Improving MemGuard Support for UMA on FreeBSD

Chang-Hsien Tsai (luke)

vBSDcon 2015

Outline

- Kernel memory error
- INVARIANTS, RedZone, MemGuard
- UMA (zone allocator)
- Enhancement on MemGuard
- Bugs found

Memory Error

- Read-before-initialization (INVARIANTS)
- Use-after-free
  - write-after-free (INVARIANTS)
  - read-after-free
- Memory-overrun
  - write-overrun (RedZone)
  - read-overrun

INVARIANTS detection

- Added by jeff in 2002
- When free()
  - overwrite the memory with 0xdeadc0de
- When malloc()
  - check the memory with 0xdeadc0de

RedZone

- Added by pjd in 2006
- When malloc()
  - add 16 bytes of 0x42 before and after data
- When free()
  - check with the 0x42

MemoryGuard

- 2005/01/21 by bmilekic
  - when free(), vm_map_protect() sets read-only
- 2010/8/11 by mdf
  - when malloc(), vm_map_findspace() find KVA, kmem_back() allocate physical address
  - option to add addition one page before and after data
  - when free(), vm_map_delete() free physical address
- 2011/10/12 by glebius
  - support UMA

UMA

```
uma_zone_t
uma_zcreate(const char *name, size_t size, uma_ctor ctor, uma_dtor dtor,
        uma_init uminit, uma_fini fini, int align, uint32_t flags)
```

- VM ---> UMA ---> uma_zalloc ()

  uminit() ctor()

- VM <--- UMA <--- uma_zfree ()

  fini() dtor()

ENHANCEMENT ON MEMGUARD

Support UMA init/fini function

- Problem
  - zone->init() and zone->fini() are not UMA init/fini function
- Solution
  - call keg->init() and keg->fini() instead

UMA_ZONE_PCPU

- Problem
  - allocated size does not include struct pcpu
- Solution
  - allocated size + sizeof(struct pcpu) * mp_ncpus

realloc() with M_WAITOK

- Problem
  - when realloc(addr, size, type, M_WAITOK) memory allocated by memguard
  - if memguard_alloc() fails, return NULL
- Solution
  - fall back to normal malloc()

Lock Already Init

- Problem
  - lock_init() on UMA init()
  - assert fail “lock already initialized”
- Solution
  - bzero() memory before lock_init()
  - lock_init() with flag LO_NEW
    - MTX_NEW, RM_NEW, RW_NEW, SX_NEW,

Unsupported Zone

- UMA_ZONE_REFCNT
  - use the same union in struct vm_page
- UMA_ZFLAG_BUCKET, UMA_ZONE_VM
  - recursively allocation

BUGS FOUND

pipe_dtor()

```
377 void
378 pipe_dtor(struct pipe *dpipe)
379 {
380         ino_t ino;
381
382         ino = dpipe->pipe_ino;
383         funsetown(&dpipe->pipe_sigio);
384         pipeclose(dpipe);
385         if (dpipe->pipe_state & PIPE_NAMED) {
386                 dpipe = dpipe->pipe_peer;
387                 funsetown(&dpipe->pipe_sigio);
388                 pipeclose(dpipe);
389         }
```

https://bugs.freebsd.org/bugzilla/show_bug.cgi?id=197246

g_eli_auth_run(), g_eli_crypto_run()

```
521                 crp->crp_etype = 0;
522                 err = crypto_dispatch(crp);
523                 if (err != 0 && error == 0)
524                         error = err;
525         }
526         if (bp->bio_error == 0)
527                 bp->bio_error = error;
```

https://bugs.freebsd.org/bugzilla/show_bug.cgi?id=199705

```
198 static int
199 thread_init(void *mem, int size, int flags)
200 {
201         struct thread *td;
202
203         td = (struct thread *)mem;
204
205         td->td_sleepqueue = sleepq_alloc();
206         td->td_turnstile = turnstile_alloc();
207         td->td_rlqe = NULL;
208         EVENTHANDLER_INVOKE(thread_init, td);
209         td->td_sched = (struct td_sched *)&td[1];
210         umtx_thread_init(td);
211         td->td_kstack = 0;
212         return (0);
213 }
```

```
1823 static void
1824 seltdinit(struct thread *td)
1825 {
1826         struct seltd *stp;
1827
1828         if ((stp = td->td_sel) != NULL)
1829                 goto out;
1830         td->td_sel = stp = malloc(sizeof(*stp), M_SELECT, M_WAITOK|M_ZERO);
1831         mtx_init(&stp->st_mtx, "sellck", NULL, MTX_DEF);
1832         cv_init(&stp->st_wait, "select");
1833 out:
1834         stp->st_flags = 0;
1835         STAILQ_INIT(&stp->st_selq);
1836 }
```

https://bugs.freebsd.org/bugzilla/show_bug.cgi?id=199518

```
209 static int
210 sa_cache_constructor(void *buf, void *unused, int kmflag)
211 {
212         sa_handle_t *hdl = buf;
213
214         mutex_init(&hdl->sa_lock, NULL, MUTEX_DEFAULT, NULL);
215         return (0);
216 }
```

```
1387                 handle = kmem_cache_alloc(sa_cache, KM_SLEEP);
1388                 handle->sa_userp = userp;
1389                 handle->sa_bonus = db;
1390                 handle->sa_os = os;
1391                 handle->sa_spill = NULL;
1392                 handle->sa_bonus_tab = NULL;
1393                 handle->sa_spill_tab = NULL;
1394
1395                 error = sa_build_index(handle, SA_BONUS);
1396
1397                 if (hdl_type == SA_HDL_SHARED) {
1398                         dmu_buf_init_user(&handle->sa_dbu, sa_evict, NULL);
```

```
561 inline void
562 dmu_buf_init_user(dmu_buf_user_t *dbu, dmu_buf_evict_func_t *evict_func,
563     dmu_buf_t **clear_on_evict_dbufp)
564 {
565         ASSERT(dbu->dbu_evict_func == NULL);
```

https://bugs.freebsd.org/bugzilla/show_bug.cgi?id=202358

Conclusion

- MemGuard is effective on dynamic detection of memory error
- Good for use when developing and testing
- Need more test cases to expand coverage