Vulnerabilities Details

Written by Wenqiang Li

Last updated: Jul 25, 2021

Vulnerabilities of Mbed OS MQTT

- Version
  - Nov 2, 2017
- Related Link
  - https://os.mbed.com/teams/mqtt/
  - https://github.com/eclipse/paho.mqtt.embedded-c

Bug 1 (Fixed)

Type

Doniel of Service

Description

The MQTT library is used to receive, parse and send mqtt packet between a broker and a client. The function *readMQTTLenString()* is called by the function *MQTTDeserialize_publish()* to get the length and content of the MQTT topic name. It parses the MQTT input linearly. Once a type-length-value tuple is parsed, the index is increased correspondingly. The maximum index is restricted by the length of the received packet size, as shown in line 4 of the code snippet below.

```
int readMQTTLenString(MQTTString* mqttstring, unsigned char** pptr, unsigned char* enddata)
{
    ...
    if (&(*pptr)[mqttstring->lenstring.len] <= enddata)
    {
        mqttstring->lenstring.data = (char*)*pptr;
        *pptr += mqttstring->lenstring.len;
        rc = 1;
    }
    ...
}
```

Note that *mqttstring->lenstring.len* is a part of user input, which can be manipulated. An attacker can simply change it to a larger value to invalidate the if statement so that the statements from line 6 to 8 are skipped, leaving the value of *mqttstring->lenstring.data* default to zero. Later, the value of *mqttstring->lenstring.data* is assigned to *curn* (line 4 of the code snippet below), which is zero under the attack. In line 9, **curn* is accessed. In an ARM cortex-M chip, the value at address 0x0 is actually the initialization value for the MSP register. It is highly dependent on the actual firmware. Therefore, the behavior of the program is unpredictable from this time on.

```
bool MQTT::Client<Network, Timer, a, b>::isTopicMatched(char* topicFilter, MQTTString& topicName)
{
   char* curf = topicFilter;
   char* curn = topicName.lenstring.data;
   char* curn_end = curn + topicName.lenstring.len;

   while (*curf && curn < curn_end)
   {
       if (*curn == '/' && *curf != '/')
          break;
       if (*curf != '+' && *curf != '#' && *curf != *curn)
          break;
       if (*curf == '+')
       {   // skip until we meet the next separator, or end of string
          char* nextpos = curn + 1;
          while (nextpos < curn_end && *nextpos != '/')
              nextpos = ++curn + 1;
       }
       else if (*curf == '#')
           curn = curn_end - 1;    // skip until end of string
       curf++;
       curn++;
   };

   return (curn == curn_end) && (*curf == '\0');
}
```

Vulnerabilities of Mbed OS CoAP

- Version
  - Commit d91ed5fa42ea0f32e4422a3c562e7b045a17da40
- Related Link

- https://github.com/ARMmbed/mbed-os/tree/master/features/frameworks/mbed-coap

Bug 1 (CVE-2019-17212, Fixed)

Type

Buffer overflow

Description

The CoAP parser is responsible for parsing received CoAP packets. The function *sn_coap_parser_options_parse()* parses CoAP input linearly using a while loop. Once an option is parsed in a loop, the current point (**packet_data_pptr*) is increased correspondingly. The pointer is restricted by the size of the received buffer, as well as a delimiter byte 0xFF, as shown in line 4 of the code snippet below.

```
static int8_t sn_coap_parser_options_parse(..., uint8_t **packet_data_pptr, ...)
{
    ...
    while (message_left && (**packet_data_pptr != 0xff)) {
       ...
       if (option_len == 13) {
           option_len = *(*packet_data_pptr + 1) + 13;
           (*packet_data_pptr)++;
       }
       ...
    }
    ...
}
```

Unfortunately, inside each while loop, the check of the value of **packet_data_pptr* is not strictly enforced. More specifically, inside a loop, **packet_data_pptr* could be increased and then dereferenced without checking. Moreover, there are many other functions in the format of *sn_coap_parser_xxx()* that do not check whether the pointer is within the bound of the allocated buffer. All of these lead to heap or stack buffer overflow, depending on how the CoAP packet buffer is allocated.

In the following, we list other locations which cause out-of-bound memory accesses rooted in this vulnerability.

- https://github.com/ARMmbed/mbed-os/blob/d91ed5fa42ea0f32e4422a3c562e7b045a17da40/features/frameworks/mbed-coap/source/sn_coap_parser.c#L660
- https://github.com/ARMmbed/mbed-os/blob/d91ed5fa42ea0f32e4422a3c562e7b045a17da40/features/frameworks/mbed-coap/source/sn_coap_parser.c#L331
- https://github.com/ARMmbed/mbed-os/blob/d91ed5fa42ea0f32e4422a3c562e7b045a17da40/features/frameworks/mbed-coap/source/sn_coap_parser.c#L257
- https://github.com/ARMmbed/mbed-os/blob/d91ed5fa42ea0f32e4422a3c562e7b045a17da40/features/frameworks/mbed-coap/source/sn_coap_parser.c#L310
- https://github.com/ARMmbed/mbed-os/blob/d91ed5fa42ea0f32e4422a3c562e7b045a17da40/features/frameworks/mbed-coap/source/sn_coap_parser.c#L313
- https://github.com/ARMmbed/mbed-os/blob/d0686fd30b4d3d02efdc7e4d0fbf0dfe173543b6/features/frameworks/mbed-coap/source/sn_coap_protocol.c#L2488

Bug 2 (CVE-2019-17211, Fixed)

Type

Integer overflow

Description

The CoAP builder is responsible for crafting outgoing CoAP messages. The function *sn_coap_builder_calc_needed_packet_data_size_2()* is used to calculate the needed memory for the CoAP message from the *sn_coap_hdr_s* data structure. Both *returned_byte_count* and *src_coap_msg_ptr->payload_len* are of type uint16_t. When added together, the result *returned_byte_count* will wrap around the maximum as shown in line 4. As a result, insufficient buffer is allocated for the corresponding CoAP message.

```
uint16_t sn_coap_builder_calc_needed_packet_data_size_2(const sn_coap_hdr_s *src_coap_msg_ptr, ...)
{
    ...
    returned_byte_count += src_coap_msg_ptr->payload_len;
    ...
}
```

When the data in the *sn_coap_hdr_s* is copied into the allocated buffer, out-of-bound memory access will happen as shown in line 4 of the code snippet below.

```
static int16_t sn_coap_builder_options_build_add_one_option(..., uint16_t option_len, const uint8_t *option_ptr, ...)
{
    ...
    memcpy(dest_packet, option_ptr, option_len);
    ...
}
```

In the following, we list other locations which will cause out-of-bound memory accesses rooted in this vulnerability.

- https://github.com/ARMmbed/mbed-os/blob/d0686fd30b4d3d02efdc7e4d0fbf0dfe173543b6/features/frameworks/mbed-coap/source/sn_coap_builder.c#L1090
- https://github.com/ARMmbed/mbed-os/blob/d0686fd30b4d3d02efdc7e4d0fbf0dfe173543b6/features/frameworks/mbed-coap/source/sn_coap_builder.c#L710
- https://github.com/ARMmbed/mbed-os/blob/d0686fd30b4d3d02efdc7e4d0fbf0dfe173543b6/features/frameworks/mbed-coap/source/sn_coap_builder.c#L524
- https://github.com/ARMmbed/mbed-os/blob/d0686fd30b4d3d02efdc7e4d0fbf0dfe173543b6/features/frameworks/mbed-coap/source/sn_coap_builder.c#L527
- https://github.com/ARMmbed/mbed-os/blob/d0686fd30b4d3d02efdc7e4d0fbf0dfe173543b6/features/frameworks/mbed-coap/source/sn_coap_builder.c#L528
- https://github.com/ARMmbed/mbed-os/blob/d0686fd30b4d3d02efdc7e4d0fbf0dfe173543b6/features/frameworks/mbed-coap/source/sn_coap_builder.c#L718
- https://github.com/ARMmbed/mbed-os/blob/d0686fd30b4d3d02efdc7e4d0fbf0dfe173543b6/features/frameworks/mbed-coap/source/sn_coap_builder.c#L746

Vulnerabilities of Mbed OS Client Cli

- Version
  - Commit d91ed5fa42ea0f32e4422a3c562e7b045a17da40

- Related Link
  - https://github.com/ARMmbed/mbed-os/tree/master/features/frameworks/mbed-client-cli

Bug 1 (Fixed)

Type

Buffer overflow

Description

The Mbed-Client-Cli library parses the command list from a user, which contains a string with semicolons separating each command. The library expects a null character at the end of the string. If the command list has no null character, the library will continue parsing characters following the end of the command list. Since the content of the memory after the command list is undefined, unpredictable behavior could be observed. For example, **ptr* may not be zero even at the end of *string_ptr* and the while loop will continue to execute as shown in line 4.

```
static char *next_command(char *string_ptr, operator_t *oper)
{
    ...
    while (*ptr != 0) {
        switch (*ptr) {
            ...
            case ('|'):
                if (ptr[1] == '|') {
                    ...
                }
            ...
        }
        ptr++;
    ...
}
```

In the following, we list other locations which will cause out-of-bound memory accesses rooted in this vulnerability.

- https://github.com/ARMmbed/mbed-os/blob/d91ed5fa42ea0f32e4422a3c562e7b045a17da40/features/frameworks/mbed-client-cli/source/ns_cmdline.c#L838
- https://github.com/ARMmbed/mbed-os/blob/d91ed5fa42ea0f32e4422a3c562e7b045a17da40/features/frameworks/mbed-client-cli/source/ns_cmdline.c#L872
- https://github.com/ARMmbed/mbed-os/blob/d91ed5fa42ea0f32e4422a3c562e7b045a17da40/features/frameworks/mbed-client-cli/source/ns_cmdline.c#L858
- https://github.com/ARMmbed/mbed-os/blob/d91ed5fa42ea0f32e4422a3c562e7b045a17da40/features/frameworks/mbed-client-cli/source/ns_cmdline.c#L581

Bug 2 (Fixed)

Type

Heap overflow

Description

The function *cmd_push()* is used to split a list of commands donated as *cmd_str* into separate commands. The delimiter is a semicolon in this case. Each command is then copied into a string array *cmd_ptr*. However, if the command does not terminate with a null character, the *strcpy()* will cause a buffer overflow as shown in line 4.

```
static void cmd_push(char *cmd_str, operator_t oper)
{
    ...
    strcpy(cmd_ptr->cmd_s, cmd_str);
    ...
}
```

Bug 3 (Fixed)

Type

Stack overflow

Description

The function *cmd_parse_argv()* is used to split a command string into separate arguments. The parameter *argv* of *cmd_parse_argv()* is a pointer array which points to each argument of a command represented by *string_ptr* as shown in line 6. The pointer array has **MAX_ARGUMENTS** entries, which is checked as shown in line 10. However, there is a boundary checking error. In particular, when *argc* is equal to MAX_ARGUMENTS, the if statement returns false and the while loop continues. Therefore, *argv[MAX_ARGUMENTS]* is assigned with a random value following the end of *string_ptr*. Since *argv[MAX_ARGUMENTS]* is on the stack, depending on the compilation option, it might overwrite the saved return address (LR register) or any other saved registers.

```
static int cmd_parse_argv(char *string_ptr, char **argv)
{
    ...
        str_ptr = string_ptr;
        do {
        argv[argc] = str_ptr;
        ...
        argc++;
        ...
        if (argc > MAX_ARGUMENTS) {
```

```
                tr_warn("Maximum arguments (%d) reached", MAX_ARGUMENTS);
                break;
        }
    ...
    } while (*str_ptr != 0);
}
```

Vulnerabilities of AWS FreeRTOS MQTT

- Version
  - Master branch and latest release (1.4.8) as of June 12, 2019
- Related Link
  - https://docs.aws.amazon.com/iot/latest/developerguide/mqtt.html

Bug 1 (CVE-2019-13120, Fixed)

Type

Buffer overflow

Description

After MQTT client receives a publish message from broker, it needs to respond with an acknowledge message. This is implemented in the function *prvProcessReceivedPublish()*.

As shown in the link https://github.com/aws/amazon-freertos/blob/master/lib/mqtt/aws_mqtt_lib.c#L1707, in the function *prvProcessReceivedPublish()*, from line 1707 to 1710, the library extracts topic length from the received network packet and stores the result in *xEventCallbackParams.u.xPublishData.usTopicLength*. However, no length checking is performed. If the network packet is manipulated, attacker can control the value of *usTopicLength*.

The variable *usTopicLength* is used in the program to further calculate the offset of pvData and the value of *ulDataLength*, all of which can be controlled. Note that *pvData* points to the received message.

The bug can also be exploited to leak arbitrary memory on the device to the attacker. For example, to craft the acknowledge message, *usTopicLength* is used to get the offset to the original message identifier. With manipulated *usTopicLength*, two bytes at arbitrary location are copied to the buffer *ucPUBACKPacket* as message identifier, which is later sent out.

Vulnerabilities of AWS FreeRTOS MQTTv2

- Version
  - 201808.00
- Related URL
  - https://github.com/aws/amazon-freertos/tree/master/libraries/c_sdk/standard/mqtt

Bug 1 (CVE-2019-17210, Fixed)

Type

Buffer overflow

Description

The MQTT library is used to receive, parse and send mqtt packets between a broker and a client. The function *_IotMqtt_SerializePublish()* is used to serialize the publish packet information represented by the param pPublishInfo. However, it doesn't check whether the length of *pPublishInfo->pPayload* matches with *pPublishInfo->payloadLength*. When the *memcpy()* copies *pPublishInfo->pPayload* into pBuffer, which points to a buffer that stores serialized MQTT publish packet, if the real length of *pPublishInfo->pPayload* is shorter than *pPublishInfo->payloadLength*, the memory immediately after *pPublishInfo->pPayload* will be copied into the MQTT publish packet, leading to information leakage as shown in the link https://github.com/aws/amazon-freertos/blob/390618c93c0be2c96059a865610c326a96de1e47/libraries/c_sdk/standard/mqtt/src/iot_mqtt_serialize.c#L1140.

If the out-going MQTT publish packet acknowledges the received MQTT packet by including the original received MQTT message, the current library is exploitable. Specifically, by manipulating the MQTT message, a malicious broker is able to receive memory contents on the MQTT client.

Vulnerabilities of FreeRTOS FATFS

- Version
  - 160919a
- Related URL
  - https://www.freertos.org/FreeRTOS-Labs/RTOS_labs_download.html
  - https://www.freertos.org/FreeRTOS-Labs/downloads/FreeRTOS-Plus-FAT-160919a-MIT.zip

Bug 1 (CVE-2019-18178, Fixed)

Type

Use after free

Description

FATFS is a compatible embedded FAT file system for use with or without RTOS.

The function *FF_Close()* is defined in ff_file.c. In the line 2970, the file handler pxFile is freed by the function *ffconfigFREE()*, which default is a macro definition of *vPortFree()*, but the handler pxFile reused as a part of arguments to flush modified file content from cache to disk by the function *FF_FlushCache()* in the line 2974.

The bugs type is use after free. If the freed heap of *pxFile* is reused before the function *FF_FlushCache()* is executed, which is possible due to task scheduler, random content may be writed to disk by the flushing cache operation.

Vulnerabilities of LiteOS MQTT

- Version
  - Commit a716aa039eb5bb902f369ed1d89772db8252866d
- Related Link
  - https://gitee.com/LiteOS/LiteOS

Bug 1 (Fixed)

Type

Buffer overflow

Description

The function *MQTTDeserialize_suback()* is used to deserialize the received subscribe ACK packet. It tries to deserialize the QoS and put it into the array *grantedQoSs* as shown in https://gitee.com/LiteOS/LiteOS/blob/master/components/connectivity/mqtt/MQTTPacket/src/MQTTSubscribeClient.c?_from=gitee_search#L128. The array *grantedQoSs* is declared and defined in the function *MQTTSubscribe()* and passed to the function *MQTTSubscribeWithResults()* as a parameter. However, it's a basic variable, but not an array. As a result, when the function *MQTTDeserialize_suback()* tries to parse more than one QoS, there is a stack overflow error.

Bug 2 (Fixed)

Type

Buffer overflow

Description

The function *MQTTTopicMatched()* is used to check if packet topic is matched with subscribed one. It tries to check if the first character of the packet topic or the subscribed topic is '$' as shown in https://gitee.com/LiteOS/LiteOS/blob/master/components/connectivity/mqtt/MQTTClient-C/src/MQTTClient.c?_from=gitee_search#L165. The variable topic may be aligned a value to *topic_name->lenstring.data* as shown in https://gitee.com/LiteOS/LiteOS/blob/master/components/connectivity/mqtt/MQTTClient-C/src/MQTTClient.c?_from=gitee_search#L190. The pointer topic_name gets its value by the function *readMQTTLenString()* which is called by the function *MQTTDeserialize_publish()* as shown in https://gitee.com/LiteOS/LiteOS/blob/master/components/connectivity/mqtt/MQTTPacket/src/MQTTDeserializePublish.c?_from=gitee_search#L56. The pointer *topic_name->lenstring.data* get value from the pointer *pptr*, which is the pointer to the output buffer. However, it do not check if the *pptr* is NULL or 0x0 as shown in https://gitee.com/LiteOS/LiteOS/blob/master/components/connectivity/mqtt/MQTTPacket/src/MQTTPacket.c?_from=gitee_search#L228. If so, the function *MQTTTopicMatched()* will access invalid memory.

Vulnerabilities of LiteOS LWM2M Client

- Version
  - Commit a716aa039eb5bb902f369ed1d89772db8252866d
- Related Link
  - https://gitee.com/LiteOS/LiteOS

Bug 1 (Fixed)

Type

Use after free

Description

The function *lwm2m_close()* will destroy all the information about lwm2m link. It calls the function *prv_deleteServerList()* to free the linked list contextP->serverList and the function *prv_deleteTransactionList()* to clear the linked list contextP->transactionList as shown in https://gitee.com/LiteOS/LiteOS/blob/master/components/connectivity/lwm2m/core/liblwm2m.c?_from=gitee_search#L226. When the callback function *prv_handleRegistrationReply()* is called by the function *prv_deleteTransactionList()*, it tries to access the memory pointed by the pointer *transacP->userData* as shown in https://gitee.com/LiteOS/LiteOS/blob/master/components/connectivity/lwm2m/core/registration.c?_from=gitee_search#L225. However, the pointer *transacP->userData* points to the memory server aligned by the function *prv_register()* as shown in https://gitee.com/LiteOS/LiteOS/blob/master/components/connectivity/lwm2m/core/registration.c?_from=gitee_search#L318 and already freed by the function *prv_deleteServerList()* before the function *prv_deleteTransactionList()*. This will lead to a use-after-free fault.

Bug 2 (Fixed)

Type

Buffer overflow

Description

The function *coap_parse_message()* is used to parse coap packet. When avoiding code duplication without function overhead, it traverses the memory pointed by the pointer *current_option*, but the while loop do not check if reach the end of the memory range as shown in https://gitee.com/LiteOS/LiteOS/blob/master/components/connectivity/lwm2m/core/er-coap-13/er-coap-13.c?_from=gitee_search#L732. This will lead to buffer overflow.

Vulnerabilities of LwIP

- Version
  - Master branch as of June, 2020 or release version 2.1.2
- Related Link
  - https://savannah.nongnu.org/git/?group=lwip

Bug 1 (CVE-2020-22285, v2.1.2, Fixed)

Type

Buffer overflow

Description

The function *icmp6_send_response_with_addrs_and_netif()* tries to parse an ICMPv6 packet and send it out. Inside it, the function *SMEMCPY()* as shown in line 408 of icmp6.c tries to copy a buffer pointed to by *p->payload* with length *(IP6_HLEN + LWIP_ICMP6_DATASIZE)*. However, this buffer may be smaller than *(IP6_HLEN + LWIP_ICMP6_DATASIZE)*. If this happens, it will cause a memory leakage. To fix this, the length should be compared with *p->len*.

```
static void icmp6_send_response_with_addrs_and_netif(struct pbuf *p, u8_t code, u32_t data, u8_t type, const ip6_addr_t *reply_src, const ip6_addr_t *reply_dest, struct netif *netif){
    ...
    SMEMCPY((u8_t *)q->payload + sizeof(struct icmp6_hdr), (u8_t *)p->payload, IP6_HLEN + LWIP_ICMP6_DATASIZE);
    ...
}
```

Bug 2 (CVE-2020-22283, master branch, Fixed)

Type

Buffer overflow

Description

This bug is related to bug 1. We have observed the changes made to the same place in the master branch. However, the bug still exists. The function *pbuf_take_at()* replaces the function *SMEMCPY()* in the master branch. However, it is still vulnerable.

The function *pbuf_take_at()* tries to copy fields from the original packet as shown in line 409 of icmp6.c. The parameter len of the function *pbuf_take_at()* is the length of another parameter dataptr. However, the function *icmp6_send_response_with_addrs_and_netif()* passes the parameters *p->payload* and *p->tot_len* to the function *pbuf_take_at()*, which are the total length of the *p->payload* plus all payloads length of its following pbuf. If *p->tot_len* is larger than the length of *p->payload*, the memory will leak to remote attackers through the network. To fix this, the datalen should be *p->len*, not *p->tot_len*.

```
static void icmp6_send_response_with_addrs_and_netif(struct pbuf *p, u8_t code, u32_t data, u8_t type, const ip6_addr_t *reply_src, const ip6_addr_t *reply_dest, struct netif *netif)
{
    ...
    pbuf_take_at(q, p->payload, datalen, sizeof(struct icmp6_hdr));
    ...
}
```

Bug 3 (CVE-2020-22284, master branch and 2.1.2, Fixed)

Type

Buffer overflow

Description

This bug is similar to bug 2. The function *zepif_linkoutput()* tries to parse an 6LoWPAN TX packet as UDP broadcast. When it calls the function *pbuf_take_at()* as shown in line 204 of zepif.c, the same incorrectly used parameters are passed as the Bug 2. In particular, the *p->tot_len* is the total length of the *p->payload* and all payloads length of its following pbuf. If *p->tot_len* is larger than the length of *p->payload*, the memory will leak to remote attackers through the network. To send the whole packet, it should use a loop to traverse the list of *p->next* and send all the payloads with length *p->tot_len*.

```
zepif_linkoutput(struct netif *netif, struct pbuf *p){
    ...
    err = pbuf_take_at(q, p->payload, p->tot_len, sizeof(struct zep_hdr));
    ...
}
```

Vulnerabilities of STM PLC

- Version
  - 1.0.2
- Related Link
  - https://www.st.com/content/st_com/en/products/embedded-software/mcu-mpu-embedded-software/stm32-embedded-software/stm32-ode-function-pack-sw/fp-ind-plcwifi1.html

Bug 1 (Fixed)

Type

Buffer overflow

Description

The variable *decoded_index* is used to mark the decoded byte in the parameter frame. When the first frame byte decode is an ASCII number 0-9, *decoded_index* will increase to one as shown in line 270 of Ladder_Lib.c. However, if the next frame byte is equal to '=', *decoded_index* will decrease to zero as shown in line 344 of Ladder_Lib.c. Accessing the global array *ServerData_RX* with index *decoded_index - 1*, as shown in line 345 of Ladder_Lib.c, will lead to overflow which will lead the program to jump to an unexpected branch.

```
int16_t WiFi_Decode (uint8_t* frame)
{
    ...
    decoded_index = 0;
    ...
    decode=frame[dec_index++];
    switch (decode)
    {
        ...
        case '9':
        ...
        ServerData_RX[decoded_index++]= (decode - 0x30);
        ...
        case '=':
        if ((decoded_index > 0 )&&(ServerData_RX[decoded_index-1]<=9)&&
        (ServerData_RX[decoded_index-1] >=0))
        {
            decoded_index--;
            if((ServerData_RX[decoded_index-1]>0)&&(ServerData_RX[decoded_index-1] <=9))
        }
    }
    ...
}
```

Bug 2 (Fixed)

Type

Buffer overflow

Description

The global variable *rung_pos* is initialized to zero when the program starts. If no changes are made to it, accessing the global array output with index *rung_pos - 1* which equals -1 as shown in line 387 of Ladder_Lib.c will lead to a buffer overflow.

```
uint8_t rung_pos=0;
int16_t WiFi_Decode (uint8_t* frame)
{
    ...
    memset(&output[rung_pos-1],0,EXPRESSION_MAX_SIZE);
    ...
}
```

Bug 3 (Fixed)

Type

Buffer overflow

Description

The global variable *component_index* is initialized to zero when the program starts. If no changes are made to it or changing to 1, accessing the global array Component with index *component_index-1* as shown in line 265 of Ladder_Lib.c or *component_index-2* as shown in line 267 of Ladder_Lib.c will lead to a buffer overflow and the program may jump to unexpected branch.

```
uint16_t component_index=0;
int16_t WiFi_Decode (uint8_t* frame)
{
    ...
    if((((Component[0]=='T')||(Component[0]=='C'))&&Component[component_index-1]!=')')
    {
        if(Component[component_index-2]!=')')
        ...
}
```

Bug 4 (Fixed)

Type

Buffer overflow

Description

Inside the loop of the function *Component_parser*, the variable a is used to access the global array *comp_param* as the index as shown in line 502 of Ladder_Lib.c. However, the function hasn't checked the availability of the value of the index a which may lead to a buffer overflow after four iterations.

```
uint16_t comp_param[4];
int8_t Component_parser(void)
{
    ...
    uint8_t a=0;
    ...
    do
    {
        ...
        comp_param[a++]=Component[component_index];
        ...
    }while (Component[component_index]!='#');
    ...
}
```

Bug 5 (Fixed)

Type

Buffer overflow

Description

The local variable a is initialized to zero as the index to access the global array *comp_param*. When no changes are made to the index a, accessing the array *comp_param* with index *a-1* as shown in line 458 of Ladder_Lib.c will lead to a buffer overflow.

```
uint16_t comp_param[4];
int8_t Component_parser(void)
{
    ...
    uint8_t a=0;
    ...
    comp_param[a-1]=(comp_param[a-1]*(uint8_t)pow(10,1))+(Component[component_index]-0x30);
    ...
}
```

Bug 6 (Fixed)

Type

Buffer overflow

Description

In the function *Evalute_Expression*, the local variable *output_pos* is derived from the variable argument as shown in line 569 of Ladder_Lib.c, which equals *output[output_index].Expression[index++]* as shown in line 536 of Ladder_Lib.c. When used as an index to access the global array output as shown in line 570 of Ladder_Lib.c, no availability check will lead to a buffer overflow and the program may jump to an unexpected branch.

```
#define MAX_OUTPUT_NUMBER               30
...
OutputStructure_Typedef         output[MAX_OUTPUT_NUMBER];
...
uint8_t Evalute_Expression(uint8_t output_index)
{
    while((argument=output[output_index].Expression[index++])!=0)
    ...
    uint8_t output_pos=(argument&0x1F)-1;
    if(output[output_pos].output_value!=-1)
    ...
}
```

Bug 7 (Fixed)

Type

Buffer overflow

Description

The variable *index_c*, which equals *comp_param[1]-1* as shown in line 486 of Ladder_Lib.c, is used as the index to access the global array *counter_up* and *num_obj* as shown in line from 487 to 493 of Ladder_Lib.c which may lead to a buffer overflow and unexpected branch jump.

```
#define MAX_COMPONENT_NUMBER             50
...
uint16_t num_obj[MAX_COMPONENT_NUMBER];
...
CounterStruct_Typedef           counter_up[MAX_COMPONENT_NUMBER];
...
int8_t Component_parser(void)
{
...
```

```
index_c=comp_param[1]-1;
counter_up[index_c].CNT_number=comp_param[1];
counter_up[index_c].CNT_val=comp_param[2];
counter_up[index_c].CNT_dir=comp_param[3];
counter_up[index_c].CNT_output=Component[component_index+1];
if(counter_up[index_c].CNT_dir==0)
num_obj[index_c]=counter_up[index_c].CNT_val;
```

Bug 8 (Fixed)

Type

Buffer overflow

Description

The variable *component_index* is used as an index for accessing the array *Component* and will increase with many operations for example the code snippet as shown in line 337 of Ladder_Lib.c. It may lead to a buffer overflow without checking the availability of the index *component_index* when accessing the array *Component* as shown in line 421 of Ladder_Lib.c.

```
uint8_t Component[512];
...
int16_t WiFi_Decode (uint8_t* frame)
{
    ...
    Component[component_index++]=decode;
    ...
    Component[component_index]='#';
    ...
}
```

Bug 9 (Fixed)

Type

Buffer overflow

Description

Within the while loop as shown in line 536 of Ladder_Lib.c, the variable *res_index* will increase iteratively. However, no check on the availability of the index *res_index* when accessing the array *element_buffer* as shown in line 761 of Ladder_Lib.c will lead to a buffer overflow.

```
uint8_t Evalute_Expression(uint8_t output_index)
{
    ...
    while((argument=output[output_index].Expression[index++])!=0)
    {
        ...
        element_buffer[res_index++] = Get_Input(Input_CHS,(argument&0x0F));
        ...
    }
    ...
}
```

Vulnerabilities of uTasker Modbus

- Version
  - Jun, 2020
- Related Link
  - https://www.utasker.com/modbus.html

Bug 1 (Fixed)

Type

Buffer overflow

Description

With MODBUS_ASCII enabled, if no character received within the expected time when receiving in ASCII mode, the library tries to write to global array *iModbusSerialState* with status code MODBUS_ASCII_HUNTING as shown in line 437 of MODBUS.c. However, the library hasn't checked the availability of index calculated from input message *ucInputMessage[MSG_TIMER_EVENT]* as shown in line 435 of MODBUS.c, which may lead to a global buffer overflow in the array *iModbusSerialState*. (buffer overflow or index integer overflow to negative)

```
extern void fnMODBUS(TTASKTABLE *ptrTaskTable)
{
    ...
    if (ucInputMessage[MSG_TIMER_EVENT] <= T_ASCII_INTER_CHAR_TIMEOUT) {
    #if defined MODBUS_ASCII
    iModbusSerialState[T_ASCII_INTER_CHAR_TIMEOUT - ucInputMessage[MSG_TIMER_EVENT]] = MODBUS_ASCII_HUNTING;
```

```
    ...
}
```

Bug 2 (Fixed)

Type

Buffer overflow

Description

With MODBUS_GATE_WAY_ROUTING enabled and the library tries to write status to indicate canceling return routing information on failure, the library tries to write global array *open_routes* with the index calculated from *ucInputMessage[MSG_TIMER_EVENT]* as shown in line 461 of MODBUS.c, which may lead to a global buffer overflow. (buffer overflow or index integer overflow to negative)

```
extern void fnMODBUS(TTASKTABLE *ptrTaskTable)
{
    ...
    #if defined MODBUS_GATE_WAY_ROUTING
    open_routes[T_TIMER_SLAVE - ucInputMessage[MSG_TIMER_EVENT]].Valid = 0;
    #endif
    ...
}
```

Bug 3 (Fixed)

Type

Buffer overflow

Description

The function *fnNextSerialQueue()* tries to access the global array *modbus_queue* with index *ucMODBUSport* as shown in line 1992 of MODBUS.c, which is equal to *modbus_rx_function->ucMODBUSport* as shown in line 1990 of MODBUS.c. The parameter *modbus_rx_function* is passed from the function fnMODBUS, and *modbus_rx_function->ucMODBUSport* is calculated from *ucInputMessage[MSG_TIMER_EVENT]* as shown in line 464 of MODBUS.c, which may lead to the global array *modbus_queue* overflow.

```
extern void fnMODBUS(TTASKTABLE *ptrTaskTable)
{
    ...
    modbus_rx_function.ucMODBUSport = (unsigned char)(T_TIMER_SLAVE - ucInputMessage[MSG_TIMER_EVENT]);
    ...
    fnNextSerialQueue(&modbus_rx_function);
    ...
}

static void fnNextSerialQueue(MODBUS_RX_FUNCTION *modbus_rx_function)
{
    unsigned char ucMODBUSport = modbus_rx_function->ucMODBUSport;
    ...
    if (modbus_queue[ucMODBUSport].ucOutstanding > 1)
    ...
}
```

Bug 4 (Fixed)

Type

Buffer overflow

Description

The function *fnMODBUS()* tries to judge the serial status of the specific receiver Modbus port as shown in line 466 of MODBUS.c. However, the index, *modbus_rx_function.ucMODBUSport*, is calculated from *ucInputMessage[MSG_TIMER_EVENT]* as shown in line 464 of MODBUS.c, which may lead to a buffer overflow and jump to an unexpected branch.

```
extern void fnMODBUS(TTASKTABLE *ptrTaskTable)
{
    ...
    modbus_rx_function.ucMODBUSport = (unsigned char)(T_TIMER_SLAVE - ucInputMessage[MSG_TIMER_EVENT]);
    #if defined MODBUS_ASCII
    if (iModbusSerialState[modbus_rx_function.ucMODBUSport] >= MODBUS_ASCII_HUNTING) {
    ...
}
```

Bug 5 (Fixed)

Type

Buffer overflow

Description

When the function *fnMODBUS()* tries to check if the received frame is too large as shown in line 495 of MODBUS.c, the index *ucMODBUSport* of the array *rxFrameLength* calculated from *ucInputMessage[MSG_INTERRUPT_EVENT]* as shown in line 487 of MODBUS.c, may lead to a buffer overflow and the program may jump to an unexpected branch.

```
extern void fnMODBUS(TTASKTABLE *ptrTaskTable)
{
    ...
    unsigned char ucMODBUSport = (EVENT_VALID_FRAME_RECEIVED - ucInputMessage[MSG_INTERRUPT_EVENT]);
    ...
    if (rxFrameLength[ucMODBUSport] > MODBUS_RX_BUFFER_SIZE) {
    ...
}
```

Vulnerabilities of NXP SDK USB Driver

- Version
  - 2.7.0
- Related Link
  - https://mcuxpresso.nxp.com/en/welcome

Bug 1 (CVE-2021-38258, Fixed)

Type

Buffer overflow

Description

In the C file middleware/usb/host/usb_host_devices.c, function *USB_HostProcessCallback()* is responsible for handling usb device enumeration. When parsing the configuration descriptor, it extracts the wTotalLength field in line 454.

```
deviceInstance->configurationLen = USB_SHORT_FROM_LITTLE_ENDIAN_ADDRESS(configureDesc->wTotalLength);
```

However, this field is controlled by the attackers and can be 0. In line 481, a buffer is allocated for the configuration description. The return value of *malloc(0)* is undefined in C standard. In the default tool chain (arm-none-eabi-gcc), a non-NULL pointer is returned. Although the return value is checked in line 484. However, it just passes the check.

Since *wTotalLength* is 0, no request is sent again, and the state machine goes to line 490. The check at line 491 is passed, and *deviceInstance->configurationDesc* is being parsed. However, *deviceInstance->configurationDesc* has zero length and should not be accessed. For example, line 495 triggers a buffer overflow. Also, the function *USB_HostParseDeviceConfigurationDescriptor()* can be exploited.

Bug 2 (CVE-2021-38258, Fixed)

Type

Buffer overflow

Description

This bug is similar to bug1. This time, *wTotalLength* could be a value less than 8 (e.g., 7) and usb_host_devices.c:472 allocates 8 bytes for *deviceInstance->configurationDesc* (due to alignment). However, at line 495, the field bMaxPower exceeds the range of *deviceInstance->configurationDesc* (at offset 9), leading to a buffer overread.

Bug 3 (CVE-2021-38260, Fixed)

Type

Buffer overflow

Description

In function *USB_HostParseDeviceConfigurationDescriptor()* which parses the configuration descriptor, there multiple locations subject to buffer overflow. For example, line 790 makes sure that *unionDes* is always less than *endPos*. However, it does not check the real memory address accessed with *unionDes*. At line 792, the field *common.bDescriptorType* is at offset 4. Therefore, *unionDes->common.bDescriptorType* addresses *unionDes+4*, which can exceed the range of allocated configurationDesc, leading to a buffer overflow.

Vulnerabilities of STM SDK USB Driver

- Version
  - STM32Cube_FW_H7_V1.8.0
- Related Link
  - https://www.st.com/content/st_com/en/products/embedded-software/mcu-mpu-embedded-software/stm32-embedded-software/stm32cube-mcu-mpu-packages/stm32cubeh7.html
  - https://github.com/STMicroelectronics/STM32CubeH7

Bug 1 (CVE-2021-34268, Fixed)

Type

Denial of Service

Description

The function *USBH_ParseDevDesc()* parses the device descriptor by input data from a USB device.

The valid max packet size of the device descriptor should be 8, 16, 32, and 64 as USB specification required. However, the function *USBH_ParseDevDesc()* doesn't check the value of *dev_desc->bMaxPacketSize* as shown in https://github.com/STMicroelectronics/STM32CubeH7/blob/79196b09acfb720589f58e93ccf956401b18a191/Middlewares/ST/STM32_USB_Host_Library/Core/Src/usbh_ctlreq.c#L355. The variable *dev_desc->bMaxPacketSize* will be used as the size to construct the control pipe between host and device as shown in https://github.com/STMicroelectronics/STM32CubeH7/blob/79196b09acfb720589f58e93ccf956401b18a191/Middlewares/ST/STM32_USB_Host_Library/Core/Src/usbh_core.c#L828. If *bMaxPacketSize* is zero, the firmware will get the error status USBH_FAIL in the function *USBH_HandleControl()* called by the function *USBH_CtlReq()* when trying to communicate with the outside world by IN and OUT pipe in the future and the host will try to re-enumerate. This process will loop again and again.

Bug 2 (CVE-2021-34259, Fixed)

Type

Buffer overflow

Description

The function *USBH_ParseCfgDesc()* parses the configuration descriptor, interface descriptor, and endpoint descriptor by input data from a USB device.

However, it doesn't check the validity of the variable *cfg_desc->wTotalLength* compared with the total length of the input buffer as shown in https://github.com/STMicroelectronics/STM32CubeH7/blob/79196b09acfb720589f58e93ccf956401b18a191/Middlewares/ST/STM32_USB_Host_Library/Core/Src/usbh_ctlreq.c#L395. This will cause the following program including calling to the function *USBH_GetNextDesc()*, *USBH_ParseInterfaceDesc()* and *USBH_ParseEPDesc()* configure the system incorrectly.

Bug 3 (CVE-2021-34259, Fixed)

Type

Buffer overflow

Description

The function *USBH_ParseCfgDesc()* parses the configuration descriptor, interface descriptor, and endpoint descriptor by input data from a USB device.

However, it doesn't check the validity of the variable *cfg_desc->bLength* compared with the total length of the input buffer and the restriction of enumeration requirement by USB specification as shown in https://github.com/STMicroelectronics/STM32CubeH7/blob/79196b09acfb720589f58e93ccf956401b18a191/Middlewares/ST/STM32_USB_Host_Library/Core/Src/usbh_ctlreq.c#L393. This will cause the following function *USBH_GetNextDesc()* and others access wrong memory region and the system will be configured incorrectly.

Bug 4 (CVE-2021-34261, Fixed)

Type

Denial of Service

Description

The function *USBH_ParseCfgDesc()* parses the configuration descriptor, interface descriptor, and endpoint descriptor by input data from a USB device.

And it set the variable *cfg_desc->bmAttributes* by the input data from the USB device. This variable will be used as part of a judgment in the function *USBH_Process()* as shown in https://github.com/STMicroelectronics/STM32CubeH7/blob/79196b09acfb720589f58e93ccf956401b18a191/Middlewares/ST/STM32_USB_Host_Library/Core/Src/usbh_core.c#L643. With a malformed value, the remote wakeup may be enabled as shown in https://github.com/STMicroelectronics/STM32CubeH7/blob/79196b09acfb720589f58e93ccf956401b18a191/Middlewares/ST/STM32_USB_Host_Library/Core/Src/usbh_core.c#L643. If the hardware doesn't support this feature, the system will hang due to a FAIL return value by the function *USBH_HandleControl()*.

Bug 5 (CVE-2021-34259, Fixed)

Type

Buffer overflow

Description

The function *USBH_ParseCfgDesc()* parses the configuration descriptor, interface descriptor, and endpoint descriptor by input data from a USB device.

It tries to parse the endpoint descriptor when the variable *pdesc->bDescriptorType* is equal to USB_DESC_TYPE_ENDPOINT as shown in https://github.com/STMicroelectronics/STM32CubeH7/blob/79196b09acfb720589f58e93ccf956401b18a191/Middlewares/ST/STM32_USB_Host_Library/Core/Src/usbh_ctlreq.c#L420. If the variable *pdesc->bDescriptorType* is not equal to USB_DESC_TYPE_ENDPOINT, the firmware will try to access further memory until reaching the number *pif->bNumEndpoints* as shown in https://github.com/STMicroelectronics/STM32CubeH7/blob/79196b09acfb720589f58e93ccf956401b18a191/Middlewares/ST/STM32_USB_Host_Library/Core/Src/usbh_ctlreq.c#L417. If the variable *cfg_desc->wTotalLength* is also malformed, the firmware will overflow. If not, some endpoint descriptors will be left unset as shown in https://github.com/STMicroelectronics/STM32CubeH7/blob/79196b09acfb720589f58e93ccf956401b18a191/Middlewares/ST/STM32_USB_Host_Library/Core/Src/usbh_ctlreq.c#L422. And the firmware behaves unpredictably as shown from line 180 to 200 in https://github.com/STMicroelectronics/STM32CubeH7/blob/79196b09acfb720589f58e93ccf956401b18a191/Middlewares/ST/STM32_USB_Host_Library/Class/MSC/Src/usbh_msc.c.

Bug 6 (CVE-2021-34267, Fixed)

Type

Denial of Service

Description

The function *USBH_MSC_InterfaceInit()* inits the status of MSC handler. It initializes the IN endpoint and OUT endpoint as shown from line 180 to line 200 in https://github.com/STMicroelectronics/STM32CubeH7/blob/79196b09acfb720589f58e93ccf956401b18a191/Middlewares/ST/STM32_USB_Host_Library/Class/MSC/Src/usbh_msc.c.

However, when the variable *bEndpointAddress* of endpoint descriptor are both masked as IN or OUT without checking as shown in https://github.com/STMicroelectronics/STM32CubeH7/blob/79196b09acfb720589f58e93ccf956401b18a191/Middlewares/ST/STM32_USB_Host_Library/Core/Src/usbh_ctlreq.c#L468, the MSC handler will also only initialize the IN or OUT part as shown from line 180 to line 200 in https://github.com/STMicroelectronics/STM32CubeH7/blob/79196b09acfb720589f58e93ccf956401b18a191/Middlewares/ST/STM32_USB_Host_Library/Class/MSC/Src/usbh_msc.c.

Bug 7 (CVE-2021-34262, Fixed)

Type

Denial of Service

Description

The function *USBH_ParseEPDesc()* parses the endpoint descriptor of a USB device.

It doesn't check if the variable *ep_descriptor->wMaxPacketSize* is greater than zero as shown in https://github.com/STMicroelectronics/STM32CubeH7/blob/79196b09acfb720589f58e93ccf956401b18a191/Middlewares/ST/STM32_USB_Host_Library/Core/Src/usbh_ctlreq.c#L470. If zero, the MSC handler will not able to communicate with outside world as shown from line 180 to line 200 in https://github.com/STMicroelectronics/STM32CubeH7/blob/79196b09acfb720589f58e93ccf956401b18a191/Middlewares/ST/STM32_USB_Host_Library/Class/MSC/Src/usbh_msc.c.

Bug 8 (CVE-2021-34262, Fixed)

Type

Denial of Service

Description

The function *USBH_ParseEPDesc()* parses the endpoint descriptor of a USB device.

It doesn't check if t he variable *ep_descriptor->bInterval* is greater than zero. This variable will be used in the class to set the polling interval, for example as shown in https://github.com/STMicroelectronics/STM32CubeH7/blob/79196b09acfb720589f58e93ccf956401b18a191/Middlewares/ST/STM32_USB_Host_Library/Class/AUDIO/Src/usbh_audio.c#L858. If set zero, the system will hang when polling operation.

Bug 9 (CVE-2021-34260, Fixed)

Type

Buffer overflow

Description

The function *USBH_ParseInterfaceDesc()* parse interface descriptor. It's called by the function *USBH_ParseCfgDesc()* as shown in https://github.com/STMicroelectronics/STM32CubeH7/blob/79196b09acfb720589f58e93ccf956401b18a191/Middlewares/ST/STM32_USB_Host_Library/Core/Src/usbh_ctlreq.c#L413.

It doesn't check the validity of the variable *if_descriptor->bLength* compared with the total length of the input buffer which may cause a buffer overflow by the following called function *USBH_GetNextDesc()* as shown in https://github.com/STMicroelectronics/STM32CubeH7/blob/79196b09acfb720589f58e93ccf956401b18a191/Middlewares/ST/STM32_USB_Host_Library/Core/Src/usbh_ctlreq.c#L419.

Bug 10 (CVE-2021-34262, Fixed)

Type

Buffer overflow

Description

The function *USBH_ParseEPDesc()* parses endpoint descriptor. It's called by the function *USBH_ParseCfgDesc()* as shown in https://github.com/STMicroelectronics/STM32CubeH7/blob/79196b09acfb720589f58e93ccf956401b18a191/Middlewares/ST/STM32_USB_Host_Library/Core/Src/usbh_ctlreq.c#L423.

It doesn't check the validity of the variable *ep_descriptor->bLength* compared with the total length of the input buffer which may cause buffer overflow by the following called function *USBH_GetNextDesc()* as shown in https://github.com/STMicroelectronics/STM32CubeH7/blob/79196b09acfb720589f58e93ccf956401b18a191/Middlewares/ST/STM32_USB_Host_Library/Core/Src/usbh_ctlreq.c#L419.
