

```

*****
39944 Mon Jun 1 15:24:25 2015
new/usr/src/uts/common/io/e1000g/e1000g_alloc.c
XXXX e1000g use after free on start failure
*****
_____unchanged_portion_omitted_____

1435 static void
1436 e1000g_free_rx_packets(e1000g_rx_data_t *rx_data, boolean_t full_release)
1437 {
1438     p_rx_sw_packet_t packet, next_packet;
1439     uint32_t ref_cnt;

1441     mutex_enter(&e1000g_rx_detach_lock);

1443     packet = rx_data->packet_area;
1444     while (packet != NULL) {
1445         next_packet = packet->next;

1447         ref_cnt = atomic_dec_32_nv(&packet->ref_cnt);
1448         if (ref_cnt > 0) {
1449             atomic_inc_32(&rx_data->pending_count);
1450             atomic_inc_32(&e1000g_mblks_pending);
1451         } else {
1452             e1000g_free_rx_sw_packet(packet, full_release);
1453         }

1455         packet = next_packet;
1456     }

1458     if (full_release)
1459         rx_data->packet_area = NULL;

1461 #endif /* ! codereview */
1462     mutex_exit(&e1000g_rx_detach_lock);
1463 }

1466 static void
1467 e1000g_free_tx_packets(e1000g_tx_ring_t *tx_ring)
1468 {
1469     int j;
1470     struct e1000g *Adapter;
1471     p_tx_sw_packet_t packet;
1472     dma_buffer_t *tx_buf;

1474     Adapter = tx_ring->adapter;

1476     for (j = 0, packet = tx_ring->packet_area;
1477         j < Adapter->tx_freelist_num; j++, packet++) {

1479         if (packet == NULL)
1480             break;

1482         /* Free the Tx DMA handle for dynamical binding */
1483         if (packet->tx_dma_handle != NULL) {
1484             switch (packet->dma_type) {
1485 #ifdef __sparc
1486                 case USE_DVMA:
1487                     dvma_release(packet->tx_dma_handle);
1488                     break;
1489 #endif
1490                 case USE_DMA:
1491                     ddi_dma_free_handle(&packet->tx_dma_handle);
1492                     break;
1493                 default:

```

```

1494             ASSERT(B_FALSE);
1495             break;
1496         }
1497         packet->tx_dma_handle = NULL;
1498     } else {
1499         /*
1500          * If the dma handle is NULL, then we don't
1501          * need to check the packets left. For they
1502          * have not been initialized or have been freed.
1503          */
1504         break;
1505     }

1507     tx_buf = packet->tx_buf;

1509     switch (packet->dma_type) {
1510 #ifdef __sparc
1511     case USE_DVMA:
1512         e1000g_free_dvma_buffer(tx_buf);
1513         break;
1514 #endif
1515     case USE_DMA:
1516         e1000g_free_dma_buffer(tx_buf);
1517         break;
1518     default:
1519         ASSERT(B_FALSE);
1520         break;
1521     }

1523     packet->dma_type = USE_NONE;
1524 }
1525 if (tx_ring->packet_area != NULL) {
1526     kmem_free(tx_ring->packet_area, TX_SW_PKT_AREA_SZ);
1527     tx_ring->packet_area = NULL;
1528 }
1529 }

1531 /*
1532 * e1000g_release_dma_resources - release allocated DMA resources
1533 *
1534 * This function releases any pending buffers that has been
1535 * previously allocated
1536 */
1537 void
1538 e1000g_release_dma_resources(struct e1000g *Adapter)
1539 {
1540     e1000g_free_descriptors(Adapter);
1541     e1000g_free_packets(Adapter);
1542 }

1544 /* ARGSUSED */
1545 void
1546 e1000g_set_fma_flags(int dma_flag)
1547 {
1548     if (dma_flag) {
1549         e1000g_tx_dma_attr.dma_attr_flags = DDI_DMA_FLAGERR;
1550         e1000g_buf_dma_attr.dma_attr_flags = DDI_DMA_FLAGERR;
1551         e1000g_desc_dma_attr.dma_attr_flags = DDI_DMA_FLAGERR;
1552     } else {
1553         e1000g_tx_dma_attr.dma_attr_flags = 0;
1554         e1000g_buf_dma_attr.dma_attr_flags = 0;
1555         e1000g_desc_dma_attr.dma_attr_flags = 0;
1556     }
1557 }

```