloc_32
kmem_cache_free_debug+12f
kmem_cache_free+53
kmem_free+1f7
bar+29
foo+e
taskq_thread+16
thread_start+8

panic[cpu0]/thread=ffffff00012fac80:
kernel heap corruption detected


ffffff00012fabf0 genunix:kmem_error+4a9 ()
ffffff00012fac10 genunix:kmem_free+209 ()
ffffff00012fac30 multif:bar+37 ()
ffffff00012fac40 multif:foo+e ()
ffffff00012fac60 multif:taskq_thread+16 ()
ffffff00012fac70 unix:thread_start+8 ()
```

```
void bar() {
     char * l = kmem_alloc(32, KM_SLEEP);
     kmem_free(l, 32);
     kmem_free(l, 32);
}

void foo() {
     bar();
}
```

Buffer overrun

- **Writing outside allocated buffer**
  - Usually past end of allocated space

```
kernel memory allocator:
redzone violation: write past end of buffer
buffer=ffffff00ba0fa660  bufctl=ffffff00ba724b78  cache: kmem_alloc_8
previous transaction on buffer ffffff00ba0fa660:
thread=ffffff000142fc80  time=T-0.000000514  slab=ffffff00b8ffbe30  cache: kmem_alloc_8
kmem_cache_alloc_debug+283
kmem_cache_alloc+aa
kmem_alloc+8c
foo+17
taskq_thread+16
thread_start+8

panic[cpu0]/thread=ffffff000142fc80:
kernel heap corruption detected


ffffff000142fc00 genunix:kmem_error+4a9 ()
ffffff000142fc20 genunix:kmem_free+21b ()
ffffff000142fc40 bufovr:foo+32 ()
ffffff000142fc60 bufovr:taskq_thread+16 ()
ffffff000142fc70 unix:thread_start+8 ()
```

```
void bar(char *c) {
     for (int i = 0; i < 10; i++)
          c[i] = i;
}

void foo() {
     char * l = kmem_alloc(8, KM_SLEEP);
     bar(l);
     kmem_free(l, 8);
}
```

Buffer overrun (2)

- **Impact**
  - User data → undefined behavior
  - Redzone, free patterns → detected by allocator (panic)
    - Allocator may detect corruption too late
      - Hidden root cause
        - The guilty thread not shown in panic stack
        - Need to analyze raw memory and find source buffer
      - Firewall
        - Each buffer allocated before an unmapped page
        - Slow, high address space fragmentation

Use of uninitialized memory

- **Properly allocated memory with uninitialized content**
  - Complex data structures, large number of members, partial initialization, etc.
  - Structure visible to other threads before proper initialization
  - Read values can trigger page faults
    - `baddcafe` value in registers

Use after free

- **Memory used/dereferenced after deallocation**
  - Race condition, broken reference count, etc.
    - First thread deallocates, second thread accesses afterwards
  - Read value can trigger page faults
    - `deadbeef` value in registers
    - Typically far away from original deallocation
  - Writes detected by allocator on next transaction

Use after free (2)

```
panic[cpu0]/thread=ffffff0001107c80:
BAD TRAP: type=d (#gp General protection) rp=ffffff0001107b20 addr=0


sched:
#gp General protection
pid=0, pc=0xfffffffffb9dfb5a, sp=0xffffff0001107c10, eflags=0x10286
cr0: 8005003b<pg,wp,ne,et,ts,mp,pe> cr4: 6b8<xmme,fxsr,pge,pae,pse,de>
cr2: fffffd7fff228f06
cr3: 3400000
cr8: c

        rdi: deadbeefdeadbeef rsi:               10 rdx:               10
        rcx:                 f  r8:                8  r9: deadbeefdeadbeef
        rax: deadbeefdeadbeef rbx:                0 rbp: ffffff0001107c20
        r10: ffffff00840257a0 r11: deadbeefdeadbeff r12:                0
        r13:                 0 r14:                0 r15:                0
        fsb: fffffd7fff190200 gsb: ffffffffbc2bc70  ds:                0
         es:                 0  fs:                0  gs:                0
        trp:                 d err:                0 rip: fffffffffb9dfb5a
         cs:                30 rfl:            10286 rsp: ffffff0001107c10

ffffff0001107a00 unix:die+10f ()
ffffff0001107b10 unix:trap+43e ()
ffffff0001107b20 unix:_cmntrap+e9 ()
ffffff0001107c20 genunix:kmem_free+72 ()
ffffff0001107c40 ll:free_list+21 ()
ffffff0001107c60 ll:taskq_thread+23 ()
ffffff0001107c70 unix:thread_start+8 ()
```

```
struct ll {
     struct ll *ll_next;
     clock_t ll_lbolt;
};

...

free_list(struct ll *l) {
     while (l != NULL) {
          kmem_free(l, sizeof(*l));
          l = l->ll_next;
     }
}
```

mdb support

- **::kmastat**
  - Dump kernel memory allocator statistics
    - Caches, vmem segments

```
cache                       buf    buf    buf    memory     alloc alloc
name                       size in use  total    in use   succeed  fail
------------------------- ------ ------ ------ ---------- --------- -----
kmem_magazine_1              16   1531  13554    221184B   2252528     0
kmem_magazine_3              32    454   7000    229376B   1653782     0
kmem_magazine_7              64   2065  14322    946176B   5149186     0
kmem_magazine_15            128    701  10447   1380352B   1912460     0
...
kmem_alloc_8                  8  48560  51306    417792B   2089369     0
kmem_alloc_16                16  29840  30371    495616B   3081470     0
kmem_alloc_24                24  23454  25718    630784B    204970     0
kmem_alloc_32                32  16313  17000    557056B   3244728     0
kmem_alloc_40                40  24135  26400   1081344B    734315     0
kmem_alloc_48                48   8828   8881    438272B     90683     0
```

mdb support (2)

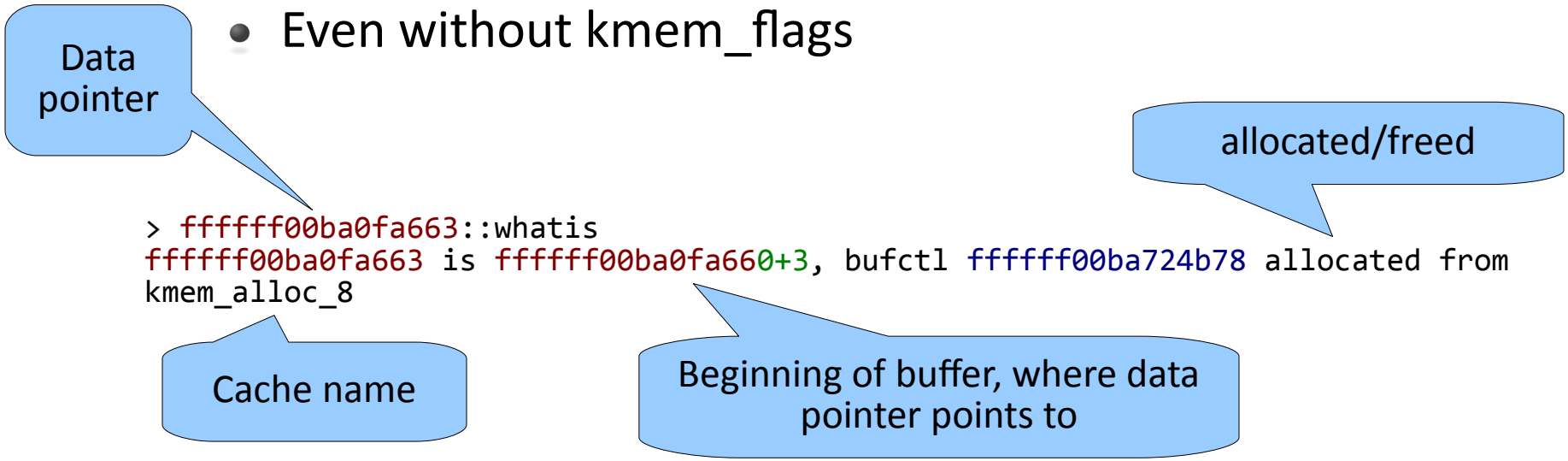
- **[addr]::kmem_cache**
  - Similar to ::kmastat
    - Print cache address (e.g. for ::walk kmem_slab)

```
ADDR             NAME                    FLAG  CFLAG  BUFSIZE  BUFTOTL
ffffff0209021020 kmem_magazine_1         0000 080000       16    13554
ffffff02090212e0 kmem_magazine_3         0000 080000       32     7000
ffffff02090215a0 kmem_magazine_7         0000 080000       64    14322
ffffff0209021860 kmem_magazine_15        0000 080000      128    10447
ffffff0209021b20 kmem_magazine_31        0000 080000      256     1395
ffffff0209022020 kmem_magazine_47        0000 080000      384      160
ffffff02090222e0 kmem_magazine_63        0000 080000      512      280
ffffff02090225a0 kmem_magazine_95        0000 080000      768       85
ffffff0209022860 kmem_magazine_143       0000 080000     1152      210
ffffff0209022b20 kmem_slab_cache         0000 080000       72   347655

...
```

mdb support (3)

- **::whatis**
  - Find buffer metadata for given address (bufctl or vmem segment)
    - Where the address belongs to
      - E.g. which cache, vmem, buffer, thread stack
    - Even without kmem_flags



mdb support (4)

- **addr::bufctl [-v]**
  - Show thread stack trace, timestamp, ...

```
> ffffff00b58bfa79::whatis
ffffff00b58bfa79 is ffffff00b58bfa78+1, bufctl ffffff00b7c222d0 allocated from
kmem_alloc_16
> ffffff00b7c222d0::bufctl -v
            ADDR          BUFADDR        TIMESTAMP           THREAD
                            CACHE          LASTLOG         CONTENTS
ffffff00b7c222d0 ffffff00b58bfa78     115bfc332611 ffffff000136dc80
                 ffffff00840257a0 ffffff00844ebd00 ffffff0084e91060
                 kmem_cache_alloc_debug+0x283
                 kmem_cache_alloc+0x16d
                 kmem_alloc+0x8c
                 alloc_list+0x24
                 taskq_thread+0x16
                 thread_start+8
```

Thread stack trace at time of last operation. Function kmem_alloc confirms that the last operation was allocation.

mdb support (5)

- **[addr]::kmem_verify**
  - Find corrupted buffers in cache(s)

```
> ::kmem_verify
Cache Name                          Addr               Cache Integrity
kmem_magazine_1                     ffffff0084021020 clean
kmem_magazine_3                     ffffff00840212a0 clean
kmem_magazine_7                     ffffff0084021520 clean
...
kmem_bufctl_audit_cache             ffffff0084022ca0 clean
kmem_alloc_8                        ffffff0084025520 1 corrupt buffer
kmem_alloc_16                       ffffff00840257a0 clean
...

> ffffff0084025520::kmem_verify
Summary for cache 'kmem_alloc_8'
  buffer ffffff00ba0fa660 (allocated) has a corrupt redzone size encoding
```

Check all caches

Details for one cache

mdb support (6)

- **[addr]::kmalog**
  - Display past transactions on buffer

```
> ffffff00ba0fa660::kmalog

T-0.000000000  addr=ffffff00ba0fa660  kmem_alloc_8
          kmem_cache_alloc_debug+0x283
          kmem_cache_alloc+0xaa
          kmem_alloc+0x8c
          foo+0x17
          taskq_thread+0x16
          thread_start+8

T-0.000409495  addr=ffffff00ba0fa660  kmem_alloc_8
          kmem_cache_free_debug+0x12f
          kmem_cache_free+0x53
          kmem_free+0x1f7
          kobj_free+0x27
          get_progbits+0x4fa
          kobj_load_module+0x27e
          modload_thread+0x77
          thread_start+8

...
```

mdb support (7)

- **::findleaks**
  - Garbage collection to find memory leaks
    - Use walker (::walk leak) to iterate over bufctls of individual leaked buffers

```
> ::findleaks
CACHE             LEAKED           BUFCTL CALLER
ffffff00840257a0      19 ffffff00b9b1ab38 alloc_list+0x24
ffffff0084032520       1 ffffff009753ba80 allocb+0x64
ffffff0084032020       1 ffffff009ce75d00 dblk_constructor+0x3b
------------------------------------------------------------------------
           Total      21 buffers, 560 bytes
```

mdb support (8)

- ::memstat
  - Basic memory usage overview

```
> ::memstat
Page Summary                Pages                MB  %Tot
------------     ----------------  ----------------  ----
Kernel                     836170              3266   65%
Anon                       364132              1422   28%
Exec and libs               30456               118    2%
Page cache                  18522                72    1%
Free (cachelist)            10550                41    1%
Free (freelist)             32038               125    2%

Total                     1291868              5046
Physical                  1291867              5046
```

mdb support (9)

- **::kmausers [-ef]**
  - Display the largest users of the kmem allocator, sorted by stack trace
    - -e Include all users, not just the largest
    - -f Display individual allocations

```
> ::kmausers
25165824 bytes for 6144 allocations with data size 4096:
         kmem_cache_alloc_debug+0x283
         kmem_cache_alloc+0xaa
         kalloca+0x11a
         i_ddi_mem_alloc+0x173
         ddi_dma_mem_alloc+0x10a
         e1000g_alloc_dma_buffer+0x9b
         e1000g_alloc_rx_sw_packet+0x62
         e1000g_alloc_rx_packets+0xc7
         e1000g_alloc_packets+0x86
         e1000g_alloc_dma_resources+0x6d
         e1000g_start+0x58
         e1000g_m_start+0x1a
...
```

User space allocators

- **Standard C library malloc**
- **Memory management debuggers**
  - Dmalloc
  - Electric Fence
  - libumem
  - AddressSanitizer
  - Purify
  - Valgrind

Debugging techniques

- **Debugging allocator**
  - Override default (libc) allocator entry points
  - Re-define malloc(), free(), etc. to analogous functions with debugging support
    - Compile-time (using #define)
    - Run-time (using LD_PRELOAD)
- **Simulated execution**
  - Tracking of allocations and all memory accesses

Debugging techniques

- **Hardware-assisted memory access checking**
  - HW feature specifically designed for memory debugging
  - Offload checks to hardware
  - Depends on HW support
    - IBM Storage Keys (POWER6+)
    - Intel MPX (Skylake+)
    - ADP (a.k.a. Oracle SSM, SPARC M7+)
  - Each method has its tradeoffs
  - May need OS, run-time, compiler support

libumem

- **User space memory management library**
  - Port of Slab allocator
  - Bundled in Solaris distributions
  - Debugging features
    - Memory leaks
    - Buffer overruns
    - Multiple frees
    - Use of uninitialized data, use of freed data
    - Logging of memory transaction history

libumem (2)

- **Detection of memory corruption**
  - Abort on memory error
    - Core dump, error description, stack trace
  - Shared library
    - Dynamic preload, no (re)compilation required
    - Enable/disable checks via environment variables
  - mdb support
    - Similar dcmds as for kernel (beginning with 'u')
      - ::umem_verify, ::umem_log, ::umalog, ::umastat

libumem (3)

- **UMEM_DEBUG**
  - Memory corruption detection
    - What information is tracked
    - Values:
      - audit[=frames] – save stack trace, time-stamp
      - contents[=count] – save content before free
      - guards – use special patterns
      - verbose – print errors to stderr
      - default – same as audit,contents,guards
      - firewall=[size] – allocations >= size followed by unmapped page

libumem (4)

- **UMEM_LOGGING**
  - Memory transaction logging
    - Values:
      - transaction[=size] – record allocation/free
      - contents[=size] – record contents
      - fail[=size] – record failed allocation

libumem (5)

- **UMEM_OPTIONS**
  - "Undocumented"
  - Memory back-end
  - Values:
    - backend=sbrk – default
    - backend=mmap – required for firewall