

# 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](https://bugs.freebsd.org/bugzilla/show_bug.cgi?id=197246)

# g\_el\_i\_auth\_run(), g\_el\_i\_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](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](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 }
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
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);
```

# Conclusion

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