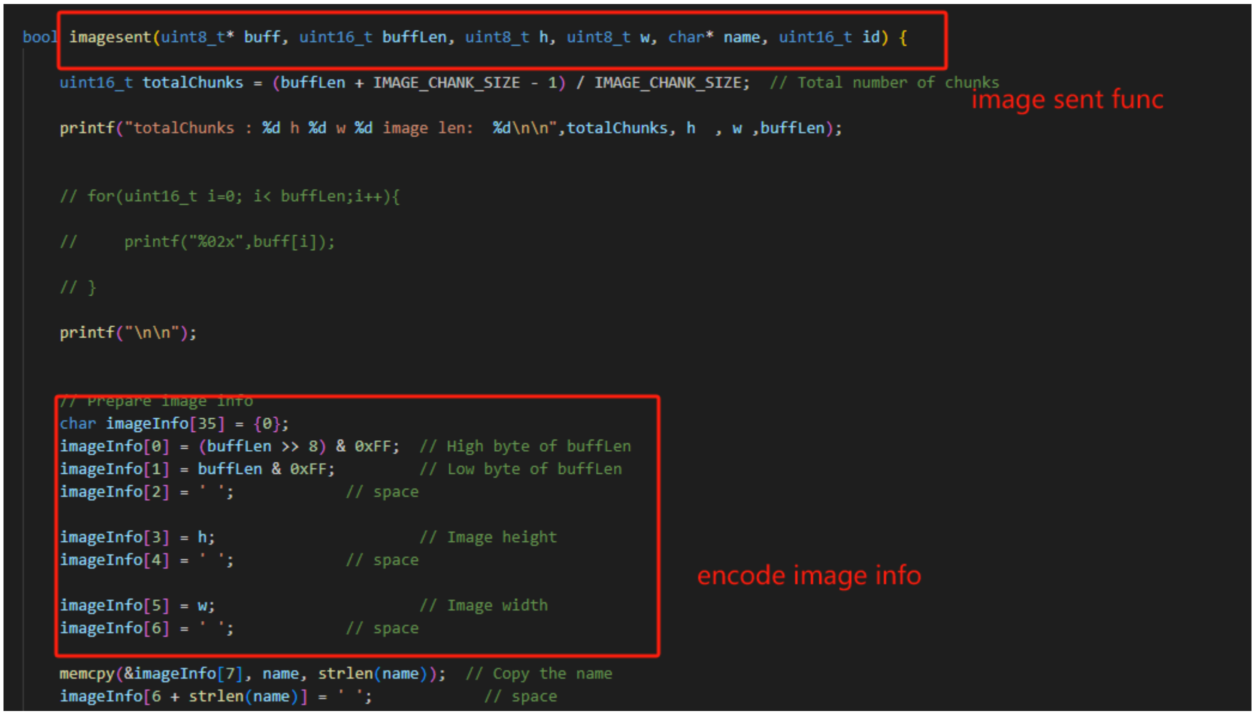
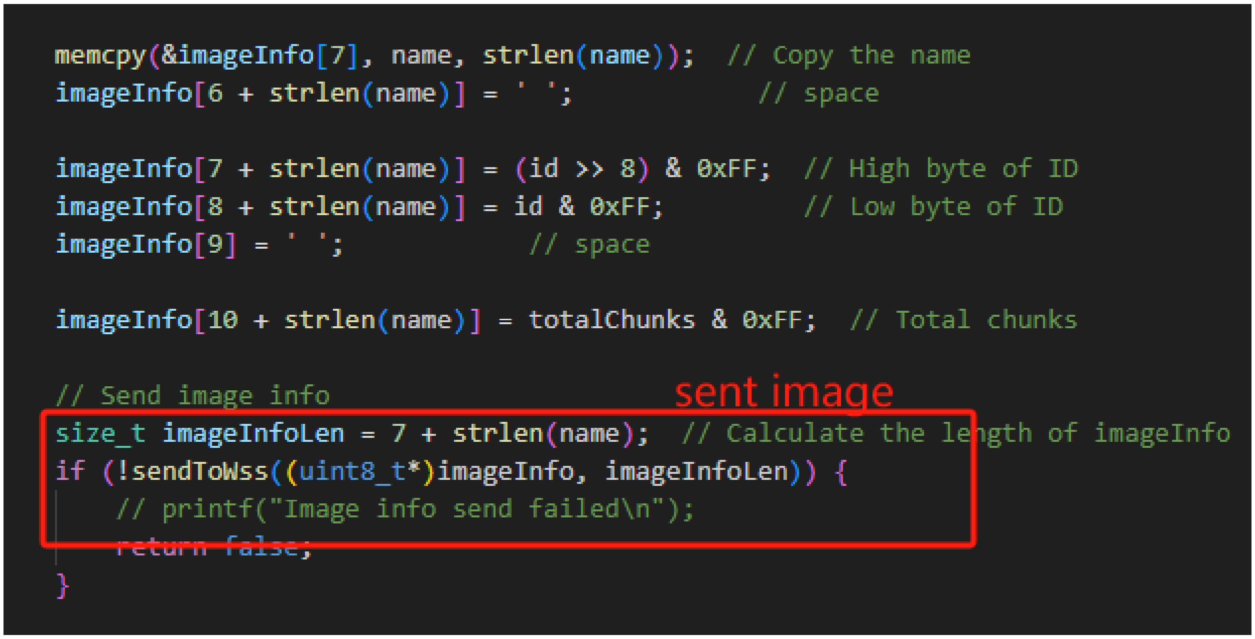
Sozib Al Mamun

Embedded Software Engineer

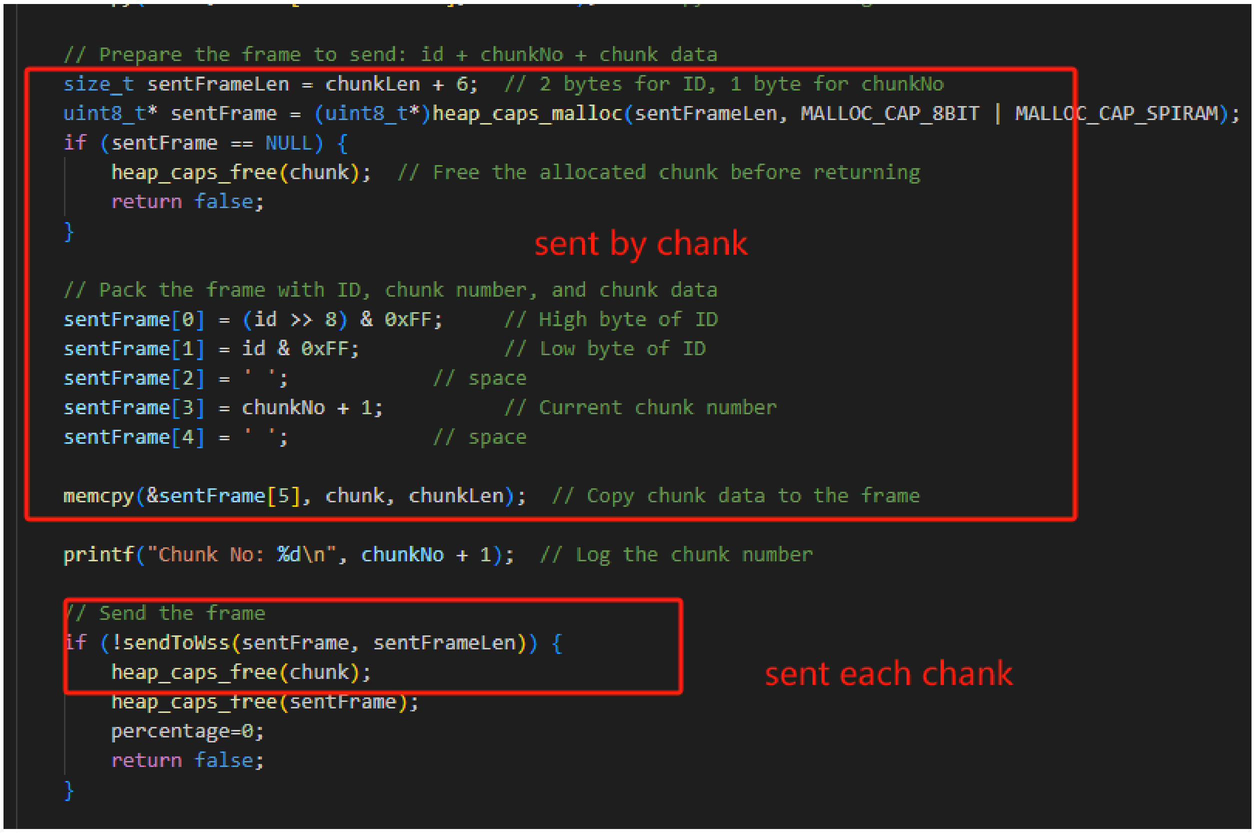


```
bool imagesent(uint8_t* buff, uint16_t buffLen, uint8_t h, uint8_t w, char* name, uint16_t id) {

    uint16_t totalChunks = (buffLen + IMAGE_CHANK_SIZE - 1) / IMAGE_CHANK_SIZE;  // Total number of chunks

    printf("totalChunks : %d h %d w %d image len:  %d\n\n",totalChunks, h  , w ,buffLen);


    // for(uint16_t i=0; i< buffLen;i++){

    //     printf("%02x",buff[i]);

    // }

    printf("\n\n");


    // Prepare image info
    char imageInfo[35] = {0};
    imageInfo[0] = (buffLen >> 8) & 0xFF;  // High byte of buffLen
    imageInfo[1] = buffLen & 0xFF;         // Low byte of buffLen
    imageInfo[2] = ' ';             // space

    imageInfo[3] = h;                      // Image height
    imageInfo[4] = ' ';             // space

    imageInfo[5] = w;                      // Image width
    imageInfo[6] = ' ';             // space

    memcpy(&imageInfo[7], name, strlen(name));  // Copy the name
    imageInfo[6 + strlen(name)] = ' ';            // space
```



```
memcpy(&imageInfo[7], name, strlen(name));  // Copy the name
imageInfo[6 + strlen(name)] = ' ';            // space

imageInfo[7 + strlen(name)] = (id >> 8) & 0xFF;  // High byte of ID
imageInfo[8 + strlen(name)] = id & 0xFF;         // Low byte of ID
imageInfo[9] = ' ';            // space

imageInfo[10 + strlen(name)] = totalChunks & 0xFF;  // Total chunks

// Send image info
size_t imageInfoLen = 7 + strlen(name);  // Calculate the length of imageInfo
if (!sendToWss((uint8_t*)imageInfo, imageInfoLen)) {
    // printf("Image info send failed\n");
    return false;
}
```

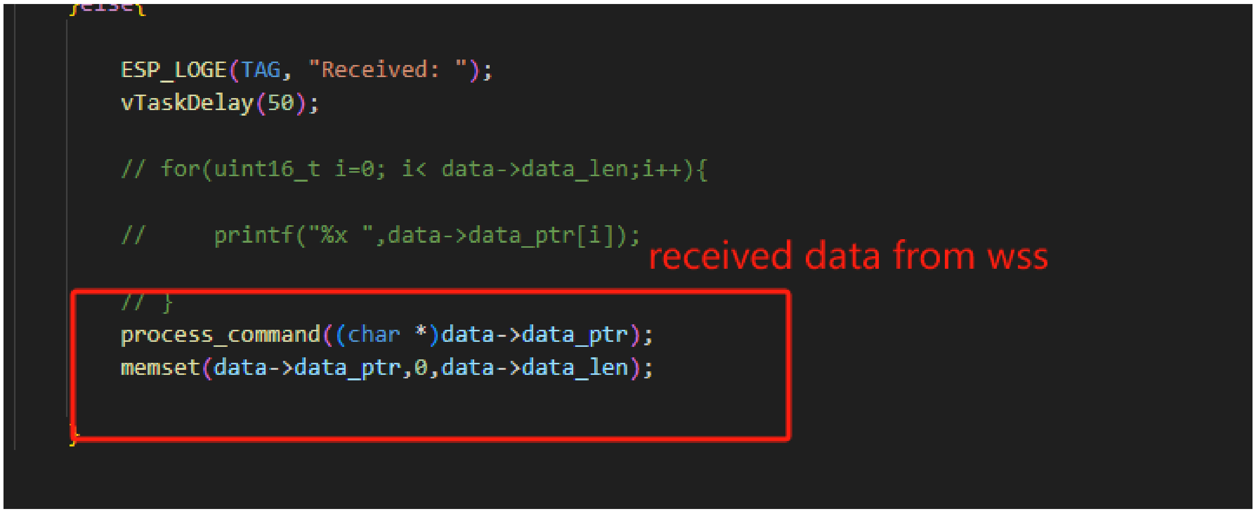
sent image



```
// Prepare the frame to send: id + chunkNo + chunk data
size_t sentFrameLen = chunkLen + 6;  // 2 bytes for ID, 1 byte for chunkNo
uint8_t* sentFrame = (uint8_t*)heap_caps_malloc(sentFrameLen, MALLOC_CAP_8BIT | MALLOC_CAP_SPIRAM);
if (sentFrame == NULL) {
    heap_caps_free(chunk);  // Free the allocated chunk before returning
    return false;
}

// Pack the frame with ID, chunk number, and chunk data
sentFrame[0] = (id >> 8) & 0xFF;     // High byte of ID
sentFrame[1] = id & 0xFF;            // Low byte of ID
sentFrame[2] = ' ';            // space
sentFrame[3] = chunkNo + 1;          // Current chunk number
sentFrame[4] = ' ';            // space

memcpy(&sentFrame[5], chunk, chunkLen);  // Copy chunk data to the frame

printf("Chunk No: %d\n", chunkNo + 1);  // Log the chunk number

// Send the frame
if (!sendToWss(sentFrame, sentFrameLen)) {
    heap_caps_free(chunk);
    heap_caps_free(sentFrame);
    percentage=0;
    return false;
}
```

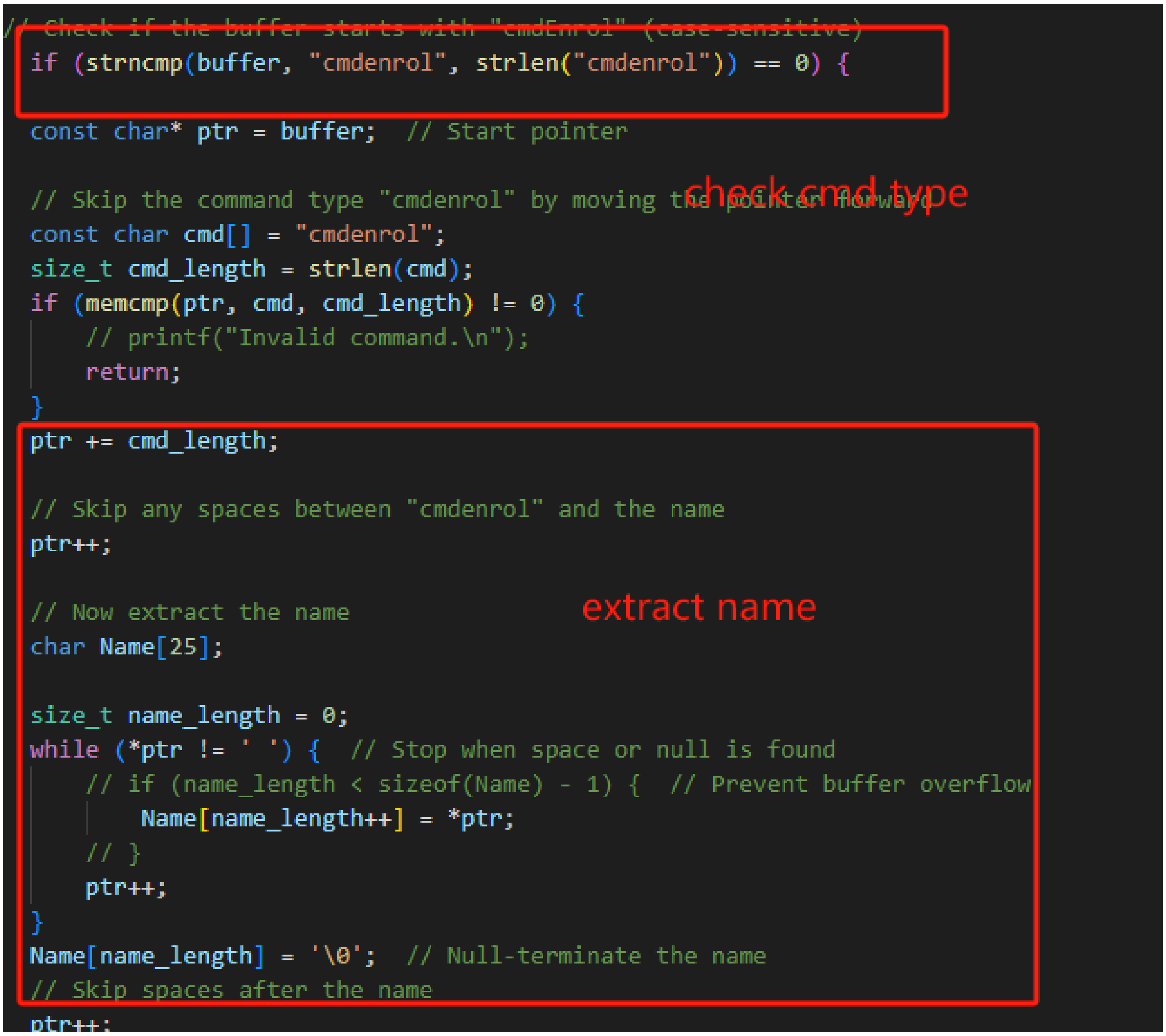
sent by chank

sent each chank



```
}else{

    ESP_LOGE(TAG, "Received: ");
    vTaskDelay(50);

    // for(uint16_t i=0; i< data->data_len;i++){

    //     printf("%x ",data->data_ptr[i]);

    // }
    process_command((char *)data->data_ptr);
    memset(data->data_ptr,0,data->data_len);

}
```

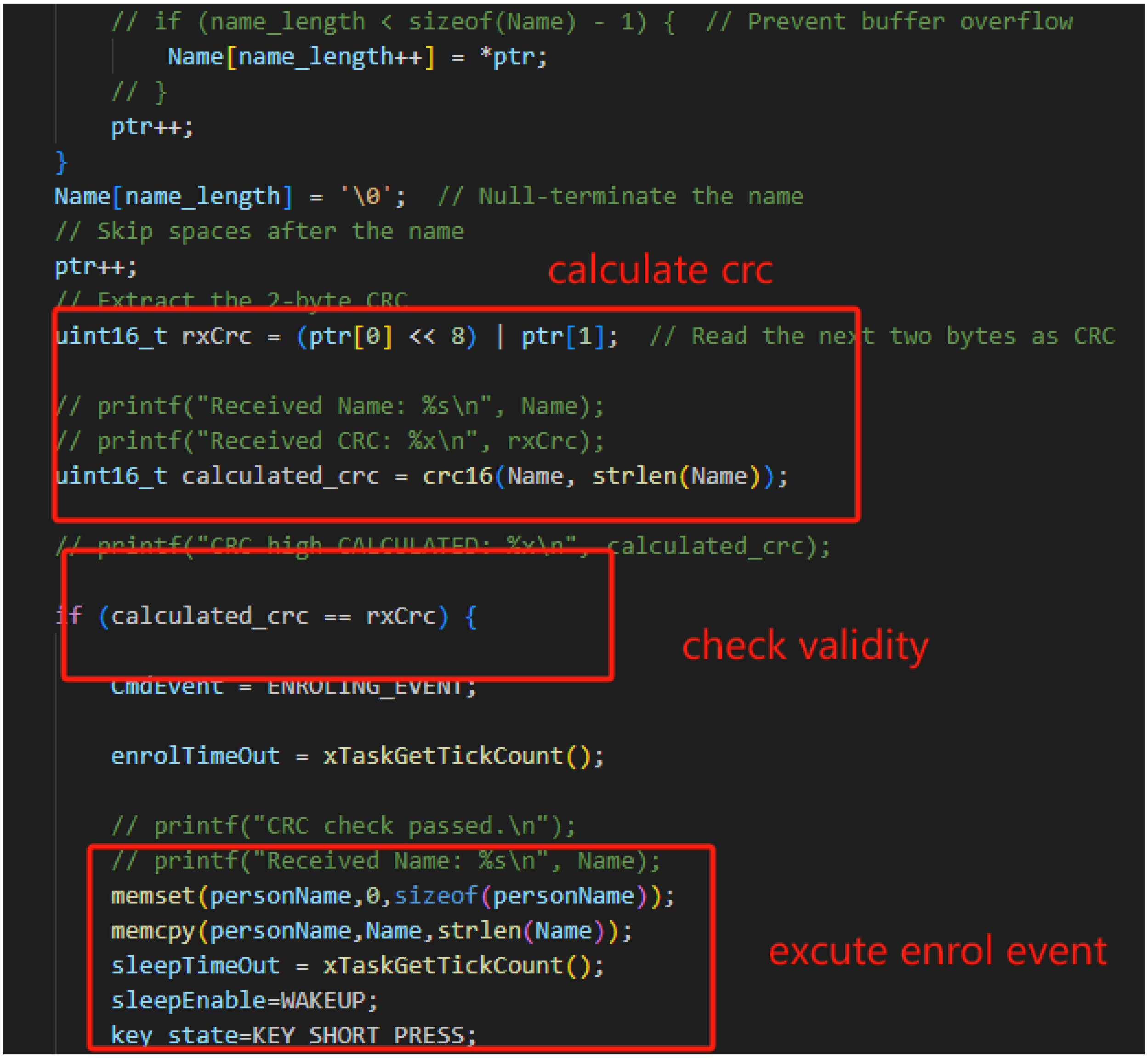
received data from wss



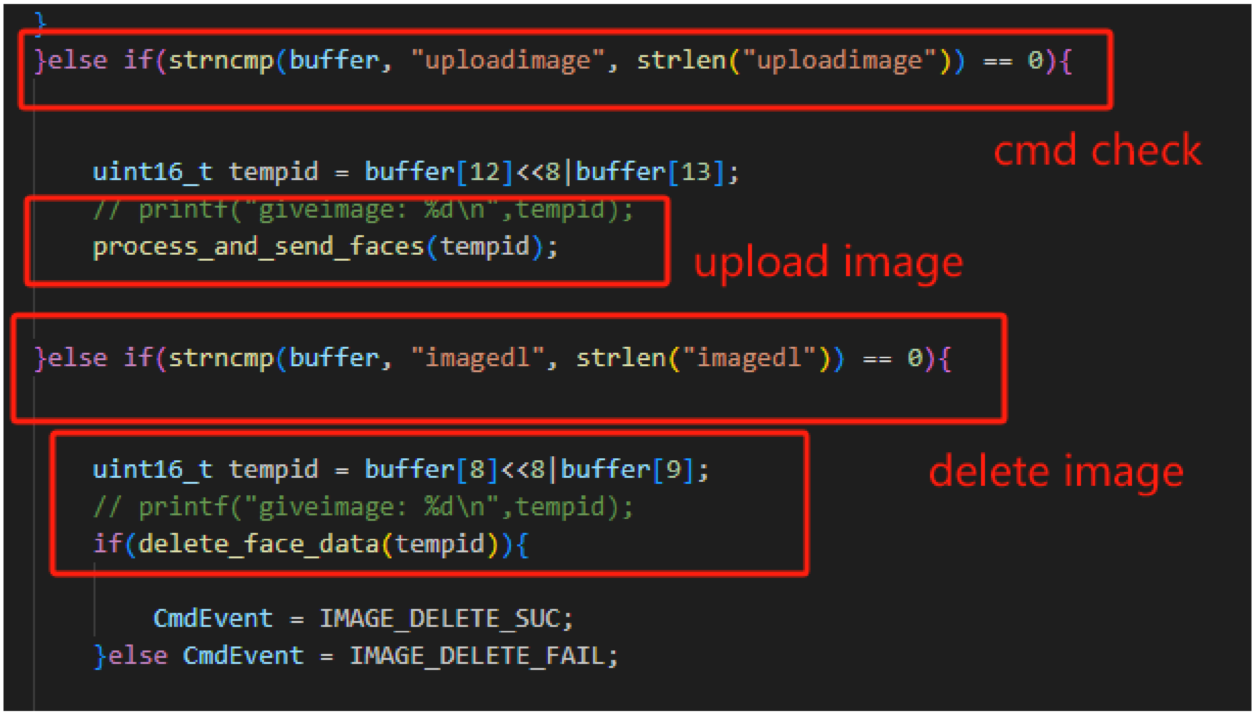
```
// Check if the buffer starts with "cmdEnrol" (case-sensitive)
if (strncmp(buffer, "cmdenrol", strlen("cmdenrol")) == 0) {

const char* ptr = buffer;  // Start pointer

// Skip the command type "cmdenrol" by moving the pointer forward
const char cmd[] = "cmdenrol";
size_t cmd_length = strlen(cmd);
if (memcmp(ptr, cmd, cmd_length) != 0) {
    // printf("Invalid command.\n");
    return;
}
ptr += cmd_length;

// Skip any spaces between "cmdenrol" and the name
ptr++;

// Now extract the name
char Name[25];

size_t name_length = 0;
while (*ptr != ' ') {  // Stop when space or null is found
    // if (name_length < sizeof(Name) - 1) {  // Prevent buffer overflow
        Name[name_length++] = *ptr;
    // }
    ptr++;
}
Name[name_length] = '\0';  // Null-terminate the name
// Skip spaces after the name
ptr++;
```

check cmd type

extract name



```
    // if (name_length < sizeof(Name) - 1) {  // Prevent buffer overflow
        Name[name_length++] = *ptr;
    // }
    ptr++;
}
Name[name_length] = '\0';  // Null-terminate the name
// Skip spaces after the name
ptr++;
// Extract the 2-byte CRC
uint16_t rxCrc = (ptr[0] << 8) | ptr[1];  // Read the next two bytes as CRC

// printf("Received Name: %s\n", Name);
// printf("Received CRC: %x\n", rxCrc);
uint16_t calculated_crc = crc16(Name, strlen(Name));

// printf("CRC high CALCULATED: %x\n", calculated_crc);

if (calculated_crc == rxCrc) {

    CmdEvent = ENROLING_EVENT;

    enrolTimeOut = xTaskGetTickCount();

    // printf("CRC check passed.\n");
    // printf("Received Name: %s\n", Name);
    memset(personName,0,sizeof(personName));
    memcpy(personName,Name,strlen(Name));
    sleepTimeOut = xTaskGetTickCount();
    sleepEnable=WAKEUP;
    key_state=KEY_SHORT_PRESS;
```

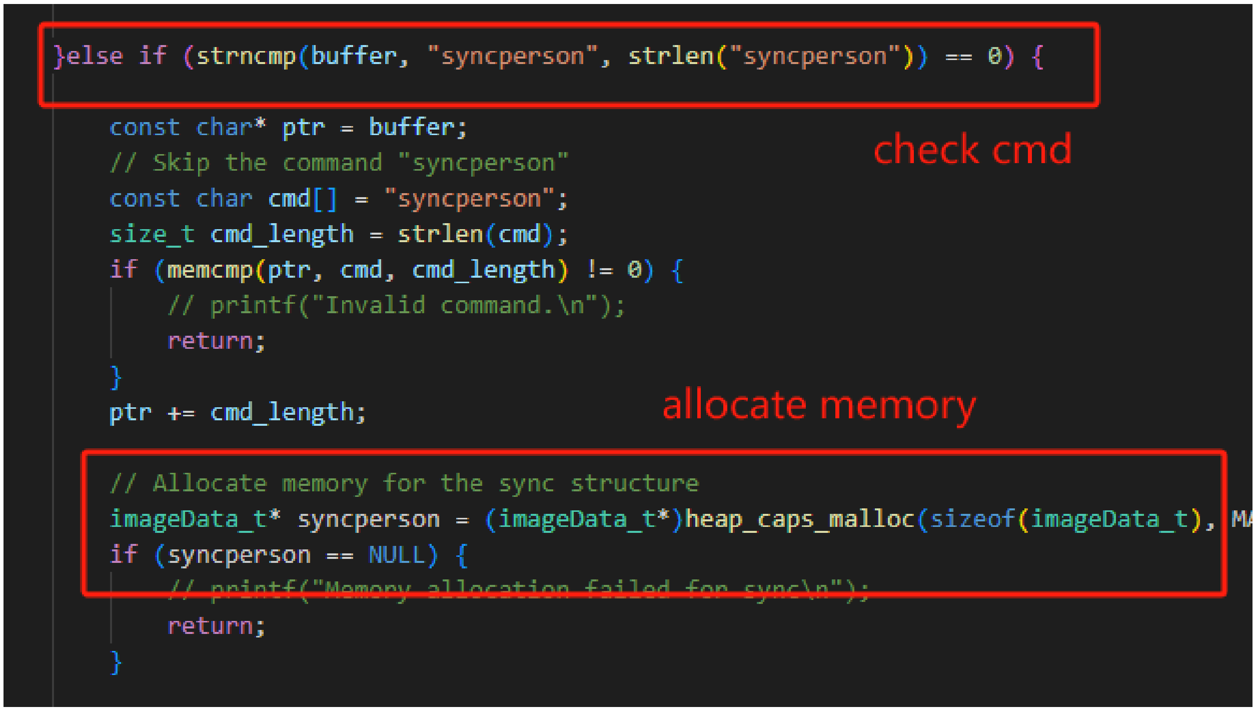


```
}
}else if(strncmp(buffer, "uploadimage", strlen("uploadimage")) == 0){

    uint16_t tempid = buffer[12]<<8|buffer[13];
    // printf("giveimage: %d\n",tempid);
    process_and_send_faces(tempid);

}else if(strncmp(buffer, "imagedl", strlen("imagedl")) == 0){

    uint16_t tempid = buffer[8]<<8|buffer[9];
    // printf("giveimage: %d\n",tempid);
    if(delete_face_data(tempid)){
        CmdEvent = IMAGE_DELETE_SUC;
    }else CmdEvent = IMAGE_DELETE_FAIL;
```

cmd check

upload image

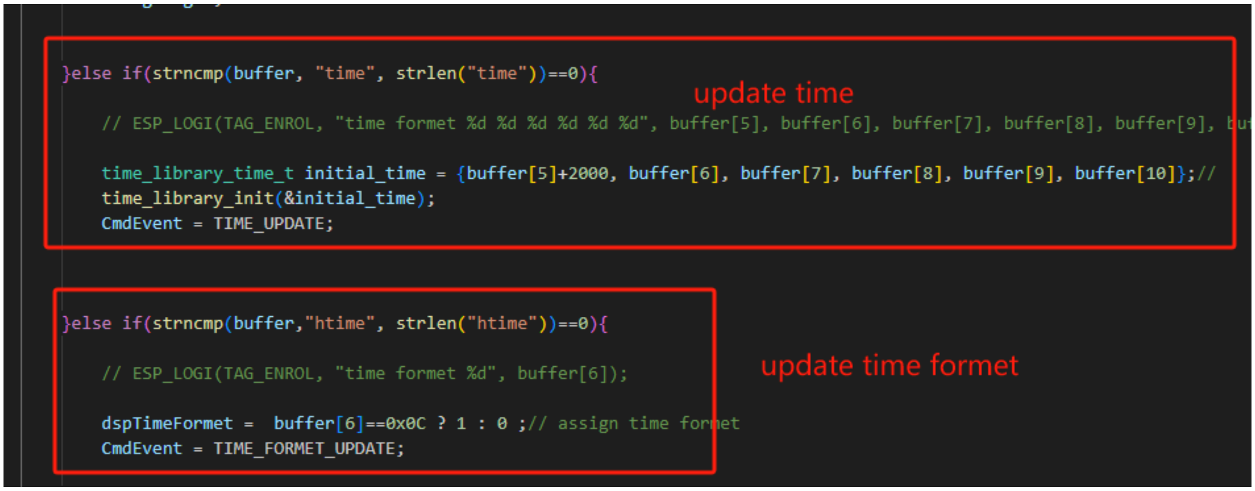
delete image



```
}else if (strncmp(buffer, "syncperson", strlen("syncperson")) == 0) {
    const char* ptr = buffer;
    // Skip the command "syncperson"
    const char cmd[] = "syncperson";
    size_t cmd_length = strlen(cmd);
    if (memcmp(ptr, cmd, cmd_length) != 0) {
        // printf("Invalid command.\n");
        return;
    }
    ptr += cmd_length;

    // Allocate memory for the sync structure
    imageData_t* syncperson = (imageData_t*)heap_caps_malloc(sizeof(imageData_t), MA
    if (syncperson == NULL) {
        // printf("Memory allocation failed for sync\n");
        return;
    }
```

check cmd

allocate memory



```
}else if(strncmp(buffer, "time", strlen("time"))==0){

    // ESP_LOGI(TAG_ENROL, "time formet %d %d %d %d %d %d", buffer[5], buffer[6], buffer[7], buffer[8], buffer[9], bu

    time_library_time_t initial_time = {buffer[5]+2000, buffer[6], buffer[7], buffer[8], buffer[9], buffer[10]};//
    time_library_init(&initial_time);
    CmdEvent = TIME_UPDATE;


}else if(strncmp(buffer,"htime", strlen("htime"))==0){

    // ESP_LOGI(TAG_ENROL, "time formet %d", buffer[6]);

    dspTimeFormet =  buffer[6]==0x0C ? 1 : 0 ;// assign time formet
    CmdEvent = TIME_FORMET_UPDATE;
```

update time

update time formet