OpenvSwitch LLDP Vulnerabilities

> Found and Reported by: Qian Chen (@cq674350529) from Codesafe Team of Legendsec at QI-ANXIN Group

Two vulnerabilities have been found in the lldp module when doing code review in ovs repository. The related code locates in https://github.com/openvswitch/ovs/blob/master/lib/lldp. The `lldp_decode()` in `lldp.c` is responsible for handling the received lldp packets. And two vulnerabilities have been found in this function.

> PS: I did't have real environment to test these issues, but I'm pretty sure the code is vulnerable through code review.

Out-of-Bounds Read in Organization Specific TLV

In `lldp_decode()` function, each TLV will be read and parsed in a while loop. Firstly, two bytes will read from the packet at `(1)`, then be parsed into `tlv_type` and `tlv_size` filed. If `tlv_type` is `0x7f` and `tlv_length` is not `0x0`, then the routine will go to `(2)`, where `tlv_size` will be checked. If it satisfy, `orgid` will be read at `(3)`, and `tlv_subtype` will be read at `(4)`. If `orgid` matches `avaya_oid`, and `tlv_subtype` equals to `LLDP_TLV_AA_ELEMENT_SUBTYPE`, then the routine will reach `(5)`. At `(5)`, another 32 bytes will be read. However, there is only a validation at `(2)` to guarantee `tlv_size >= 4`, if we craft a `Organization Specific TLV(type=0x7f)` with `tlv_length=5, org_unique_code='\x00\x04\x0d', org_subtype=11`, we can cause out-of-bounds read at `(5)` and later.

By the way, we can leave the crafted `Organization Specific TLV(type=0x7f)` at the end of lldp packet, just before the `End of LLDPDU TLV`.

```c
int
lldp_decode(struct lldpd *cfg OVS_UNUSED, char *frame, int s,
            struct lldpd_hardware *hardware, struct lldpd_chassis **newchassis,
            struct lldpd_port **newport)
{
    // ...
    while (length && !gotend) {
        if (length < 2) {
            VLOG_WARN("tlv header too short received on %s",
                      hardware->h_ifname);
            goto malformed;
        }
        tlv_size = PEEK_UINT16;     // (1)
        tlv_type = tlv_size >> 9;
        tlv_size = tlv_size & 0x1ff;
        // ...
        switch (tlv_type) {
        // ...
        case LLDP_TLV_ORG:
            CHECK_TLV_SIZE(1 + sizeof orgid, "Organisational");     // (2) ensure tlv_size >= 4
            PEEK_BYTES(orgid, sizeof orgid);                 // (3)
            tlv_subtype = PEEK_UINT8;   // (4)
            if (memcmp(dot1, orgid, sizeof orgid) == 0) {
                hardware->h_rx_unrecognized_cnt++;
            } else if (memcmp(dot3, orgid, sizeof orgid) == 0) {
                hardware->h_rx_unrecognized_cnt++;
            } else if (memcmp(med, orgid, sizeof orgid) == 0) {
                /* LLDP-MED */
                hardware->h_rx_unrecognized_cnt++;
            } else if (memcmp(avaya_oid, orgid, sizeof orgid) == 0) {
                u_int32_t aa_element_dword;
                u_int16_t aa_system_id_word;
                u_int16_t aa_status_vlan_word;
                u_int8_t aa_element_state;
                unsigned short num_mappings;

                switch(tlv_subtype) {
                case LLDP_TLV_AA_ELEMENT_SUBTYPE:
                    PEEK_BYTES(&msg_auth_digest, sizeof msg_auth_digest); // (5) out-of-bounds read

                    aa_element_dword = PEEK_UINT32; // out-of-bounds read
                    // ...
```

Integer Underflow in Organization Specific TLV

Similar to above, If `orgid` matches `avaya_oid`, and `tlv_subtype` equals to `LLDP_TLV_AA_ISID_VLAN_ASGNS_SUBTYPE`, the routine will go to `(6)`, where another 32 bytes will be read. At `(7)`, `tlv_size - 4 - LLDP_TLV_AA_ISID_VLAN_DIGEST_LENGTH` is calculated and assigned to `num_mappings`, then there is a check against `num_mappings` at `(8)`. Again, since there is no guarantee that `tlv_size` of `tlv_type=0x7f` must exceed `40`, we can craft a `Organization Specific TLV(type=0x7f, org_unique_code='\x00\x04\0xd', org_subtype=12)` with a small `tlv_length` to bypass the check at `(8)`, while cause integer underflow at `(7)`. Later, `num_mappings` will be used as a loop condition at `(9)`. As a result, by causing integer underflow at `(7)`, it may cause out-of-bounds read or segmentation fault in the for-loop later.

```
            case LLDP_TLV_ORG:
            CHECK_TLV_SIZE(1 + sizeof orgid, "Organisational");
            PEEK_BYTES(orgid, sizeof orgid);
            tlv_subtype = PEEK_UINT8;
            if (memcmp(dot1, orgid, sizeof orgid) == 0) {
                hardware->h_rx_unrecognized_cnt++;
            } else if (memcmp(dot3, orgid, sizeof orgid) == 0) {
                hardware->h_rx_unrecognized_cnt++;
            } else if (memcmp(med, orgid, sizeof orgid) == 0) {
                /* LLDP-MED */
                hardware->h_rx_unrecognized_cnt++;
            } else if (memcmp(avaya_oid, orgid, sizeof orgid) == 0) {
                u_int32_t aa_element_dword;
                u_int16_t aa_system_id_word;
                u_int16_t aa_status_vlan_word;
                u_int8_t aa_element_state;
                unsigned short num_mappings;

                switch(tlv_subtype) {
```

```
                case LLDP_TLV_AA_ELEMENT_SUBTYPE:
                    // ...
                    break;

                case LLDP_TLV_AA_ISID_VLAN_ASGNS_SUBTYPE:
                    PEEK_BYTES(&msg_auth_digest, sizeof msg_auth_digest); // (6) out-of-bounds read

                    /* Subtract off tlv type and length (2Bytes) + OUI (3B) +
                     * Subtype (1B) + MSG DIGEST (32B).
                     */
                    num_mappings = tlv_size - 4 -
                        LLDP_TLV_AA_ISID_VLAN_DIGEST_LENGTH;    // (7) integer underflow
                    if (num_mappings % 5 != 0) {                // (8)  can be bypassed with tlv_length=5
                        VLOG_INFO("malformed vlan-isid mappings tlv received");
                        goto malformed;
                    }

                    num_mappings /= 5; /* Each mapping is 5 Bytes */
                    for(; num_mappings > 0; num_mappings--) {   // (9)
                        uint8_t isid[3];

                        isid_vlan_map = xzalloc(sizeof *isid_vlan_map);
                        aa_status_vlan_word = PEEK_UINT16;

                        /* Status is first 4 most-significant bits. */
                        isid_vlan_map->isid_vlan_data.status =
                            aa_status_vlan_word >> 12;

                        /* Vlan is last 12 bits */
                        isid_vlan_map->isid_vlan_data.vlan =
                            aa_status_vlan_word & 0x0FFF;
                        PEEK_BYTES(isid, 3);
                        isid_vlan_map->isid_vlan_data.isid =
                            (isid[0] << 16) | (isid[1] << 8) | isid[2];
                        ovs_list_push_back(&port->p_isid_vlan_maps,
                                       &isid_vlan_map->m_entries);
                        isid_vlan_map = NULL;
                    }
                    break;
```