black hat ASIA 2021

May 6-7, 2021

BRIEFINGS

Wideshears: Investigating and Breaking Widevine on QTEE

Qi Zhao

360 Alpha Lab

About the Team&Speaker

Qi Zhao

- Security Researcher at 360 Alpha Lab
- Focused on mobile platform
- Report vulnerabilities to Google, Huawei and Qualcomm
- a.k.a. Joachim Hyrathon @JHyrathon

360 Alpha Team

- More than 300 vulnerabilities acknowledged by top vendors
- Break the record of highest reward in ASR program twice
- Hold a record of 8 exploits by Google
- Successful pwner of several Pwn2Own and Tianfu Cup events

https://security.googleblog.com/2021/02/vulnerability-reward-program-2020-year.html

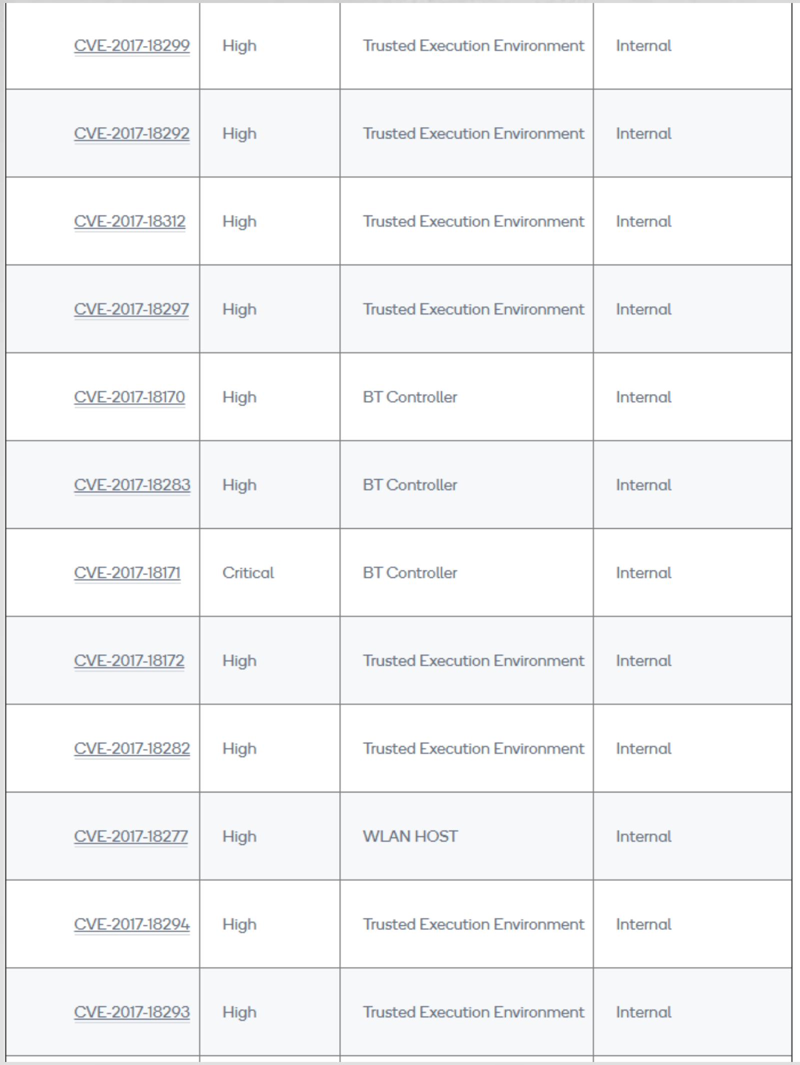
Agenda

- Introduction, backgrounds and basics
- Find vulnerabilities from Qualcomm TAs
- Understand the shared memory model
- Make the exploit work and extract Keybox from SFS
- Closing

What Makes Qualcomm's TEE a High-value Target

- Billions of devices running Qualcomm chipsets
- No successful exploit has been exposed since Gal Beniamini's excellent work 5 years ago
- Closed source
- Hard to profile or debug
- "Annihilation" in 2017-2018

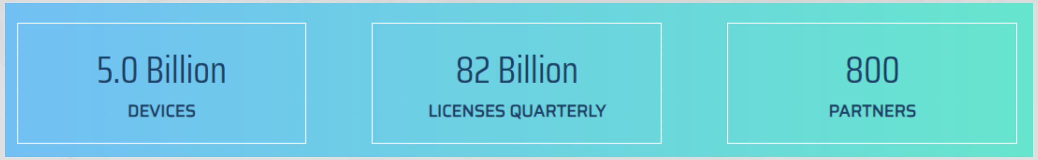
"Annihilation"



CVE-2017-18299	High	Trusted Execution Environment	Internal
CVE-2017-18292	High	Trusted Execution Environment	Internal
CVE-2017-18312	High	Trusted Execution Environment	Internal
CVE-2017-18297	High	Trusted Execution Environment	Internal
CVE-2017-18170	High	BT Controller	Internal
CVE-2017-18283	High	BT Controller	Internal
CVE-2017-18171	Critical	BT Controller	Internal
CVE-2017-18172	High	Trusted Execution Environment	Internal
CVE-2017-18282	High	Trusted Execution Environment	Internal
CVE-2017-18277	High	WLAN HOST	Internal
CVE-2017-18294	High	Trusted Execution Environment	Internal
CVE-2017-18293	High	Trusted Execution Environment	Internal

https://www.qualcomm.com/company/product-security/bulletins

What Makes Widevine TA a High-value Target



5.0 Billion	82 Billion	800
DEVICES	LICENSES QUARTERLY	PARTNERS

https://www.widevine.com/about

- Large amount of users & partners
- Affects many platforms
- De facto standard DRM solution for most Android OEMs/ODMs

TrustZone 101 in One Page

Purpose

- Trusted computing in untrusted environment
- Protect high-value content
- Observe Rich OS/Hypervisor (uncanny)

Features

- Hardware support
- Reuse Processors
- Secure/Non-Secure Switch
- Integrity Guaranteed by Secure Boot

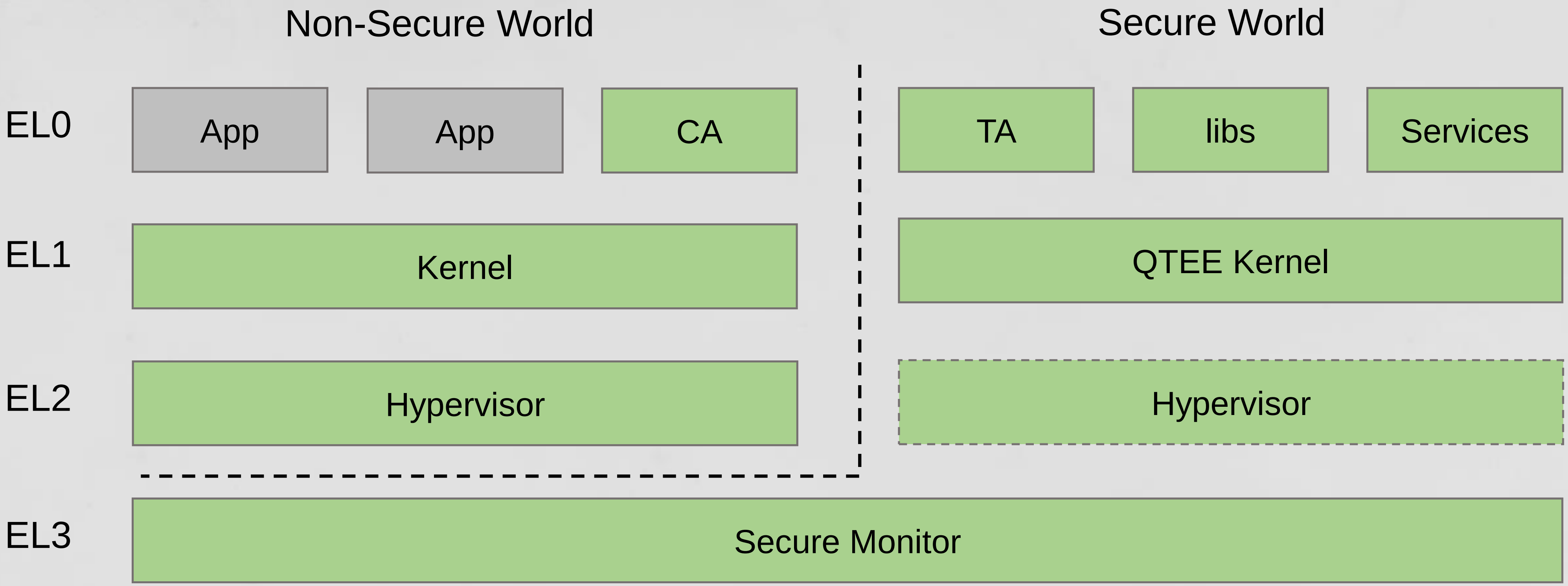
Possible Uses

- DRM
- Fingerprint
- Keystore

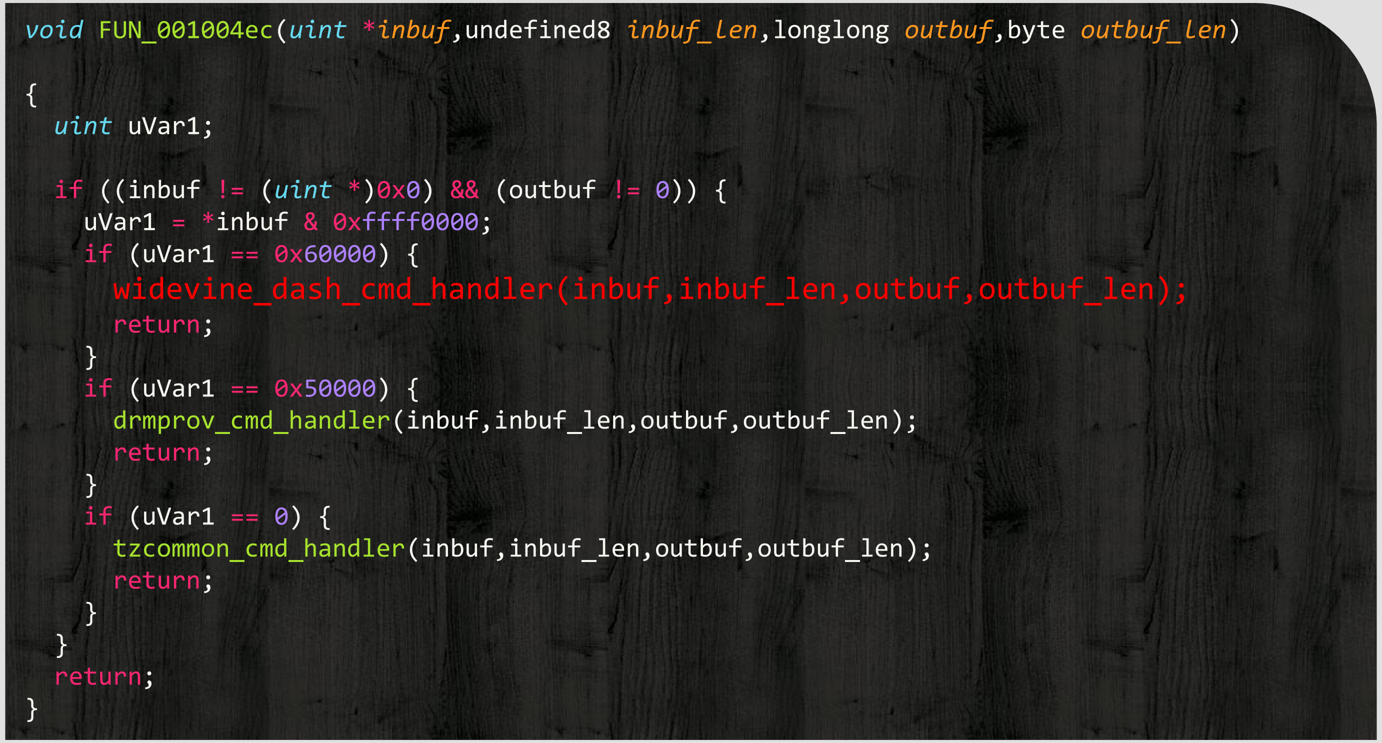
Curated lists on TEE Security:
https://github.com/enovella/TEE-reversing
https://github.com/doridori/Android-Security-Reference/blob/master/hardware/TEE/TEE.md

QTEE Architecture on Pixel 4 XL

(TZ.XF.5.2-225870, AARCH64)

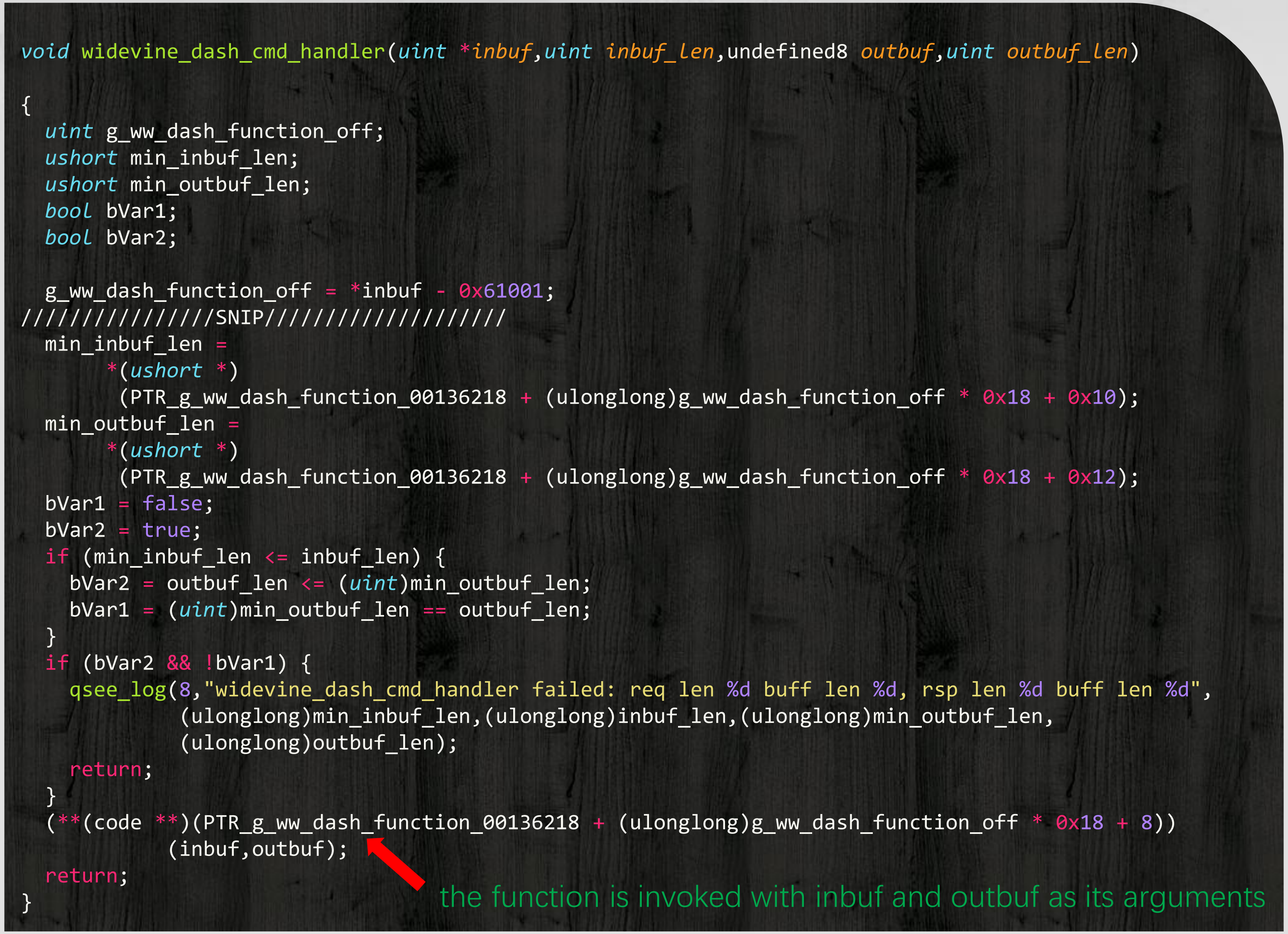


Widevine Command Dispatcher



```c
void FUN_001004ec(uint *inbuf,undefined8 inbuf_len,longlong outbuf,byte outbuf_len)

{
  uint uVar1;

  if ((inbuf != (uint *)0x0) && (outbuf != 0)) {
    uVar1 = *inbuf & 0xffff0000;
    if (uVar1 == 0x60000) {
      widevine_dash_cmd_handler(inbuf,inbuf_len,outbuf,outbuf_len);
      return;
    }
    if (uVar1 == 0x50000) {
      drmprov_cmd_handler(inbuf,inbuf_len,outbuf,outbuf_len);
      return;
    }
    if (uVar1 == 0) {
      tzcommon_cmd_handler(inbuf,inbuf_len,outbuf,outbuf_len);
      return;
    }
  }
  return;
}
```

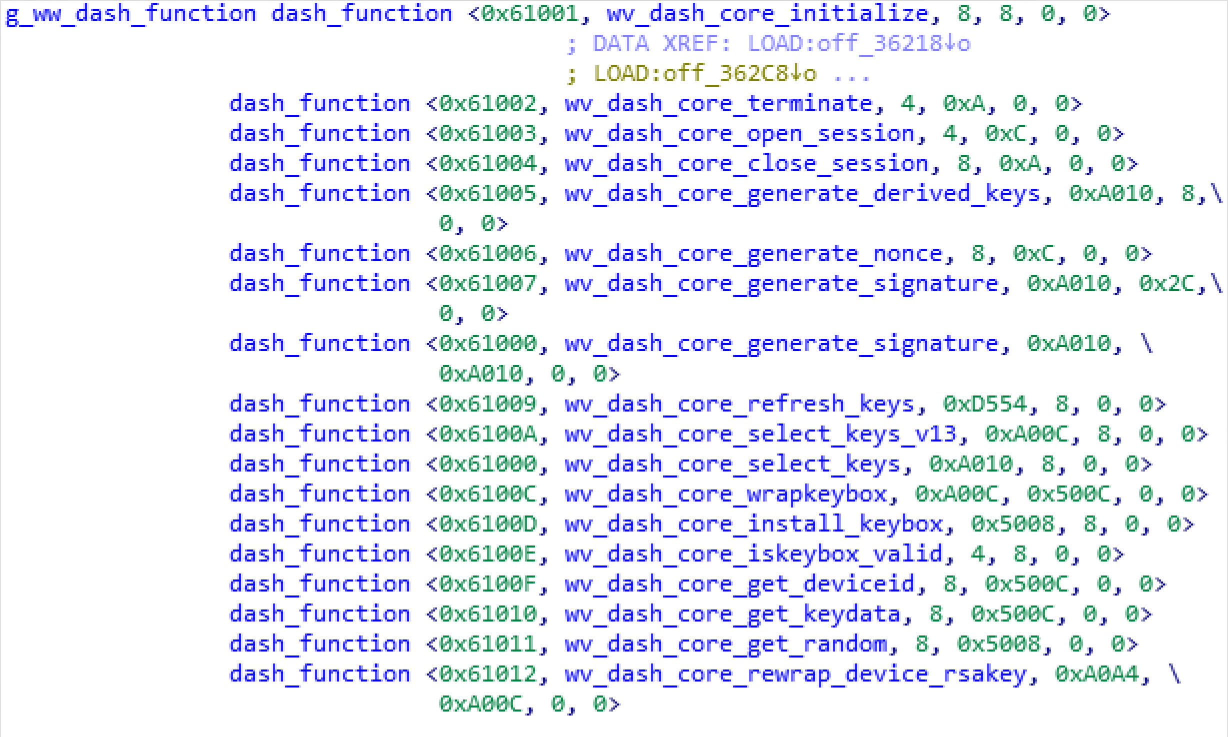
Widevine Dash Handler



```
void widevine_dash_cmd_handler(uint *inbuf,uint inbuf_len,undefined8 outbuf,uint outbuf_len)

{
  uint g_ww_dash_function_off;
  ushort min_inbuf_len;
  ushort min_outbuf_len;
  bool bVar1;
  bool bVar2;

  g_ww_dash_function_off = *inbuf - 0x61001;
////////////////SNIP///////////////////
  min_inbuf_len =
       *(ushort *)
        (PTR_g_ww_dash_function_00136218 + (ulonglong)g_ww_dash_function_off * 0x18 + 0x10);
  min_outbuf_len =
       *(ushort *)
        (PTR_g_ww_dash_function_00136218 + (ulonglong)g_ww_dash_function_off * 0x18 + 0x12);
  bVar1 = false;
  bVar2 = true;
  if (min_inbuf_len <= inbuf_len) {
    bVar2 = outbuf_len <= (uint)min_outbuf_len;
    bVar1 = (uint)min_outbuf_len == outbuf_len;
  }
  if (bVar2 && !bVar1) {
    qsee_log(8,"widevine_dash_cmd_handler failed: req len %d buff len %d, rsp len %d buff len %d",
             (ulonglong)min_inbuf_len,(ulonglong)inbuf_len,(ulonglong)min_outbuf_len,
             (ulonglong)outbuf_len);
    return;
  }
  (**(code **)(PTR_g_ww_dash_function_00136218 + (ulonglong)g_ww_dash_function_off * 0x18 + 8))
            (inbuf,outbuf);
  return;
}
```

the function is invoked with inbuf and outbuf as its arguments

The Function Table



```
g_ww_dash_function dash_function <0x61001, wv_dash_core_initialize, 8, 8, 0, 0>
                                        ; DATA XREF: LOAD:off_36218↓o
                                        ; LOAD:off_362C8↓o ...
                dash_function <0x61002, wv_dash_core_terminate, 4, 0xA, 0, 0>
                dash_function <0x61003, wv_dash_core_open_session, 4, 0xC, 0, 0>
                dash_function <0x61004, wv_dash_core_close_session, 8, 0xA, 0, 0>
                dash_function <0x61005, wv_dash_core_generate_derived_keys, 0xA010, 8,\
                               0, 0>
                dash_function <0x61006, wv_dash_core_generate_nonce, 8, 0xC, 0, 0>
                dash_function <0x61007, wv_dash_core_generate_signature, 0xA010, 0x2C,\
                               0, 0>
                dash_function <0x61000, wv_dash_core_generate_signature, 0xA010, \
                               0xA010, 0, 0>
                dash_function <0x61009, wv_dash_core_refresh_keys, 0xD554, 8, 0, 0>
                dash_function <0x6100A, wv_dash_core_select_keys_v13, 0xA00C, 8, 0, 0>
                dash_function <0x61000, wv_dash_core_select_keys, 0xA010, 8, 0, 0>
                dash_function <0x6100C, wv_dash_core_wrapkeybox, 0xA00C, 0x500C, 0, 0>
                dash_function <0x6100D, wv_dash_core_install_keybox, 0x5008, 8, 0, 0>
                dash_function <0x6100E, wv_dash_core_iskeybox_valid, 4, 8, 0, 0>
                dash_function <0x6100F, wv_dash_core_get_deviceid, 8, 0x500C, 0, 0>
                dash_function <0x61010, wv_dash_core_get_keydata, 8, 0x500C, 0, 0>
                dash_function <0x61011, wv_dash_core_get_random, 8, 0x5008, 0, 0>
                dash_function <0x61012, wv_dash_core_rewrap_device_rsakey, 0xA0A4, \
                               0xA00C, 0, 0>
```

- g_ww_dash_function is an array of function ptrs and cmd length bounds
- wv_dash_core_XXX(input_buffer, output_buffer)

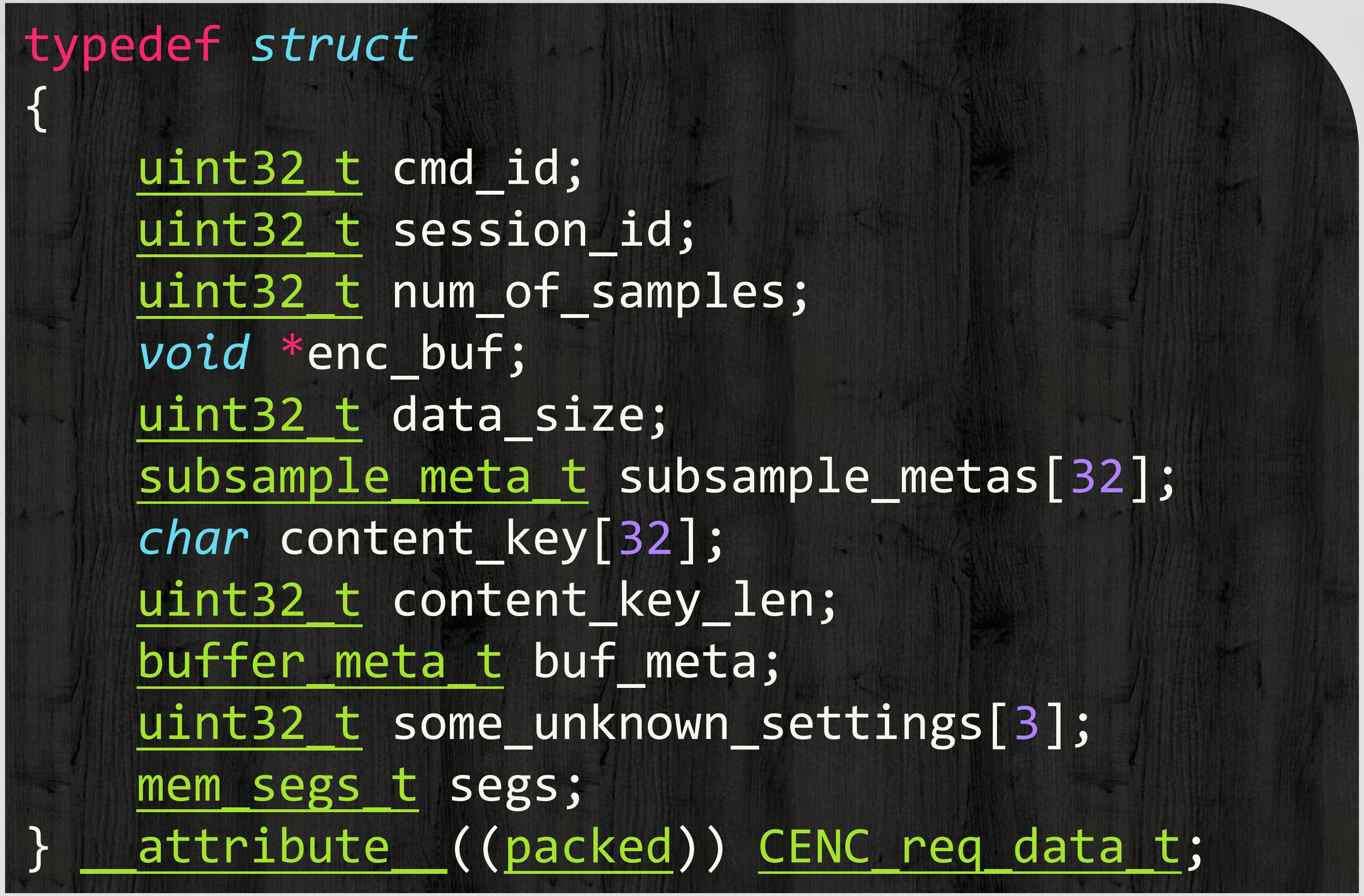
]

The First Vulnerability

wv_dash_core_decrypt_cenc()

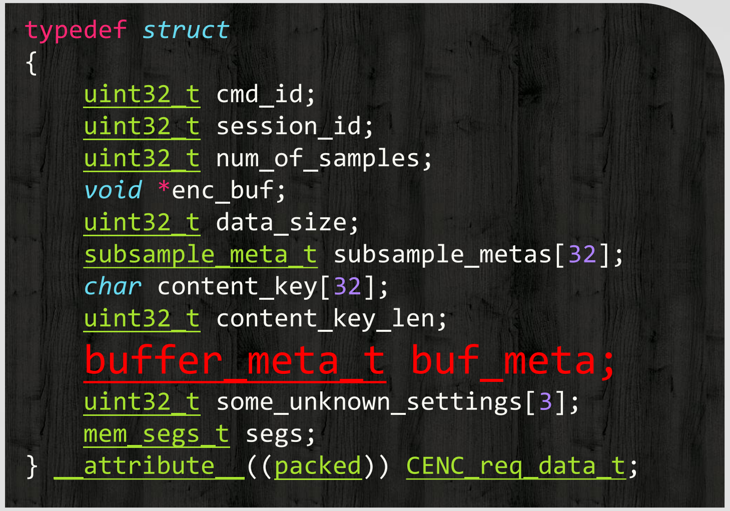
```
wv_dash_core_decrypt_cenc()
+
|
+---->wv_update_content_key()
|
|
+---->OEMCrypto_DecryptCENC()
      +
      |
      +---->alidate_register_io_buffers()
      |
      |
      +---->decrypt_CTR_unified()/decrypt_CBC_unified()
```

CENC Command(inbuf) Structure(Guessed)

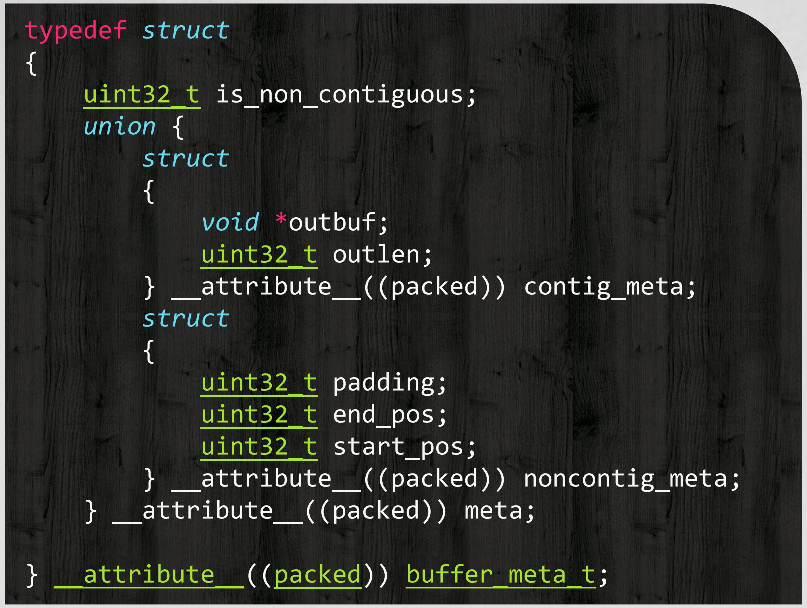


```c
typedef struct
{
    uint32_t cmd_id;
    uint32_t session_id;
    uint32_t num_of_samples;
    void *enc_buf;
    uint32_t data_size;
    subsample_meta_t subsample_metas[32];
    char content_key[32];
    uint32_t content_key_len;
    buffer_meta_t buf_meta;
    uint32_t some_unknown_settings[3];
    mem_segs_t segs;
} __attribute__((packed)) CENC_req_data_t;
```

Substructure: buf_meta, the Output Buffer

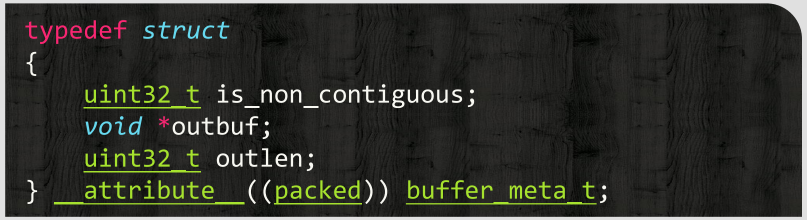


```
typedef struct
{
    uint32_t cmd_id;
    uint32_t session_id;
    uint32_t num_of_samples;
    void *enc_buf;
    uint32_t data_size;
    subsample_meta_t subsample_metas[32];
    char content_key[32];
    uint32_t content_key_len;
    buffer_meta_t buf_meta;
    uint32_t some_unknown_settings[3];
    mem_segs_t segs;
} __attribute__((packed)) CENC_req_data_t;
```



```
typedef struct
{
    uint32_t is_non_contiguous;
    union {
        struct
        {
            void *outbuf;
            uint32_t outlen;
        } __attribute__((packed)) contig_meta;
        struct
        {
            uint32_t padding;
            uint32_t end_pos;
            uint32_t start_pos;
        } __attribute__((packed)) noncontig_meta;
    } __attribute__((packed)) meta;

} __attribute__((packed)) buffer_meta_t;
```

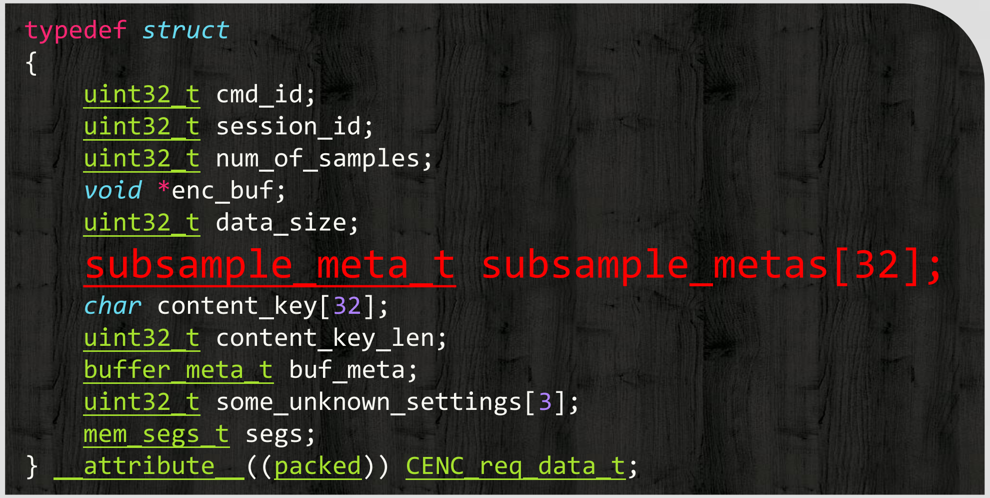
Support both
- physical contiguous
- non-contiguous(scatter list based) buffers



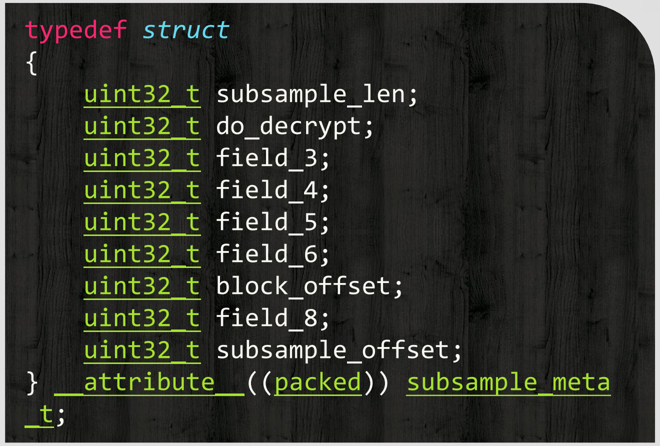
```
typedef struct
{
    uint32_t is_non_contiguous;
    void *outbuf;
    uint32_t outlen;
} __attribute__((packed)) buffer_meta_t;
```

Contiguous situation only

Substructure: subsample_metas



```
typedef struct
{
    uint32_t cmd_id;
    uint32_t session_id;
    uint32_t num_of_samples;
    void *enc_buf;
    uint32_t data_size;
    subsample_meta_t subsample_metas[32];
    char content_key[32];
    uint32_t content_key_len;
    buffer_meta_t buf_meta;
    uint32_t some_unknown_settings[3];
    mem_segs_t segs;
} __attribute__((packed)) CENC_req_data_t;
```

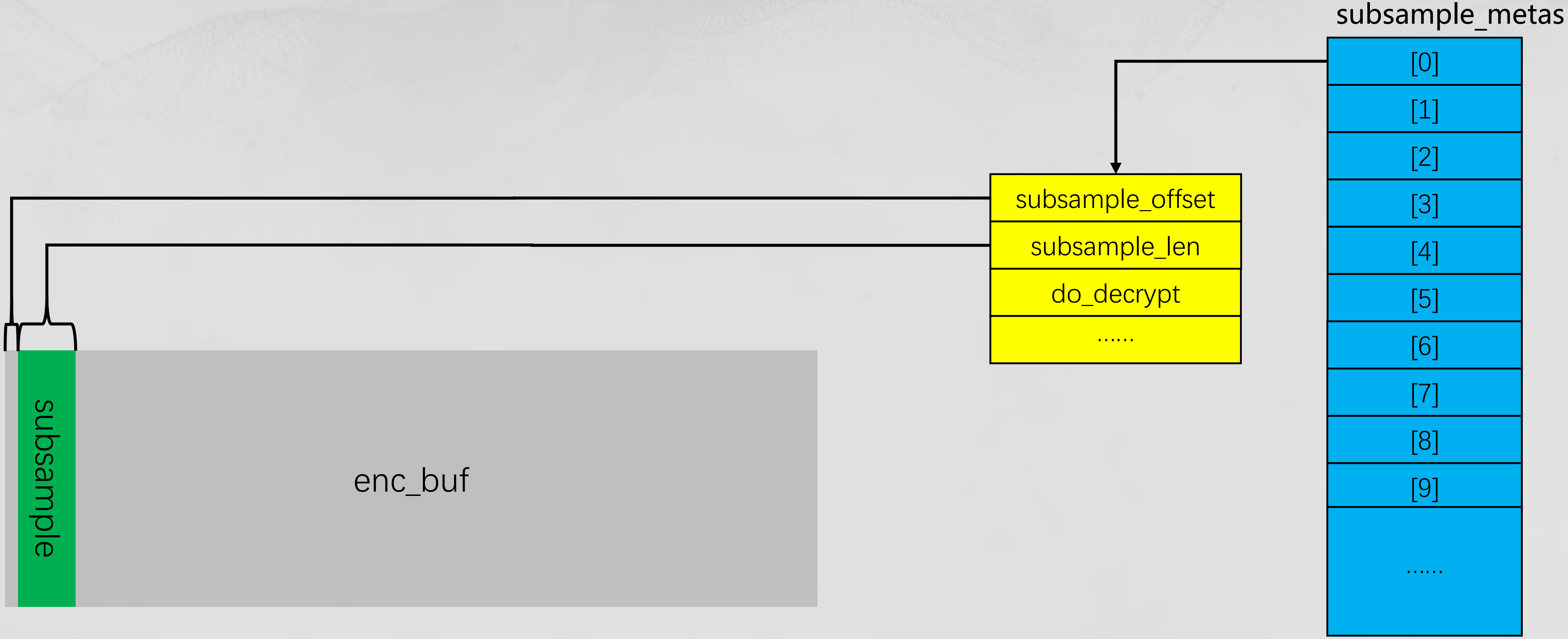


```
typedef struct
{
    uint32_t subsample_len;
    uint32_t do_decrypt;
    uint32_t field_3;
    uint32_t field_4;
    uint32_t field_5;
    uint32_t field_6;
    uint32_t block_offset;
    uint32_t field_8;
    uint32_t subsample_offset;
} __attribute__((packed)) subsample_meta
_t;
```

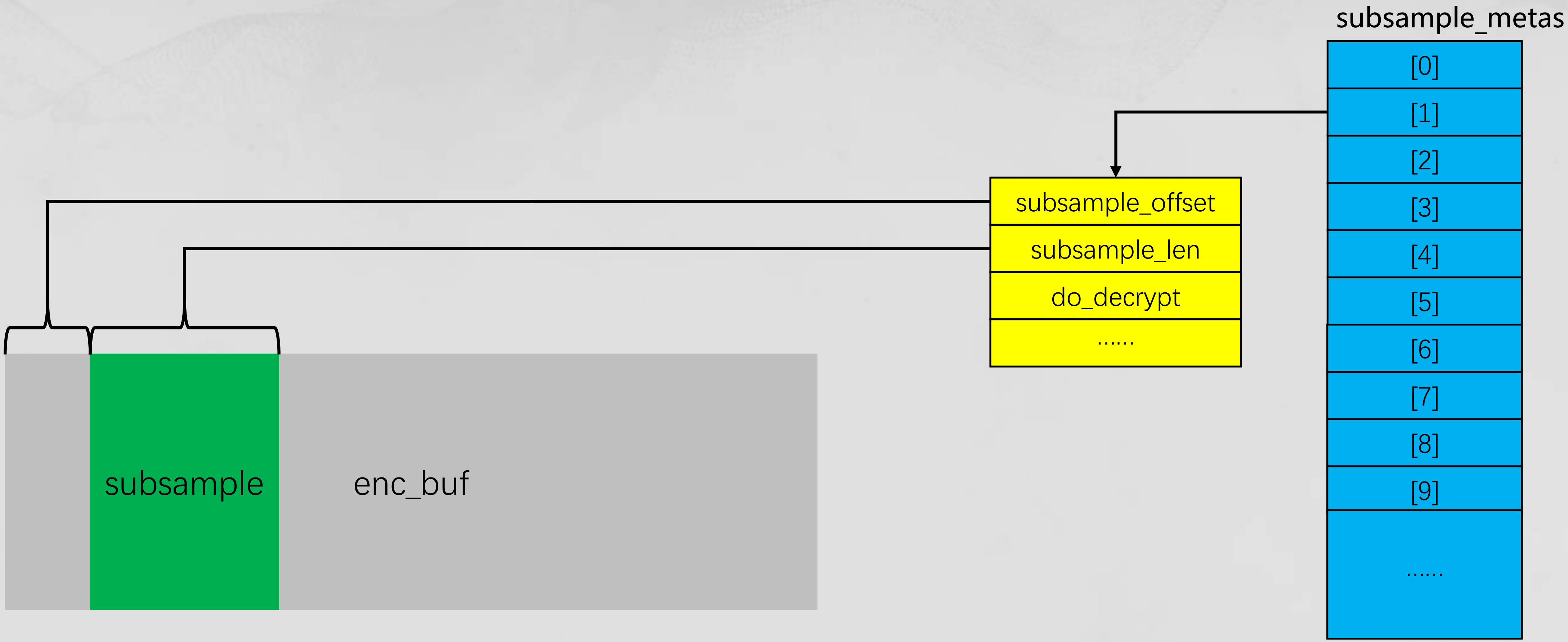
Restored some of the metadata fields

So How are "subsamples" Processed?

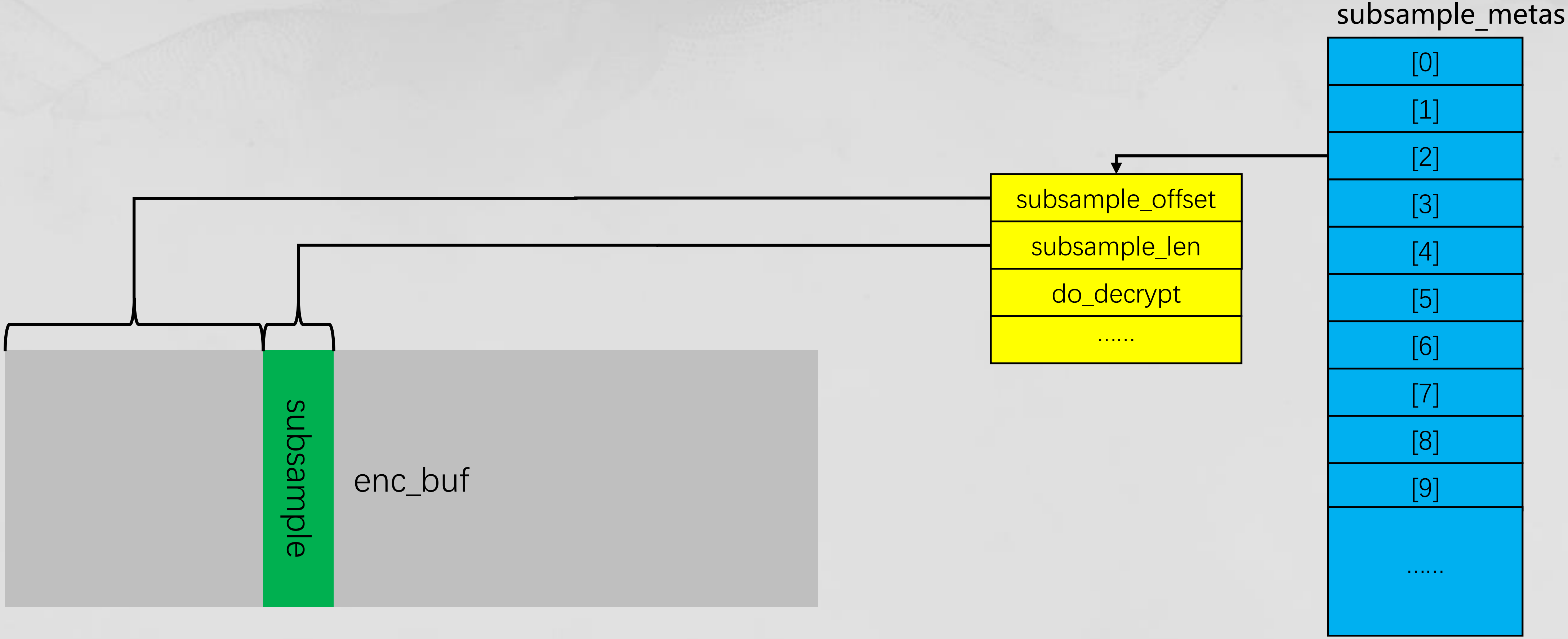
Subsample, Locating.



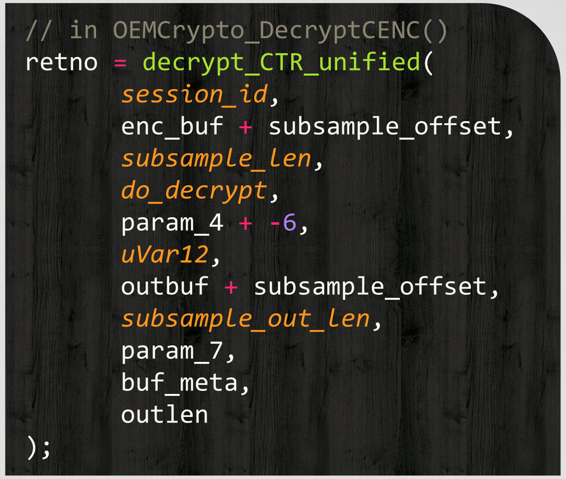
Subsample, Locating..



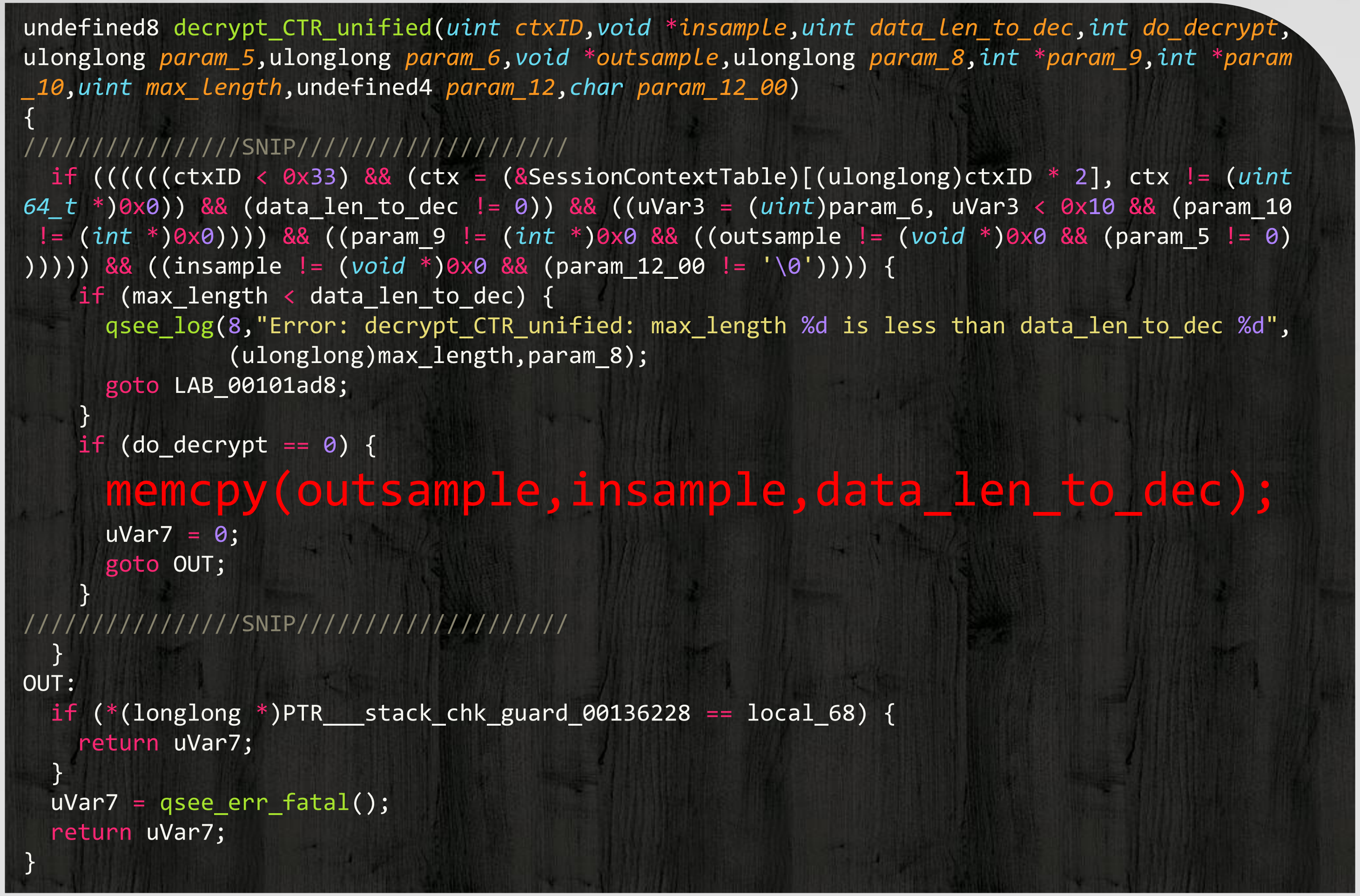
Subsample, Locating...



Subsample, Decryption

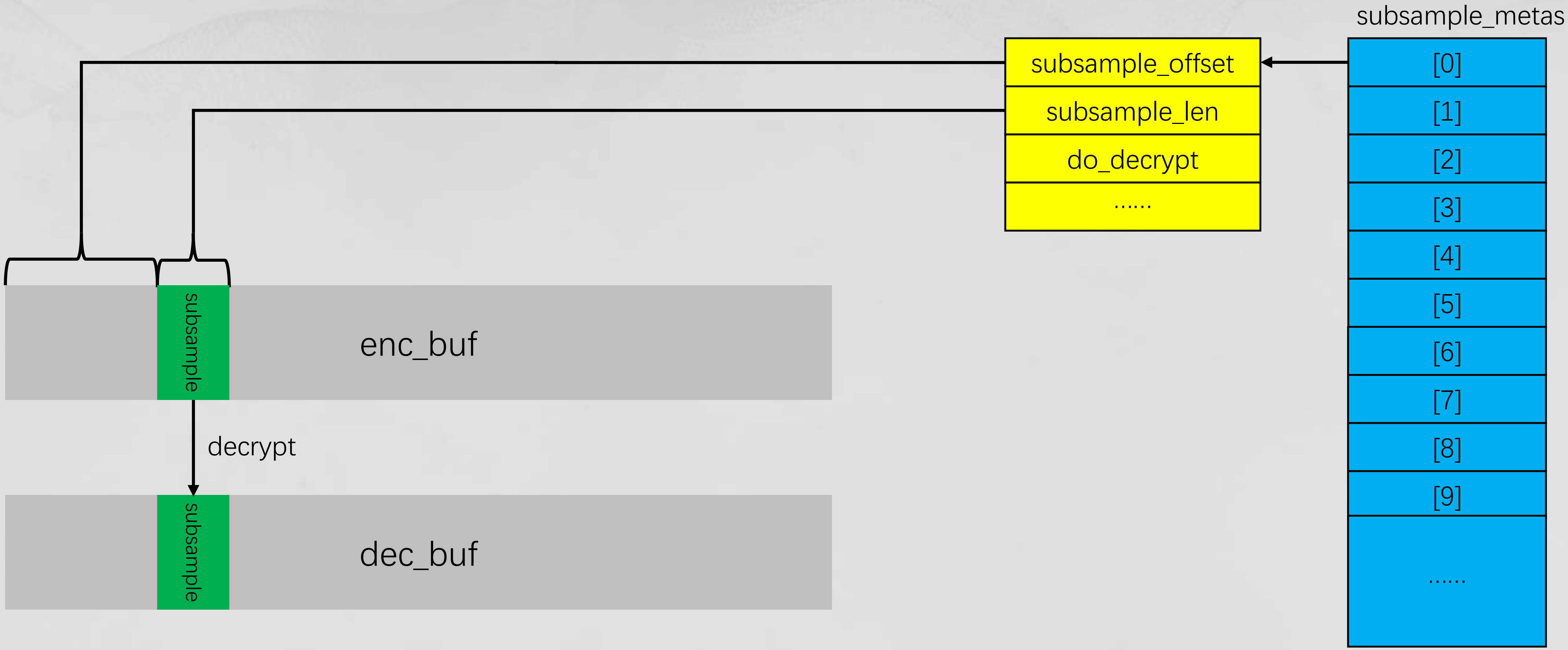


```
// in OEMCrypto_DecryptCENC()
retno = decrypt_CTR_unified(
        session_id,
        enc_buf + subsample_offset,
        subsample_len,
        do_decrypt,
        param_4 + -6,
        uVar12,
        outbuf + subsample_offset,
        subsample_out_len,
        param_7,
        buf_meta,
        outlen
);
```



```
undefined8 decrypt_CTR_unified(uint ctxID,void *insample,uint data_len_to_dec,int do_decrypt,
ulonglong param_5,ulonglong param_6,void *outsample,ulonglong param_8,int *param_9,int *param
_10,uint max_length,undefined4 param_12,char param_12_00)
{
////////////////SNIP////////////////////
  if ((((((ctxID < 0x33) && (ctx = (&SessionContextTable)[(ulonglong)ctxID * 2], ctx != (uint
64_t *)0x0)) && (data_len_to_dec != 0)) && ((uVar3 = (uint)param_6, uVar3 < 0x10 && (param_10
 != (int *)0x0)))) && ((param_9 != (int *)0x0 && ((outsample != (void *)0x0 && (param_5 != 0)
))))) && ((insample != (void *)0x0 && (param_12_00 != '\0')))) {
    if (max_length < data_len_to_dec) {
      qsee_log(8,"Error: decrypt_CTR_unified: max_length %d is less than data_len_to_dec %d",
               (ulonglong)max_length,param_8);
      goto LAB_00101ad8;
    }
    if (do_decrypt == 0) {
      memcpy(outsample,insample,data_len_to_dec);
      uVar7 = 0;
      goto OUT;
    }
////////////////SNIP////////////////////
  }
OUT:
  if (*(longlong *)PTR___stack_chk_guard_00136228 == local_68) {
    return uVar7;
  }
  uVar7 = qsee_err_fatal();
  return uVar7;
}
```

Subsample, Decryption

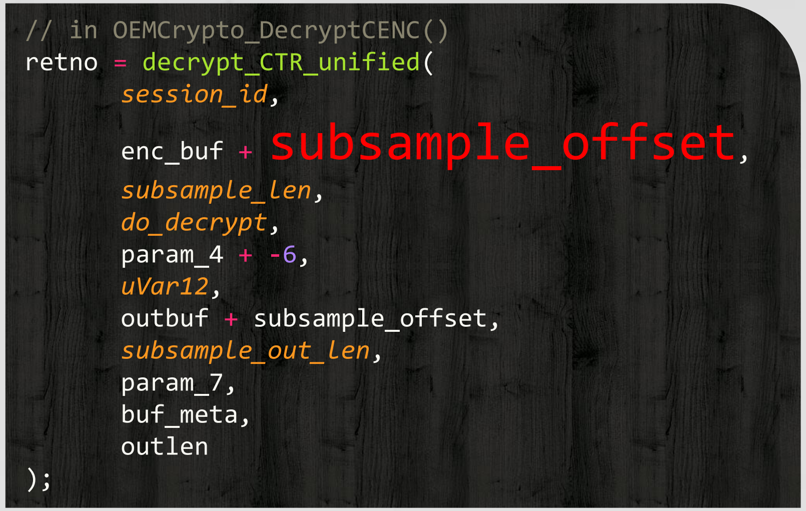


Subsample, Summary

- Embedded in enc_buf
- Length and offset are from subsample_metas
- when do_decrypt == 0, decryption will demote to memcpy()

Got a sense of vulnerability?

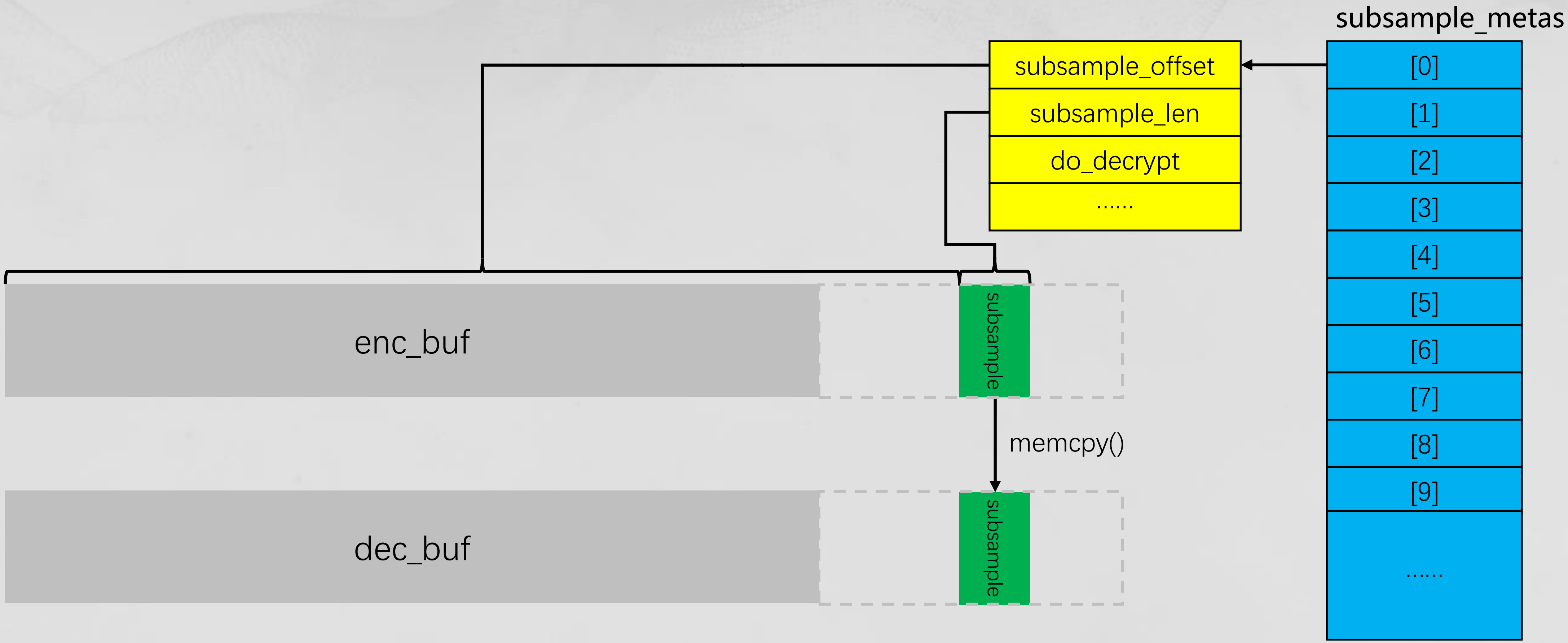
The Vulnerability



```
// in OEMCrypto_DecryptCENC()
retno = decrypt_CTR_unified(
        session_id,
        enc_buf + subsample_offset,
        subsample_len,
        do_decrypt,
        param_4 + -6,
        uVar12,
        outbuf + subsample_offset,
        subsample_out_len,
        param_7,
        buf_meta,
        outlen
);
```

- No bound check for subsample_offset

The Vulnerability



What's Next

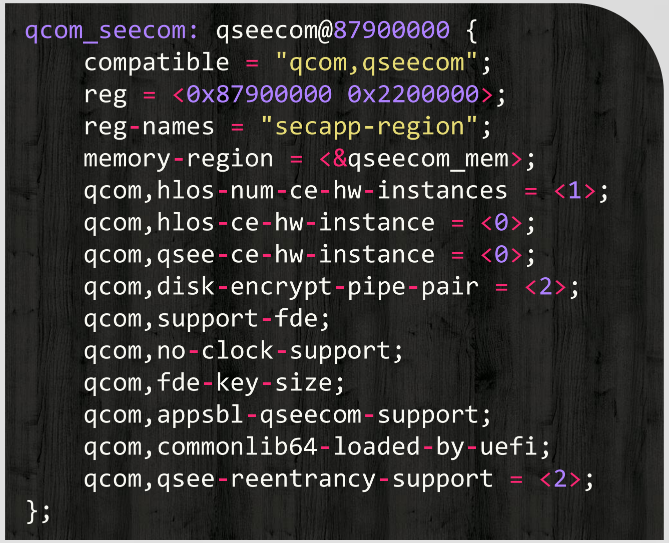
What we have

- ✓ Accurate memcpy() to single byte
- ✓ subsample_offset is a 32-bit value, not enough to cause integer overflow on 64-bit system

What we need

- □ Address of TA in memory
- □ Address of user controlled enc_buf and dec_buf in TA's view
- □ Delicate layout that lets the memory corrupt reach TA

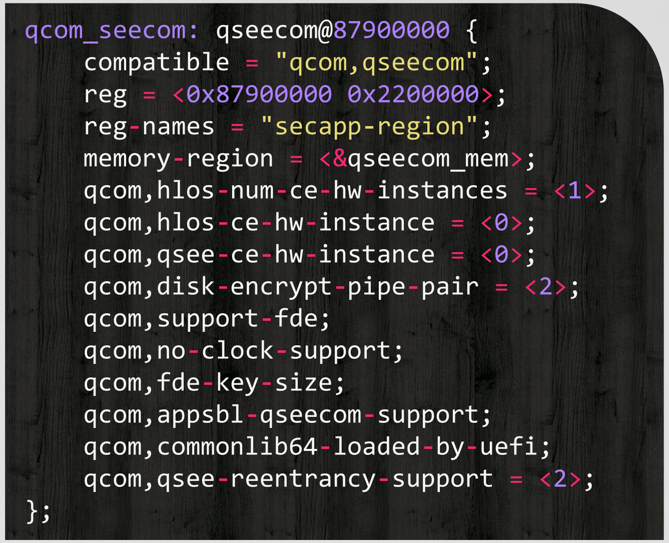
TA in Memory



```
qcom_seecom: qseecom@87900000 {
    compatible = "qcom,qseecom";
    reg = <0x87900000 0x2200000>;
    reg-names = "secapp-region";
    memory-region = <&qseecom_mem>;
    qcom,hlos-num-ce-hw-instances = <1>;
    qcom,hlos-ce-hw-instance = <0>;
    qcom,qsee-ce-hw-instance = <0>;
    qcom,disk-encrypt-pipe-pair = <2>;
    qcom,support-fde;
    qcom,no-clock-support;
    qcom,fde-key-size;
    qcom,appsbl-qseecom-support;
    qcom,commonlib64-loaded-by-uefi;
    qcom,qsee-reentrancy-support = <2>;
};
```

- Defined in a DTS file, preallocated secapp-region physical region for TAs
- Linear map, pa==va

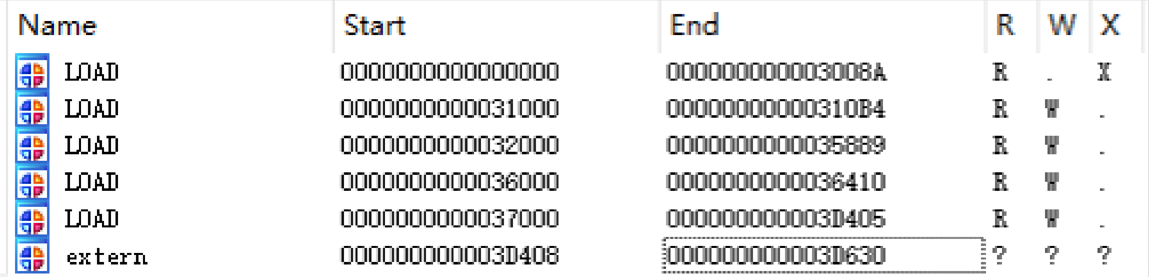
Bypass ASLR



```
qcom_seecom: qseecom@87900000 {
    compatible = "qcom,qseecom";
    reg = <0x87900000 0x2200000>;
    reg-names = "secapp-region";
    memory-region = <&qseecom_mem>;
    qcom,hlos-num-ce-hw-instances = <1>;
    qcom,hlos-ce-hw-instance = <0>;
    qcom,qsee-ce-hw-instance = <0>;
    qcom,disk-encrypt-pipe-pair = <2>;
    qcom,support-fde;
    qcom,no-clock-support;
    qcom,fde-key-size;
    qcom,appsbl-qseecom-support;
    qcom,commonlib64-loaded-by-uefi;
    qcom,qsee-reentrancy-support = <2>;
};
```

- secapp-region is limited
- pa==va
- The ASLR is easy to break

Bypass ASLR



Name	Start	End	R	W	X
LOAD	0000000000000000	000000000003008A	R	.	X
LOAD	0000000000031000	00000000000310B4	R	W	.
LOAD	0000000000032000	0000000000035889	R	W	.
LOAD	0000000000036000	0000000000036410	R	W	.
LOAD	0000000000037000	000000000003D405	R	W	.
extern	000000000003D408	000000000003D630	?	?	?

- If we have a read primitive, we have a ≈62/8704 chance to hit a page belongs to TA
- Brute-force: Keep trying to read from a page till TA process doesn't crash
- Prepare signatures to identify the page we hit

What's Next

What we have

- ✓ Accurate memcpy() to single byte
- ✓ subsample_offset is a 32-bit value, not enough to cause integer overflow on 64-bit system

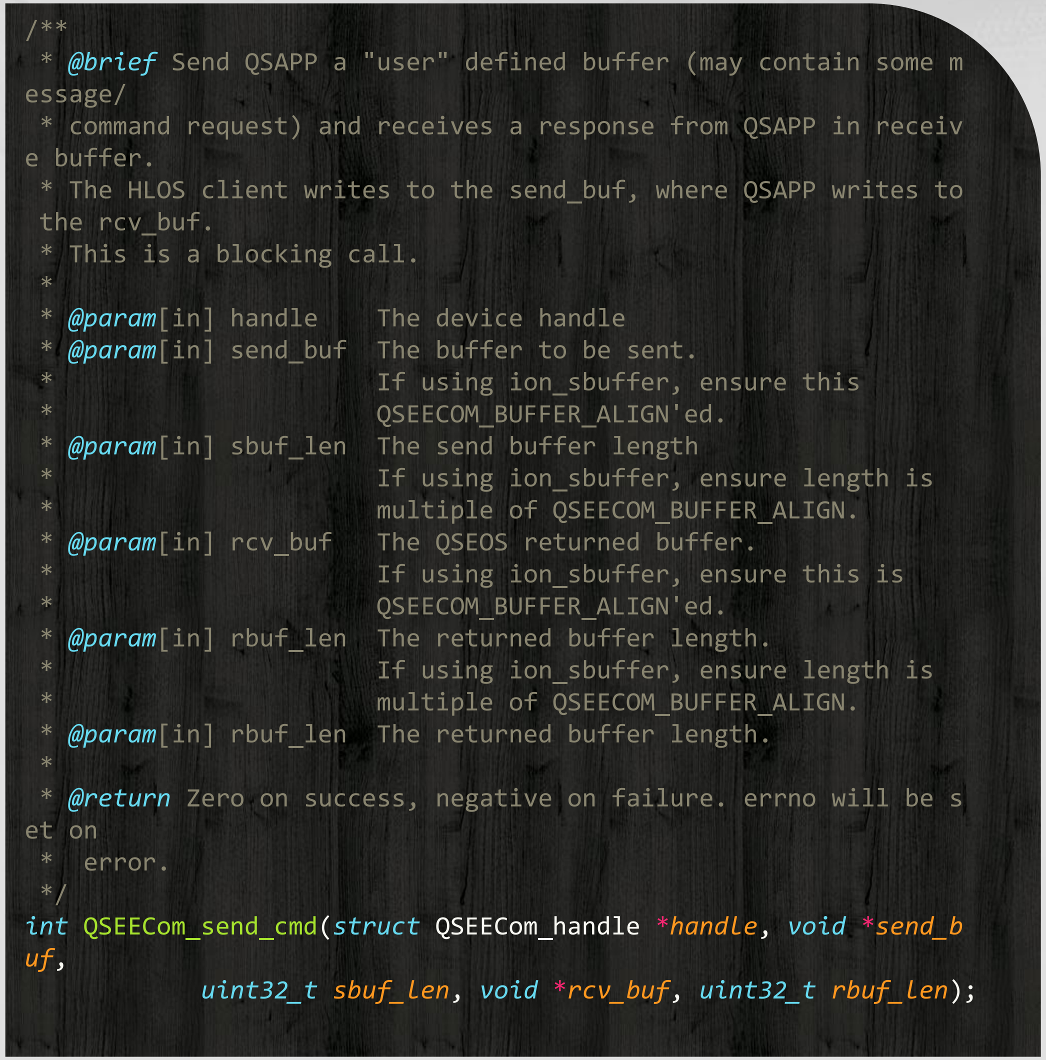
What we need

- ✓ Address of TA in memory
- □ Address of user controlled enc_buf and dec_buf in TA's view
- □ Delicate layout that lets the memory corrupt reach TA

enc_buf and dec_buf are shared buffers.

How to shared them to TA?

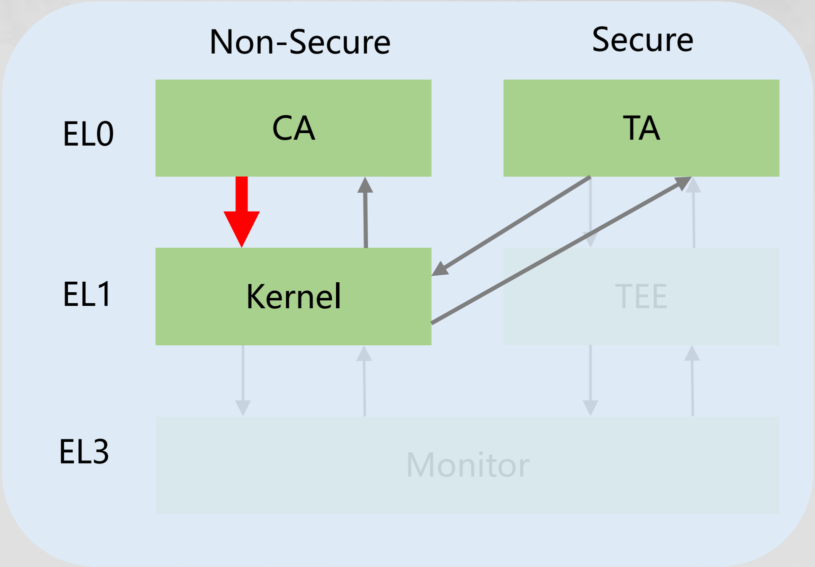
Send Commands to TA from Userspace



```
/**
 * @brief Send QSAPP a "user" defined buffer (may contain some m
essage/
 * command request) and receives a response from QSAPP in receiv
e buffer.
 * The HLOS client writes to the send_buf, where QSAPP writes to
 the rcv_buf.
 * This is a blocking call.
 *
 * @param[in] handle    The device handle
 * @param[in] send_buf  The buffer to be sent.
 *                      If using ion_sbuffer, ensure this
 *                      QSEECOM_BUFFER_ALIGN'ed.
 * @param[in] sbuf_len  The send buffer length
 *                      If using ion_sbuffer, ensure length is
 *                      multiple of QSEECOM_BUFFER_ALIGN.
 * @param[in] rcv_buf   The QSEOS returned buffer.
 *                      If using ion_sbuffer, ensure this is
 *                      QSEECOM_BUFFER_ALIGN'ed.
 * @param[in] rbuf_len  The returned buffer length.
 *                      If using ion_sbuffer, ensure length is
 *                      multiple of QSEECOM_BUFFER_ALIGN.
 * @param[in] rbuf_len  The returned buffer length.
 *
 * @return Zero on success, negative on failure. errno will be s
et on
 *  error.
 */
int QSEECom_send_cmd(struct QSEECom_handle *handle, void *send_b
uf,
            uint32_t sbuf_len, void *rcv_buf, uint32_t rbuf_len);
```

- send_buf contains commands and other data to TA
- rcv_buf contains response from TA

Send Commands to TA



send_buf

- CMD
- Field 1
- Field 2
- ……
- Field n

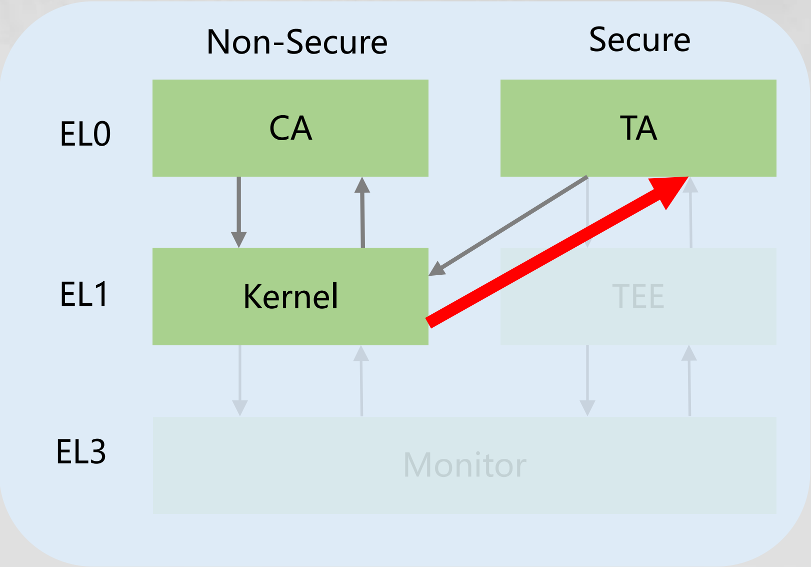
rcv_buf

- CMD
- Field 1
- Field 2
- ……
- Field n

🟠 data from CA

🔵 data from TA

Send Commands to TA



send_buf

- CMD
- Field 1
- Field 2
- ......
- Field n

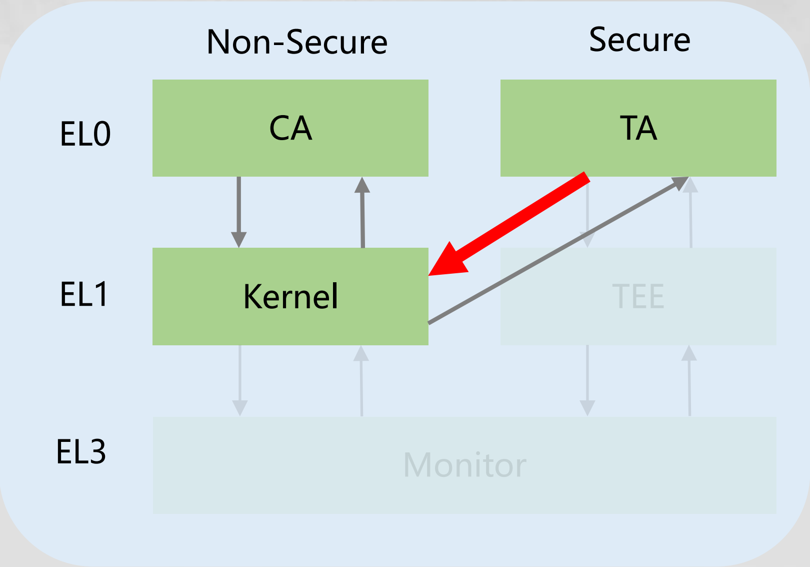
rcv_buf

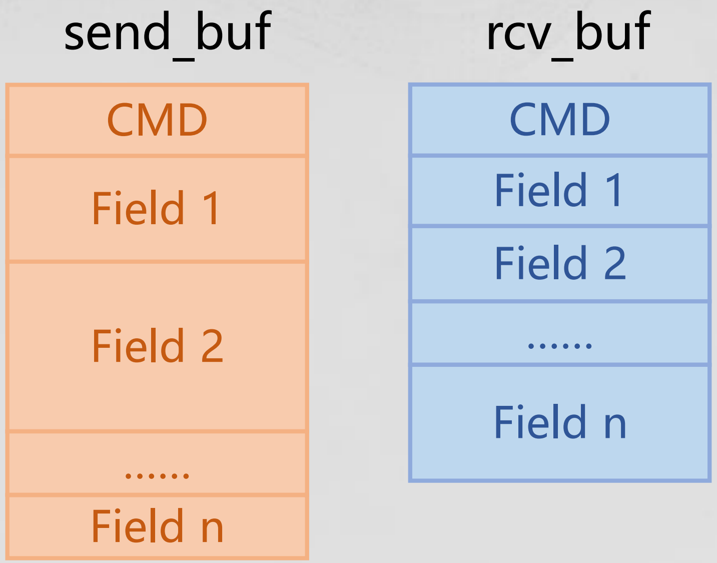
- CMD
- Field 1
- Field 2
- ......
- Field n

● data from CA

● data from TA

Send Commands to TA

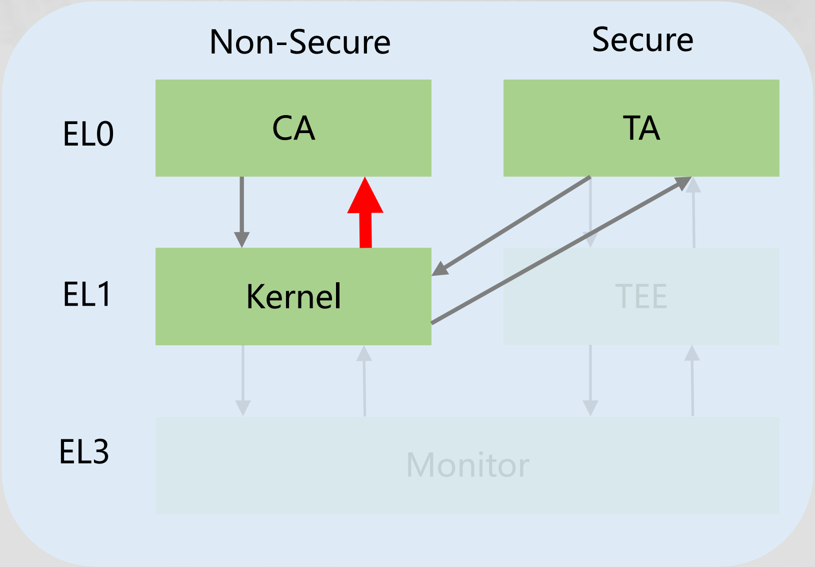




● data from CA

● data from TA

Send Commands to TA



send_buf

- CMD
- Field 1
- Field 2
- ......
- Field n

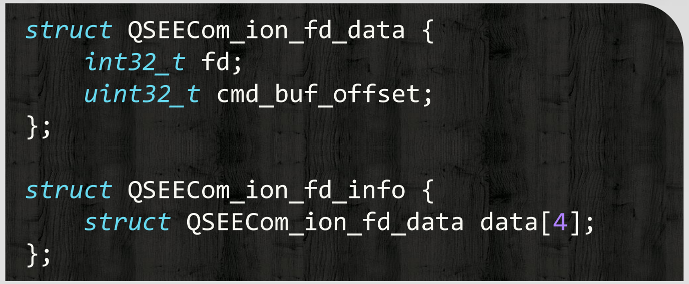
rcv_buf

- CMD
- Field 1
- Field 2
- ......
- Field n

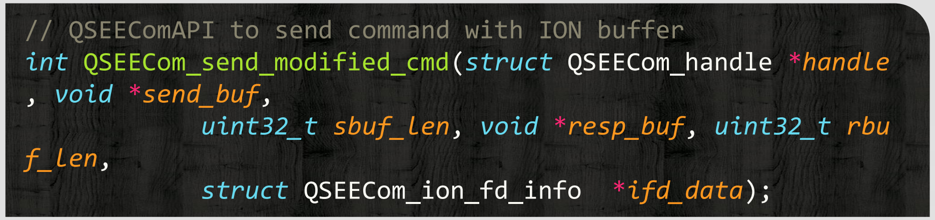
🟠 data from CA

🔵 data from TA

Send Commands to TA with Shared Memory



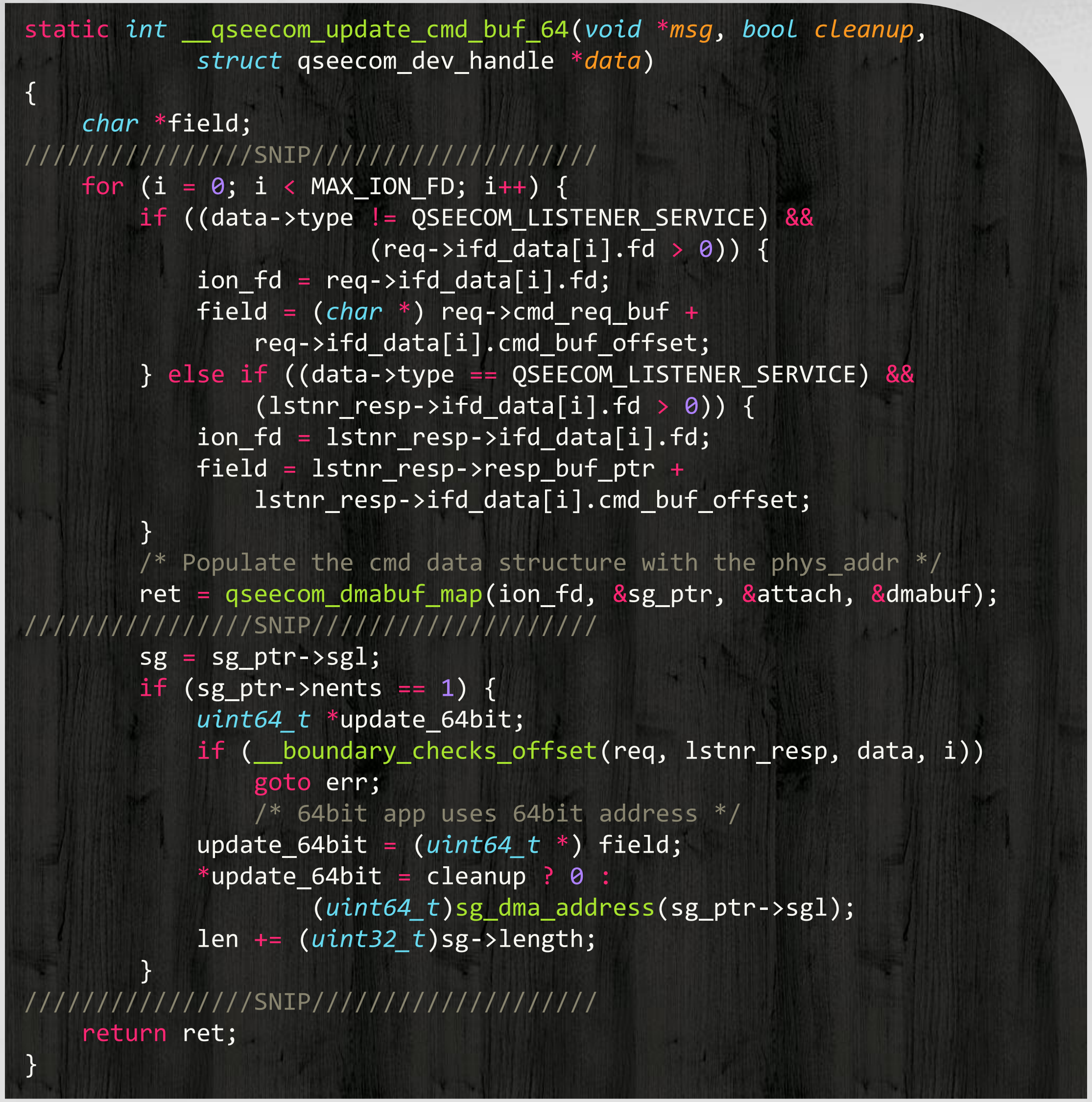
```
struct QSEECom_ion_fd_data {
    int32_t fd;
    uint32_t cmd_buf_offset;
};

struct QSEECom_ion_fd_info {
    struct QSEECom_ion_fd_data data[4];
};
```



```
// QSEEComAPI to send command with ION buffer
int QSEECom_send_modified_cmd(struct QSEECom_handle *handle
, void *send_buf,
            uint32_t sbuf_len, void *resp_buf, uint32_t rbu
f_len,
            struct QSEECom_ion_fd_info  *ifd_data);
```

- A command can share up to 4 ION buffers
- QSEECom_ion_fd_data is a record telling the kernel which field in send_buf is a shared buffer ptr thus need to be translated

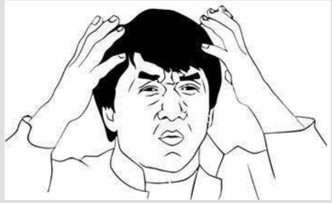
Shared Memory Processing in Kernel



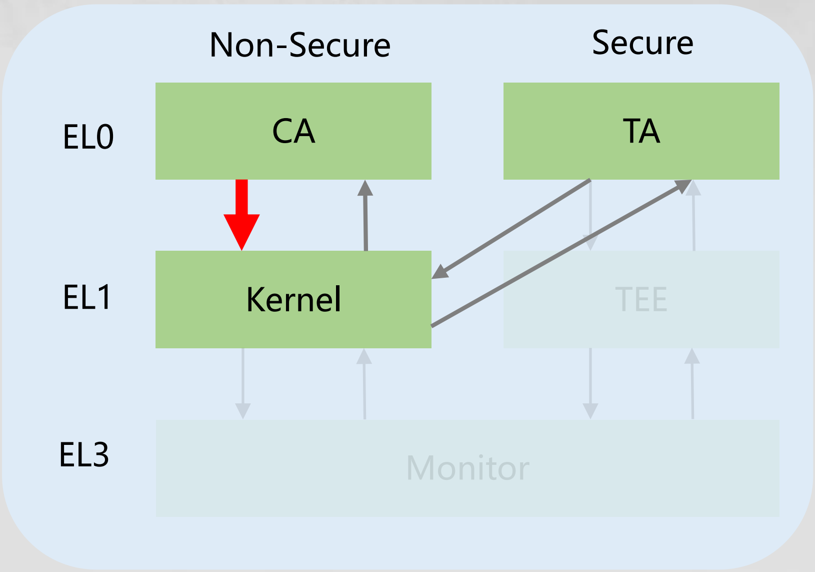
```
static int __qseecom_update_cmd_buf_64(void *msg, bool cleanup,
            struct qseecom_dev_handle *data)
{
    char *field;
////////////////SNIP////////////////////
    for (i = 0; i < MAX_ION_FD; i++) {
        if ((data->type != QSEECOM_LISTENER_SERVICE) &&
                (req->ifd_data[i].fd > 0)) {
            ion_fd = req->ifd_data[i].fd;
            field = (char *) req->cmd_req_buf +
                req->ifd_data[i].cmd_buf_offset;
        } else if ((data->type == QSEECOM_LISTENER_SERVICE) &&
                (lstnr_resp->ifd_data[i].fd > 0)) {
            ion_fd = lstnr_resp->ifd_data[i].fd;
            field = lstnr_resp->resp_buf_ptr +
                lstnr_resp->ifd_data[i].cmd_buf_offset;
        }
        /* Populate the cmd data structure with the phys_addr */
        ret = qseecom_dmabuf_map(ion_fd, &sg_ptr, &attach, &dmabuf);
////////////////SNIP////////////////////
        sg = sg_ptr->sgl;
        if (sg_ptr->nents == 1) {
            uint64_t *update_64bit;
            if (__boundary_checks_offset(req, lstnr_resp, data, i))
                goto err;
                /* 64bit app uses 64bit address */
            update_64bit = (uint64_t *) field;
            *update_64bit = cleanup ? 0 :
                    (uint64_t)sg_dma_address(sg_ptr->sgl);
            len += (uint32_t)sg->length;
        }
////////////////SNIP////////////////////
    return ret;
}
```

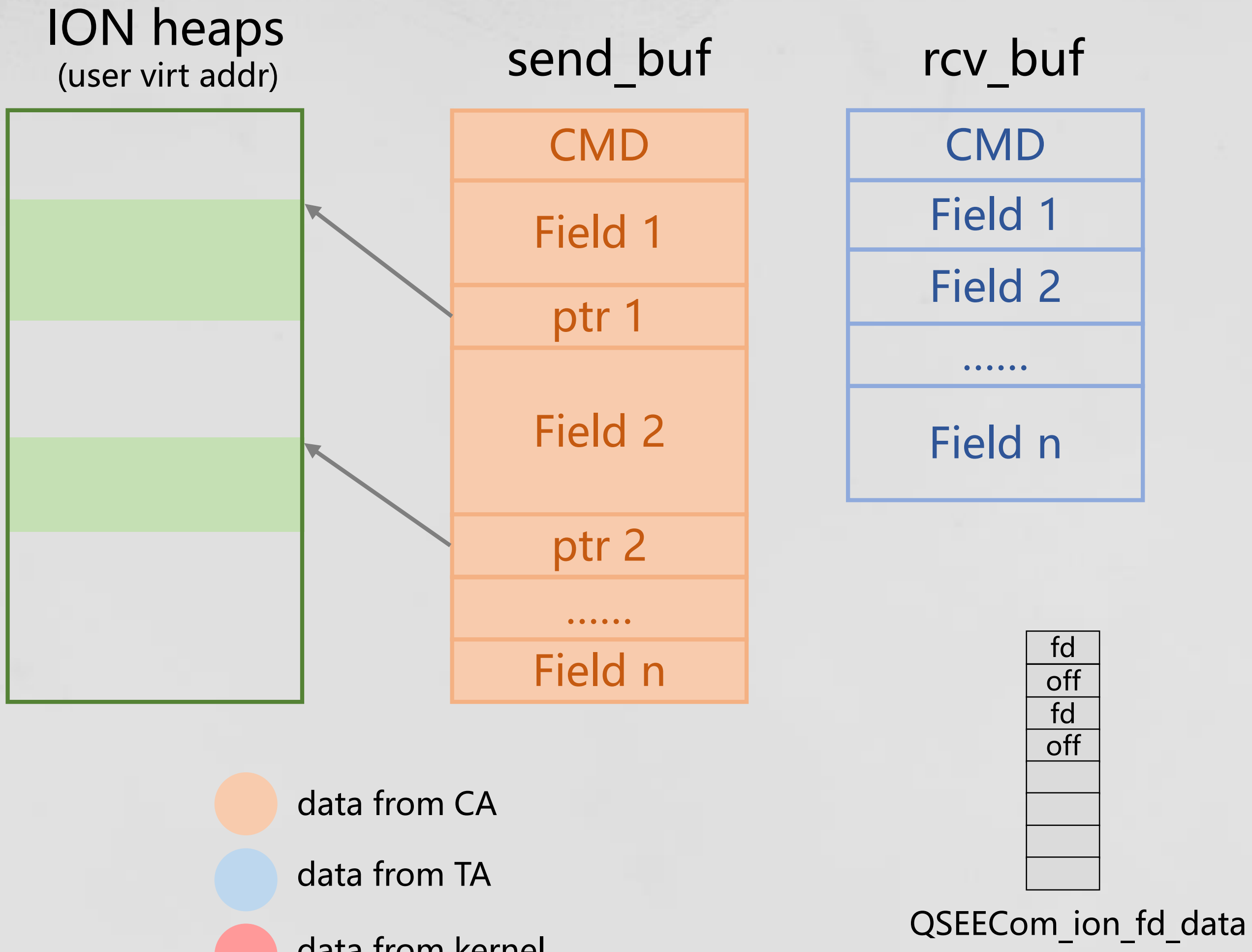
- Then before SMC invocation to TEE, the user virtual address of these buffers will be replaced by physical address according to QSEECom_ion_fd_data

I am confused, show me the pictures!



Send Commands to TA with Shared Memory





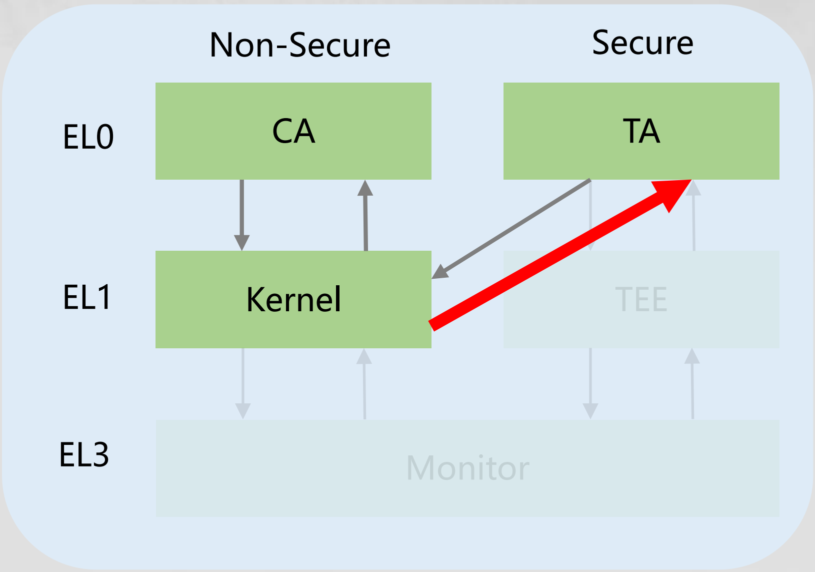
🟠 data from CA

🔵 data from TA

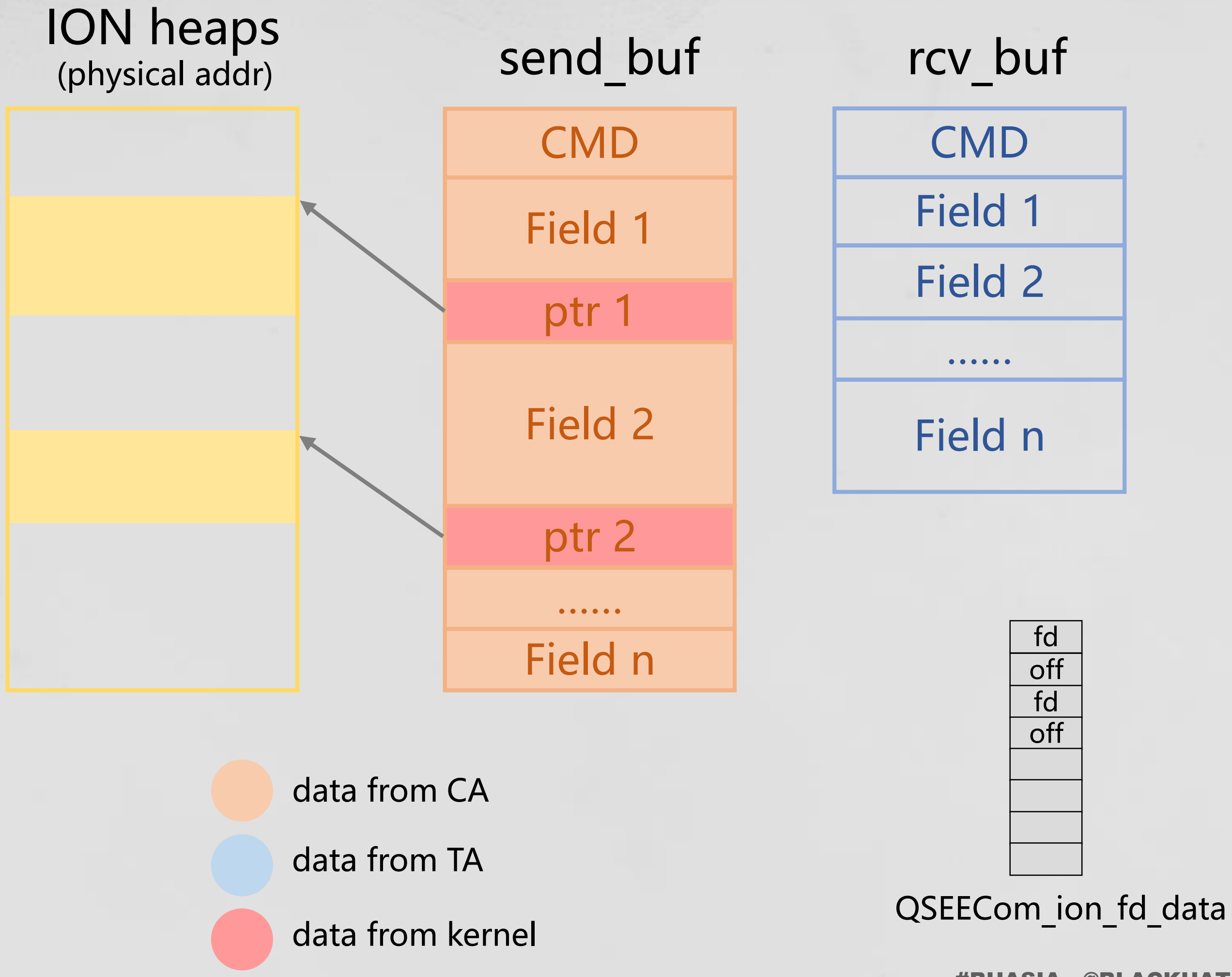
🔴 data from kernel

To share buffer allocated by ION, send_buf will contain ptrs, QSEECom_ion_fd_data parameter will also be sent to **kernel**

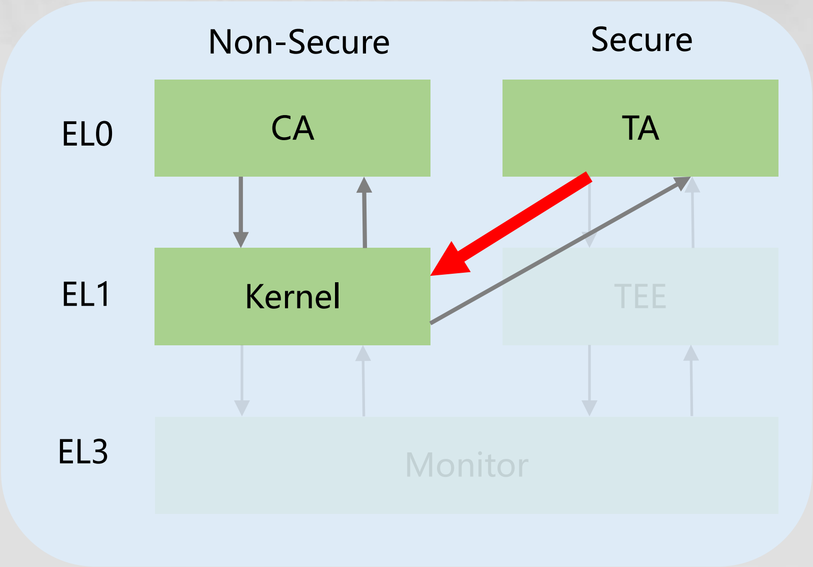
Send Commands to TA with Shared Memory

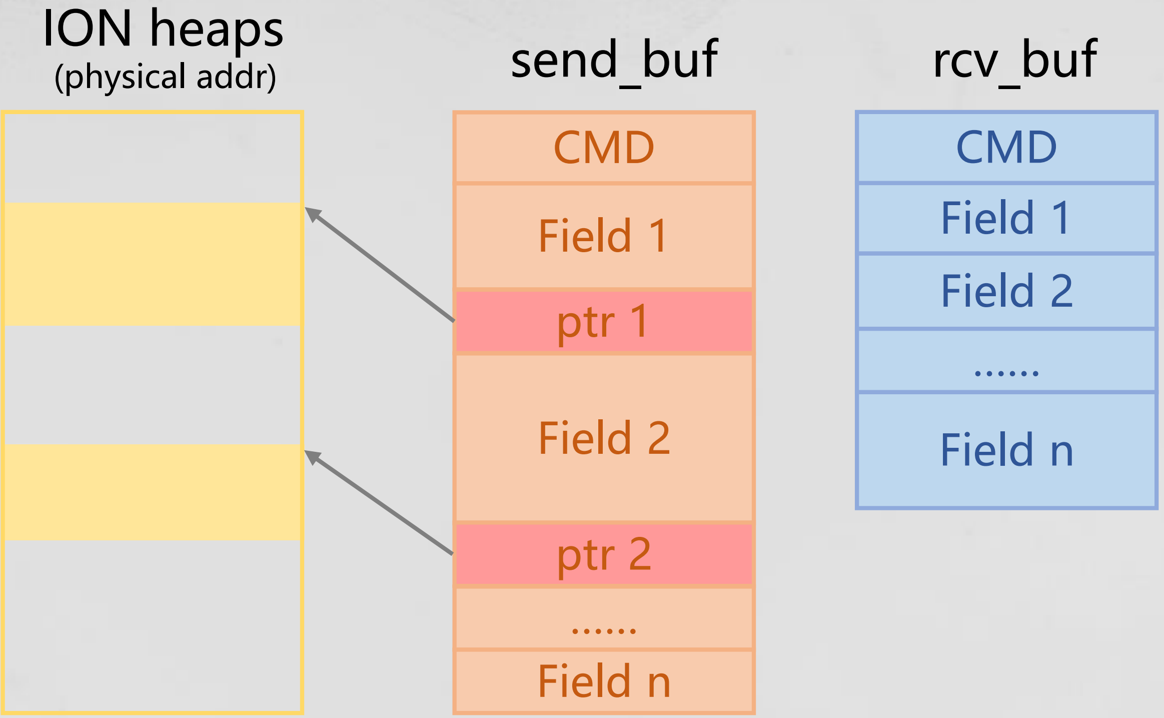


Before SMC call, the kernel will update the ptr of shared buffer with physical address in accordance with QSEECom_ion_fd_data



Send Commands to TA with Shared Memory





(Skip the processing in TEE)

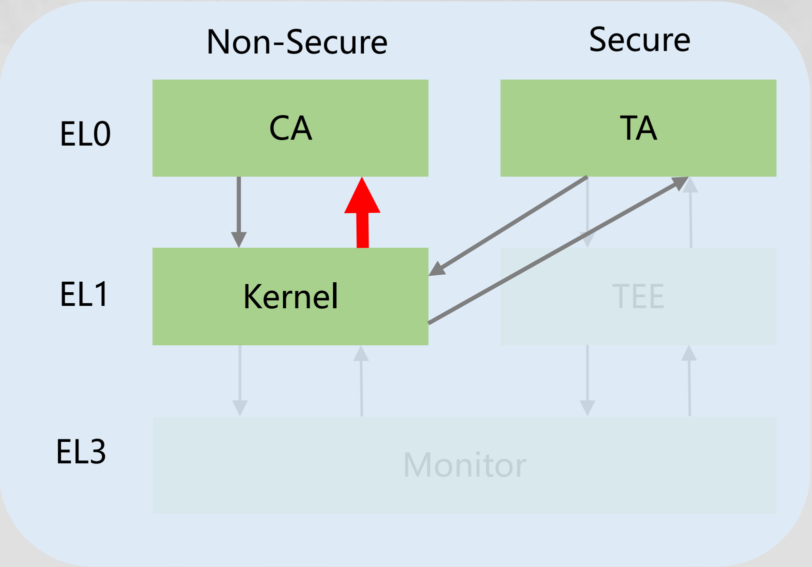
After execution, TA writes returned data into rcv_buf

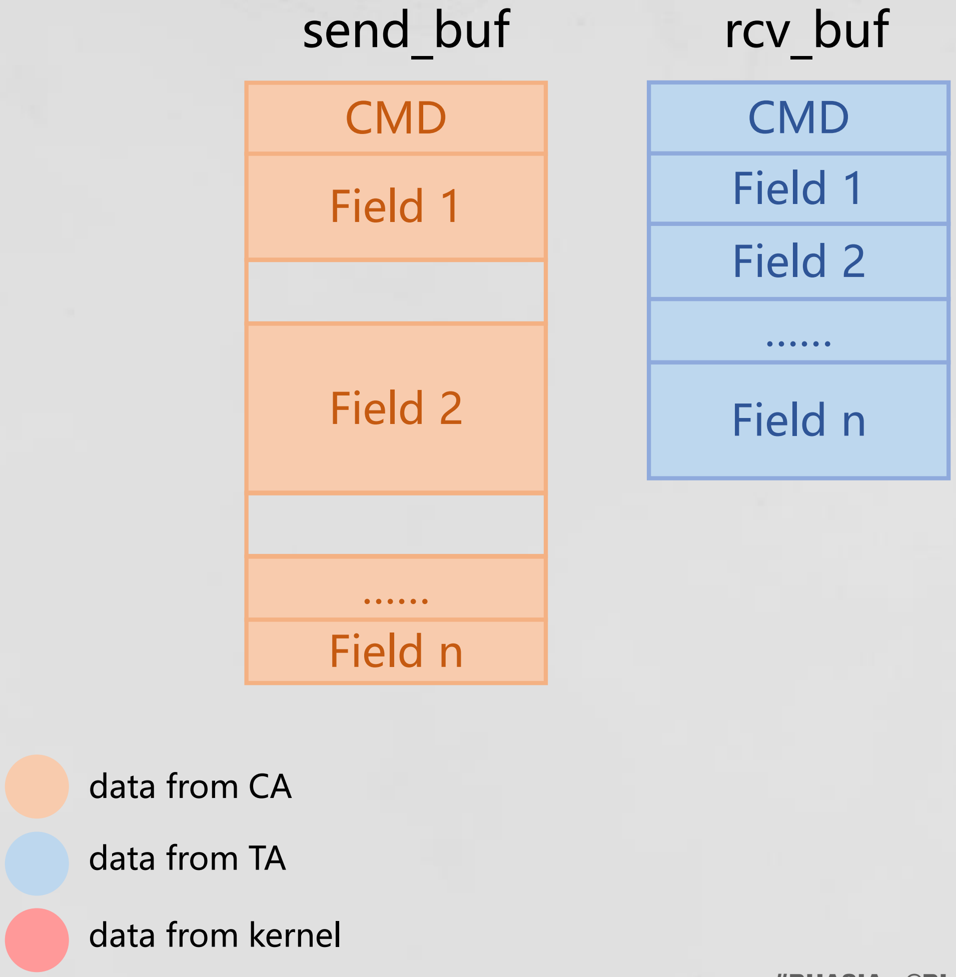
🟠 data from CA

🔵 data from TA

🔴 data from kernel

Send Commands to TA with Shared Memory

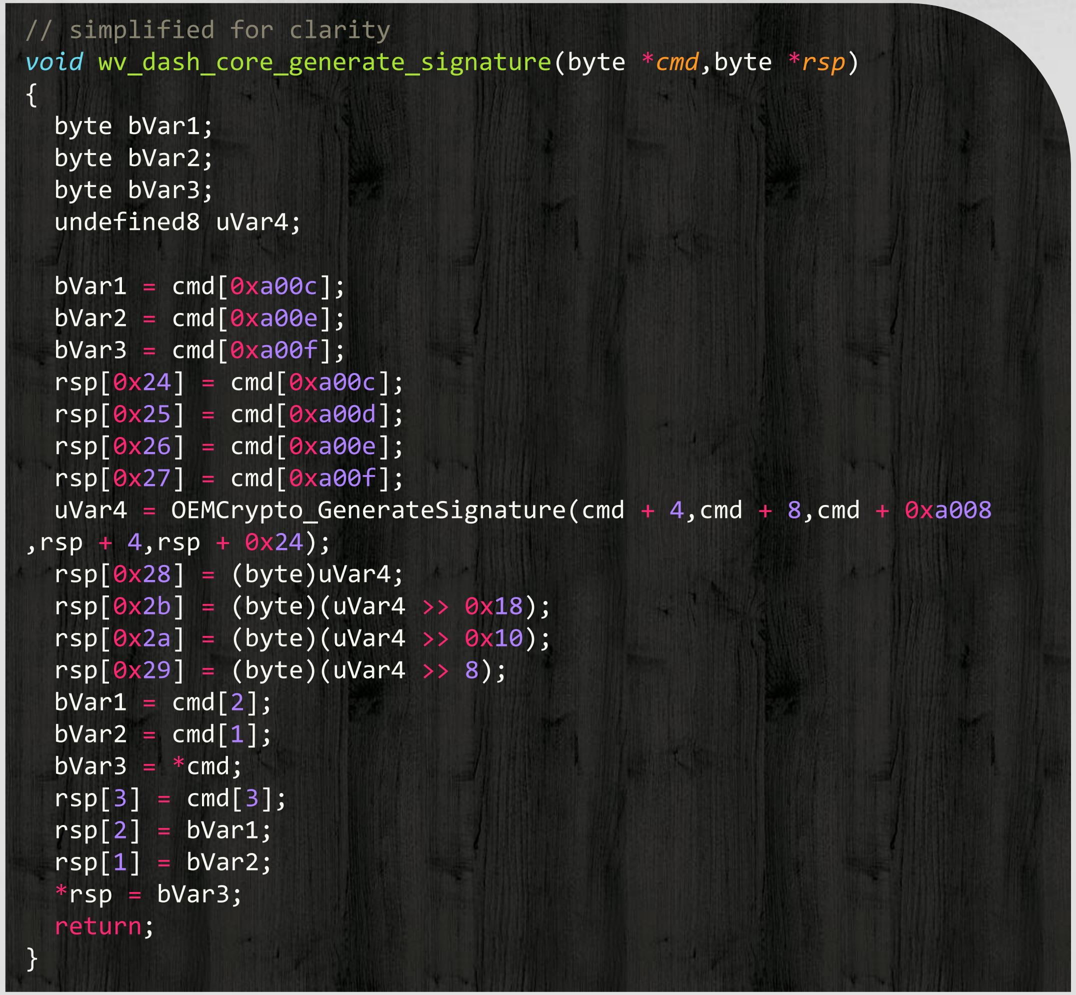




Before returning to userspace, kernel driver wipes out pa to prevent info leak

The Second Vulnerability

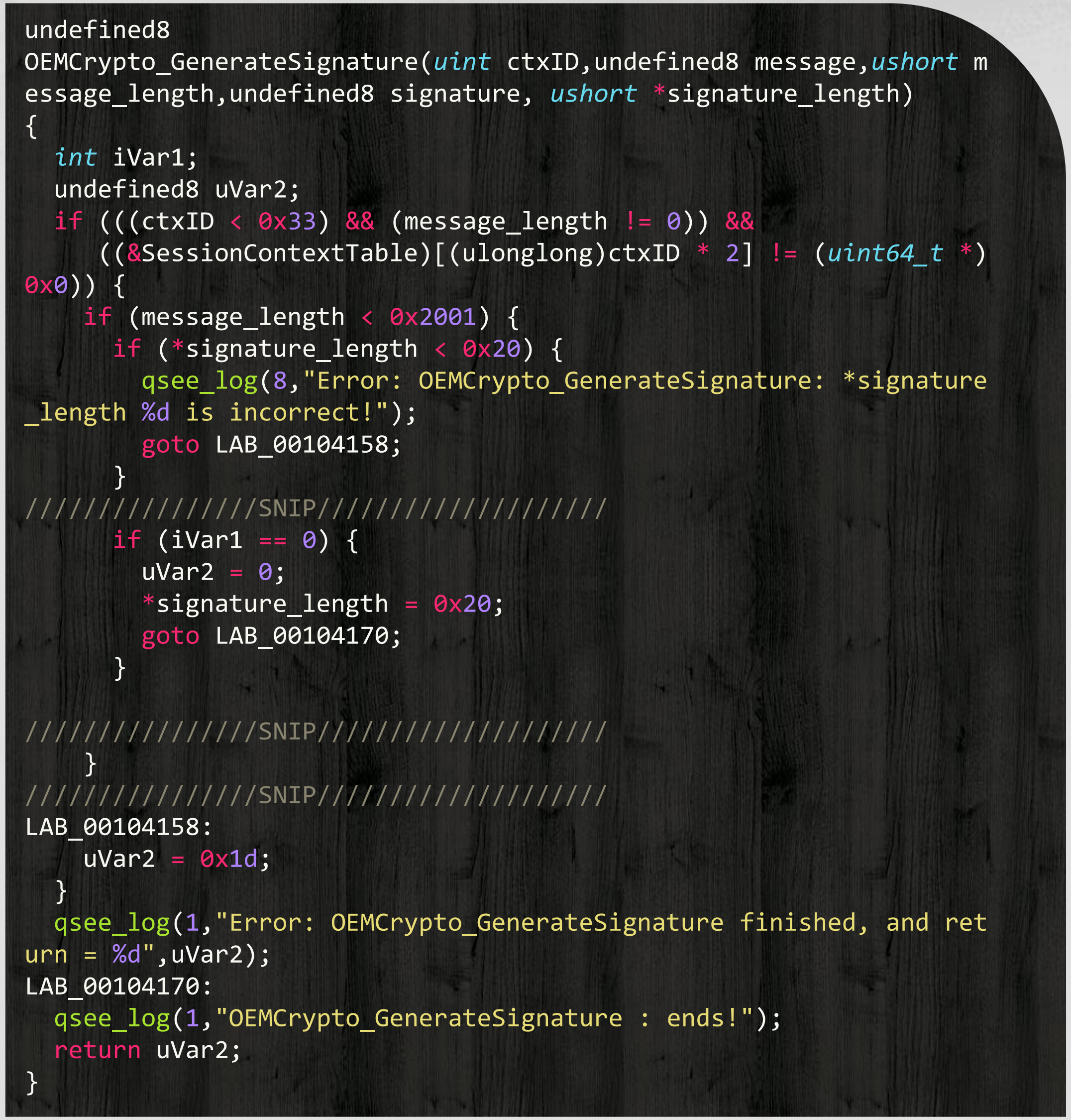
wv_dash_core_generate_signature



```
// simplified for clarity
void wv_dash_core_generate_signature(byte *cmd,byte *rsp)
{
  byte bVar1;
  byte bVar2;
  byte bVar3;
  undefined8 uVar4;

  bVar1 = cmd[0xa00c];
  bVar2 = cmd[0xa00e];
  bVar3 = cmd[0xa00f];
  rsp[0x24] = cmd[0xa00c];
  rsp[0x25] = cmd[0xa00d];
  rsp[0x26] = cmd[0xa00e];
  rsp[0x27] = cmd[0xa00f];
  uVar4 = OEMCrypto_GenerateSignature(cmd + 4,cmd + 8,cmd + 0xa008
,rsp + 4,rsp + 0x24);
  rsp[0x28] = (byte)uVar4;
  rsp[0x2b] = (byte)(uVar4 >> 0x18);
  rsp[0x2a] = (byte)(uVar4 >> 0x10);
  rsp[0x29] = (byte)(uVar4 >> 8);
  bVar1 = cmd[2];
  bVar2 = cmd[1];
  bVar3 = *cmd;
  rsp[3] = cmd[3];
  rsp[2] = bVar1;
  rsp[1] = bVar2;
  *rsp = bVar3;
  return;
}
```

- This is a simple command handler without memory sharing
- rsp[0x24-0x27]'s value is firstly copied from cmd[0xa00c-0xa00f], then modified in OEMCrypto_GenerateSignature()

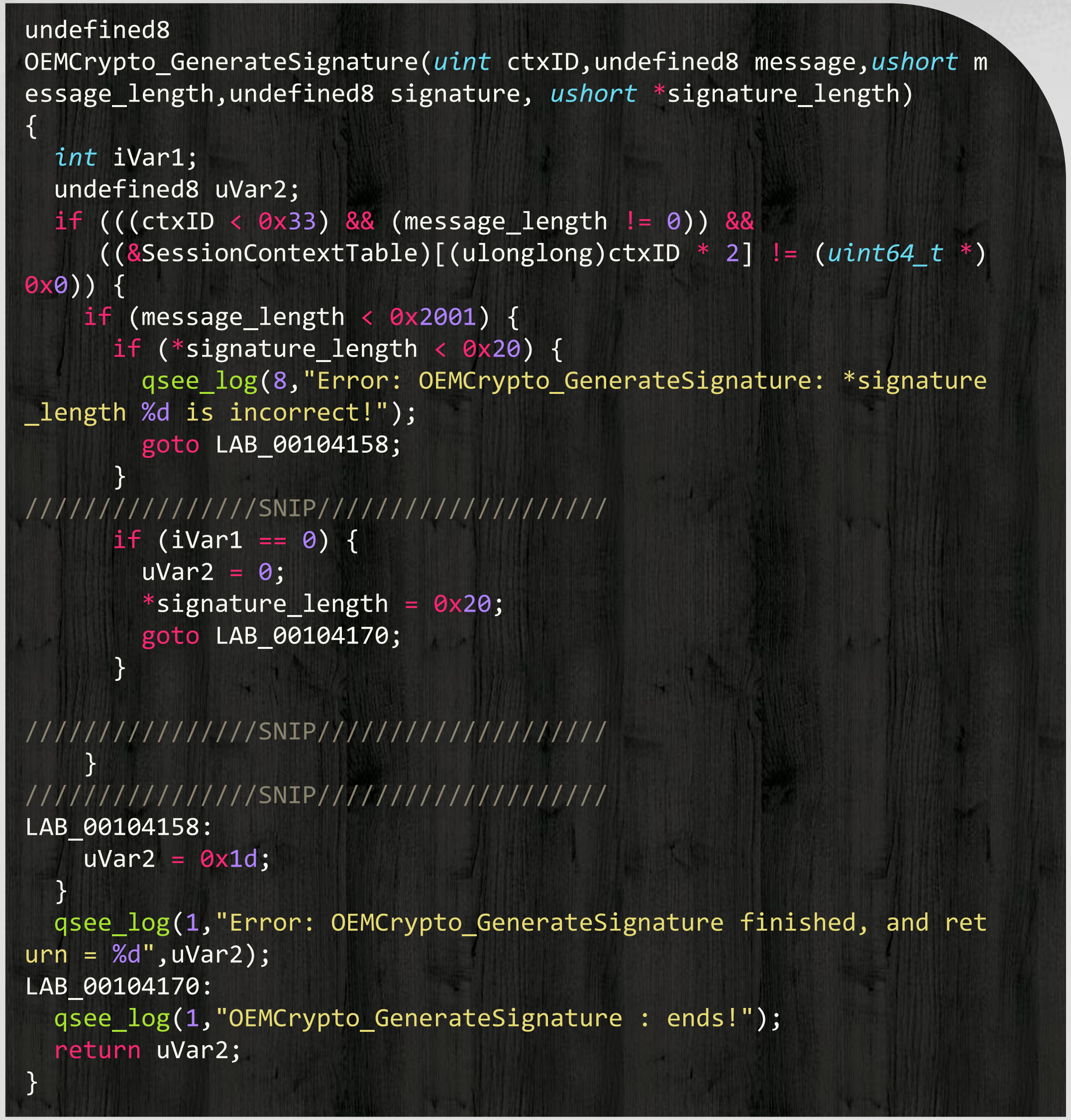
OEMCrypto_GenerateSignature



```
undefined8
OEMCrypto_GenerateSignature(uint ctxID,undefined8 message,ushort m
essage_length,undefined8 signature, ushort *signature_length)
{
  int iVar1;
  undefined8 uVar2;
  if (((ctxID < 0x33) && (message_length != 0)) &&
     ((&SessionContextTable)[(ulonglong)ctxID * 2] != (uint64_t *)
0x0)) {
    if (message_length < 0x2001) {
      if (*signature_length < 0x20) {
        qsee_log(8,"Error: OEMCrypto_GenerateSignature: *signature
_length %d is incorrect!");
        goto LAB_00104158;
      }
////////////////SNIP///////////////////
      if (iVar1 == 0) {
        uVar2 = 0;
        *signature_length = 0x20;
        goto LAB_00104170;
      }

////////////////SNIP///////////////////
    }
////////////////SNIP///////////////////
LAB_00104158:
    uVar2 = 0x1d;
  }
  qsee_log(1,"Error: OEMCrypto_GenerateSignature finished, and ret
urn = %d",uVar2);
LAB_00104170:
  qsee_log(1,"OEMCrypto_GenerateSignature : ends!");
  return uVar2;
}
```

- Here signature_length is equal to rsp[0x24-0x27]
- if *signature_length < 0x20 is met, the function will return with rsp[0x24-0x27] unchanged

OEMCrypto_GenerateSignature

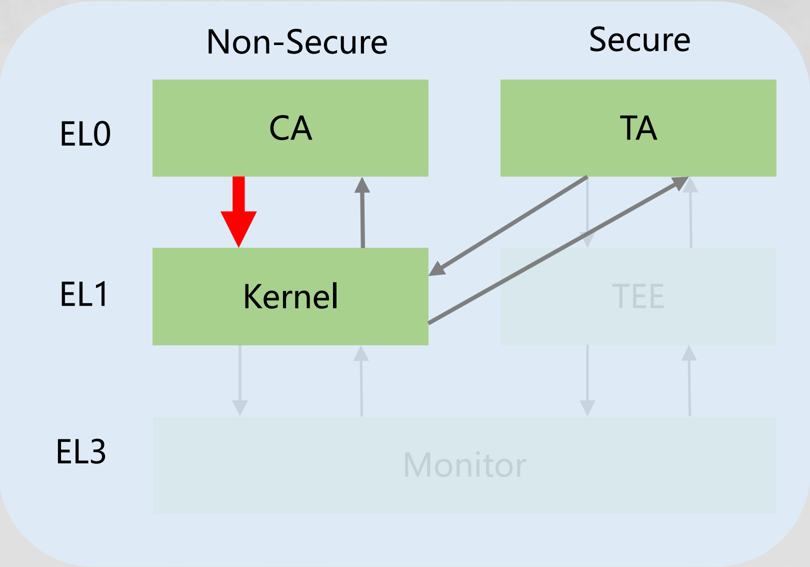


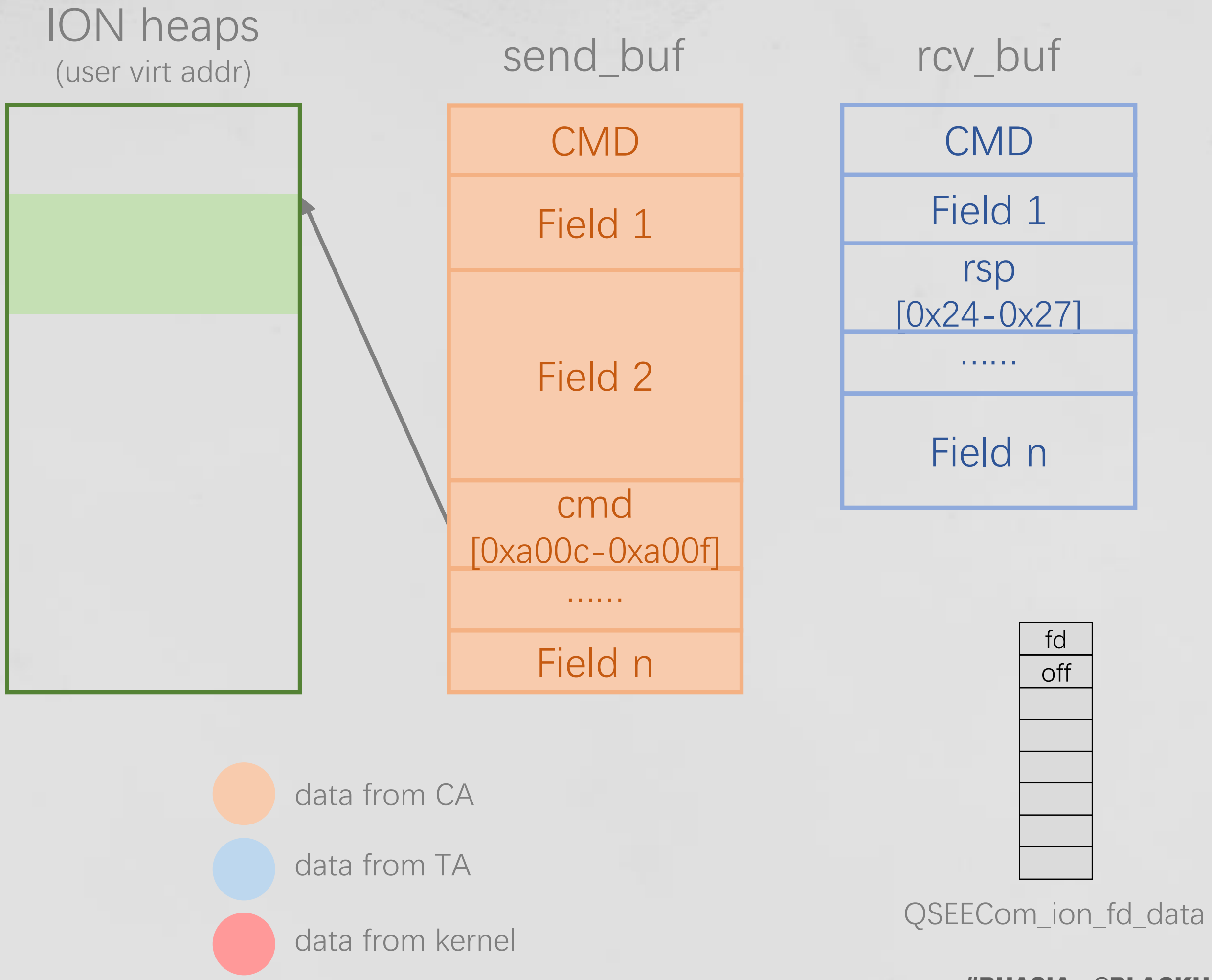
```
undefined8
OEMCrypto_GenerateSignature(uint ctxID,undefined8 message,ushort m
essage_length,undefined8 signature, ushort *signature_length)
{
  int iVar1;
  undefined8 uVar2;
  if (((ctxID < 0x33) && (message_length != 0)) &&
     ((&SessionContextTable)[(ulonglong)ctxID * 2] != (uint64_t *)
0x0)) {
    if (message_length < 0x2001) {
      if (*signature_length < 0x20) {
        qsee_log(8,"Error: OEMCrypto_GenerateSignature: *signature
_length %d is incorrect!");
        goto LAB_00104158;
      }
////////////////SNIP///////////////////
      if (iVar1 == 0) {
        uVar2 = 0;
        *signature_length = 0x20;
        goto LAB_00104170;
      }

////////////////SNIP///////////////////
    }
////////////////SNIP///////////////////
LAB_00104158:
    uVar2 = 0x1d;
  }
  qsee_log(1,"Error: OEMCrypto_GenerateSignature finished, and ret
urn = %d",uVar2);
LAB_00104170:
  qsee_log(1,"OEMCrypto_GenerateSignature : ends!");
  return uVar2;
}
```

- rsp[0x24-0x27] will be returned with the value from cmd[0xa00c-0xa00f]

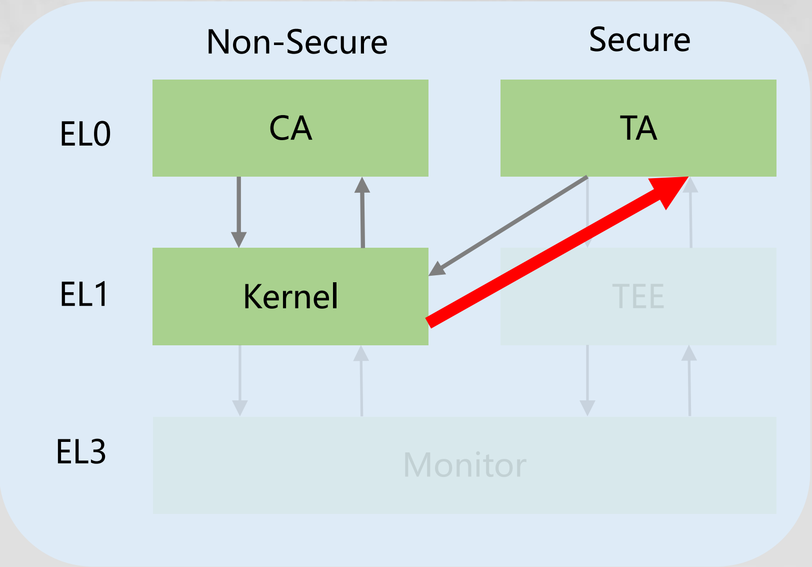
- What if cmd[0xa00c-0xa00f] holds a shared memory ptr?
- Let's see what will happen

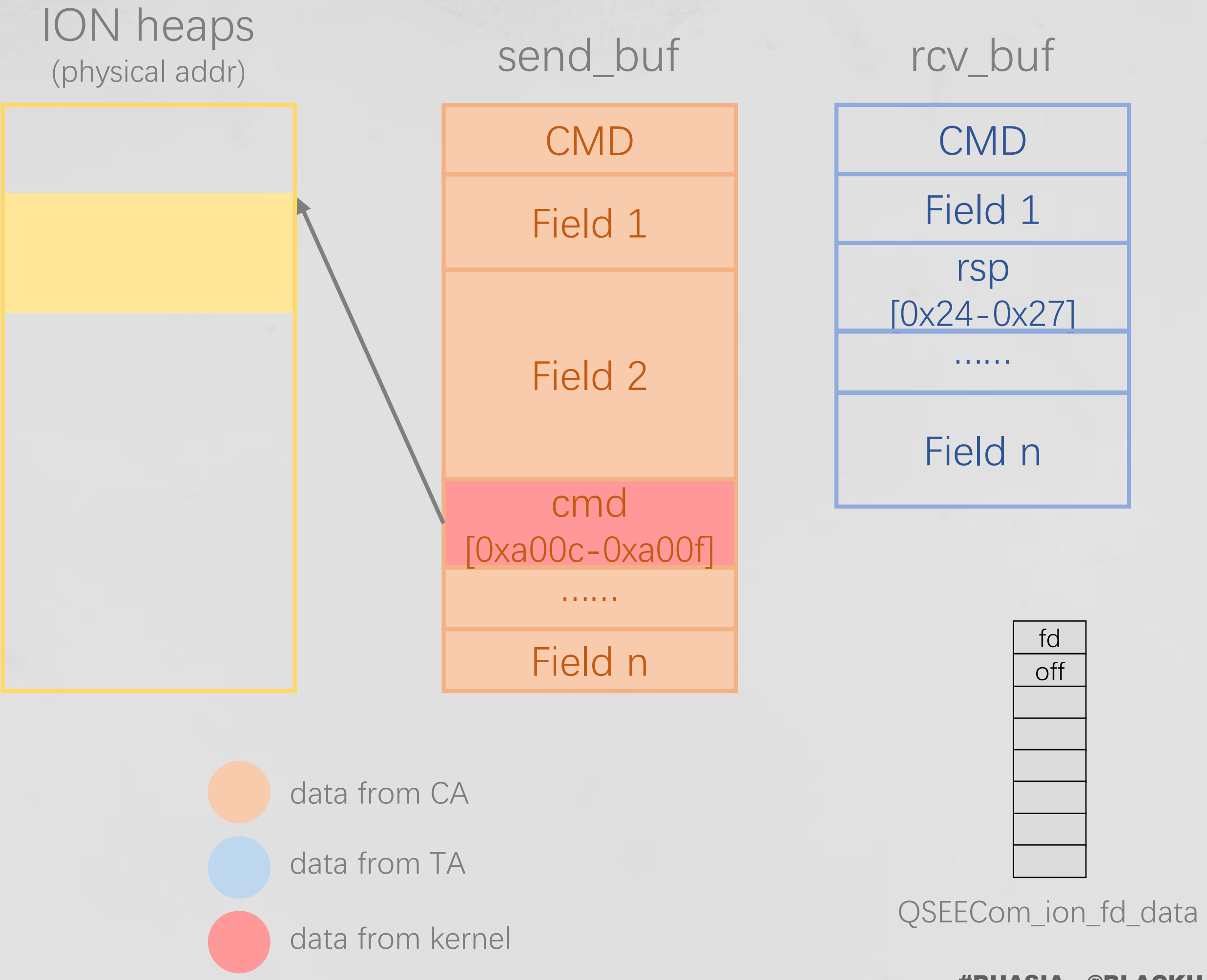
Send Commands to TA with Shared Memory





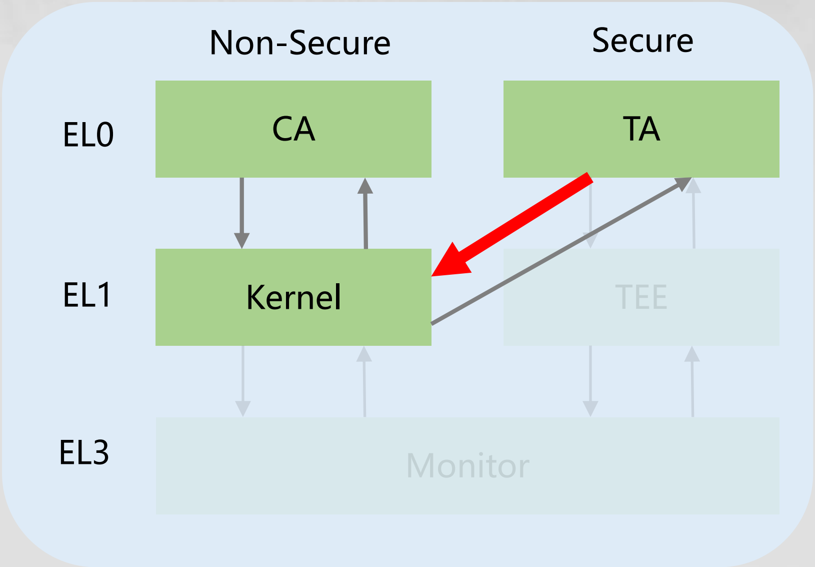
Send Commands to TA with Shared Memory





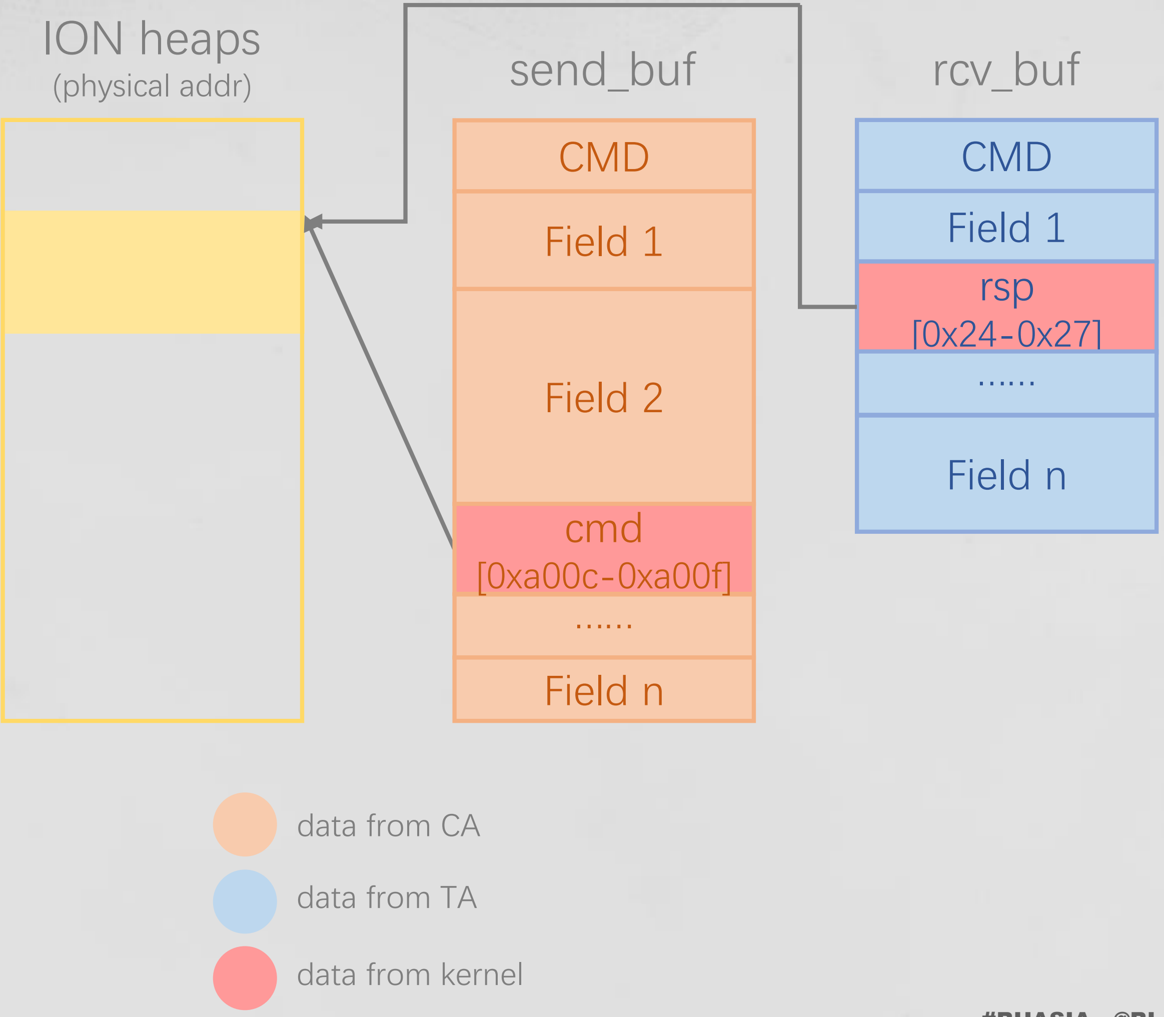
In kernel, cmd[0xa00c-0xa00f] will be updated with the pa of the ION buffer (point to yellow zone)

Send Commands to TA with Shared Memory

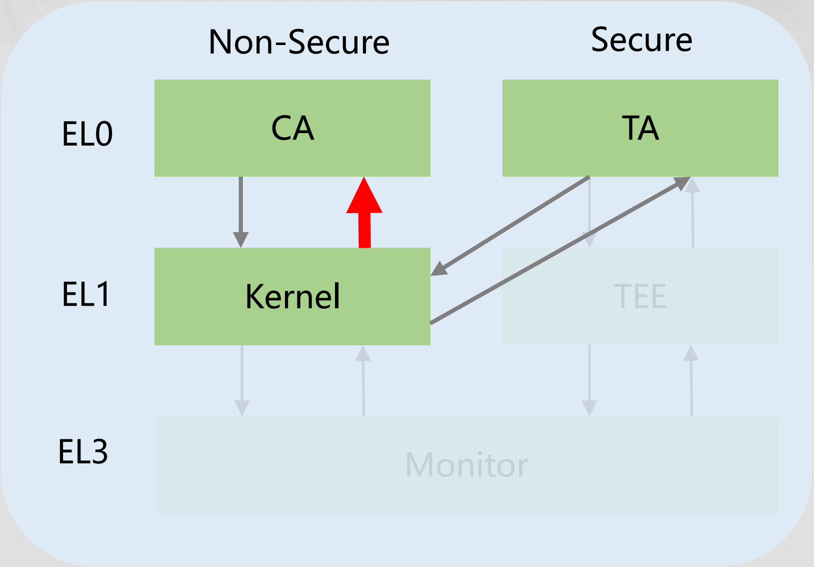


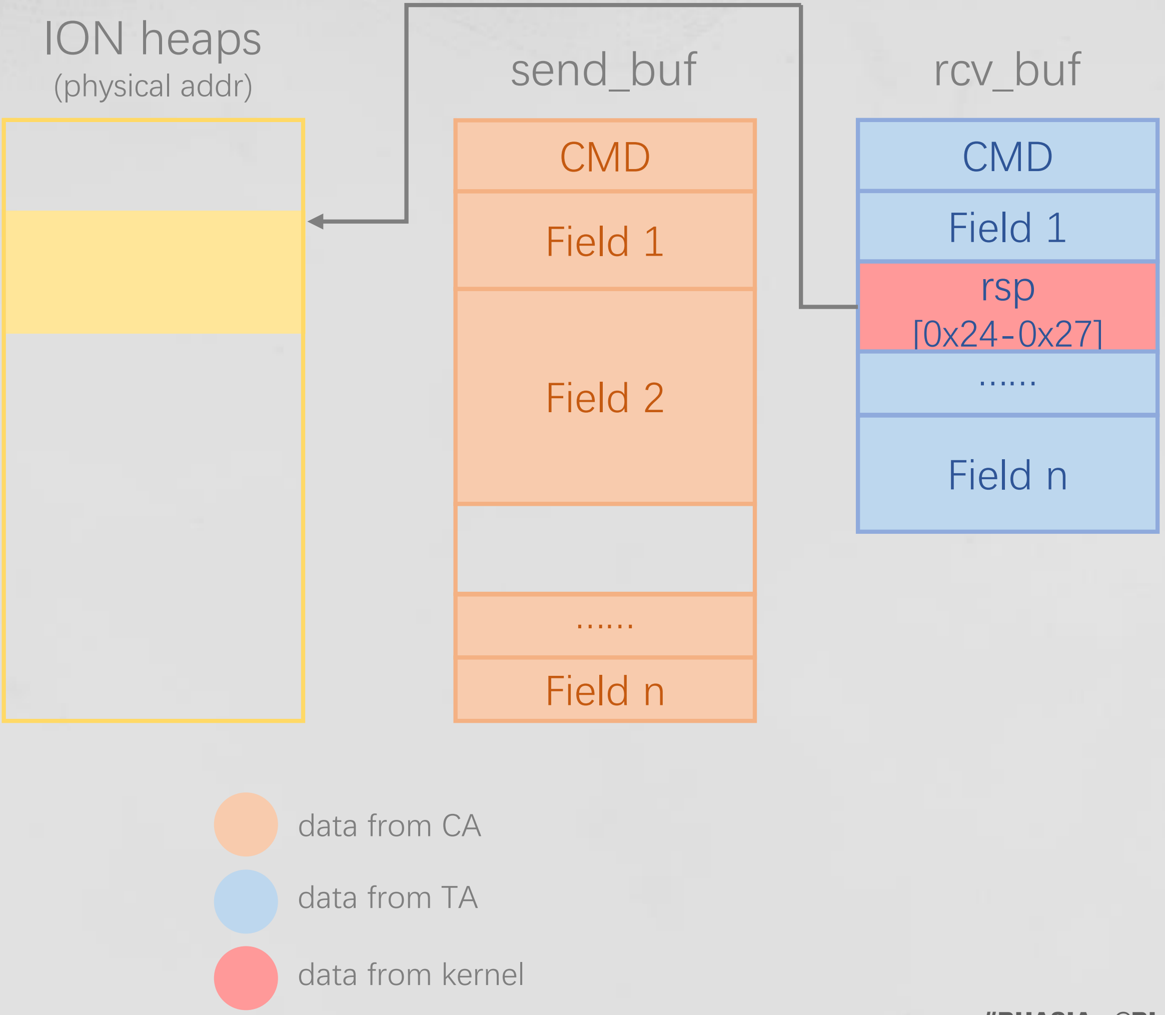
(Skip the processing in TEE)

After execution, TA writes returned data into rcv_buf



Send Commands to TA with Shared Memory





Kernel will wipe out paddr ptr in send_buf, but rsp[0x24-0x27] will hold the pa of the shared ION buffer which is user-controlled

Sum-up

Root cause

- copy data from send_buf to rcv_buf temporarily
- Function returns early when there are errors, leaving the temporary data unchanged

Similar pattern of vulnerabilities were found in other commands:

- wv_dash_core_create_usage_table_header()
- wv_dash_core_generate_rsa_signature()
- wv_dash_core_generate_signature()
- wv_dash_core_shrink_usage_table_header()
- wv_dash_core_update_usg_entry()

In practice,

4 bytes of data can't leak a full 64-bit address, we should do this twice

What's Next

What we have

- ✓ Accurate memcpy() to single byte
- ✓ subsample_offset is a 32-bit value, not enough to cause integer overflow on 64-bit system

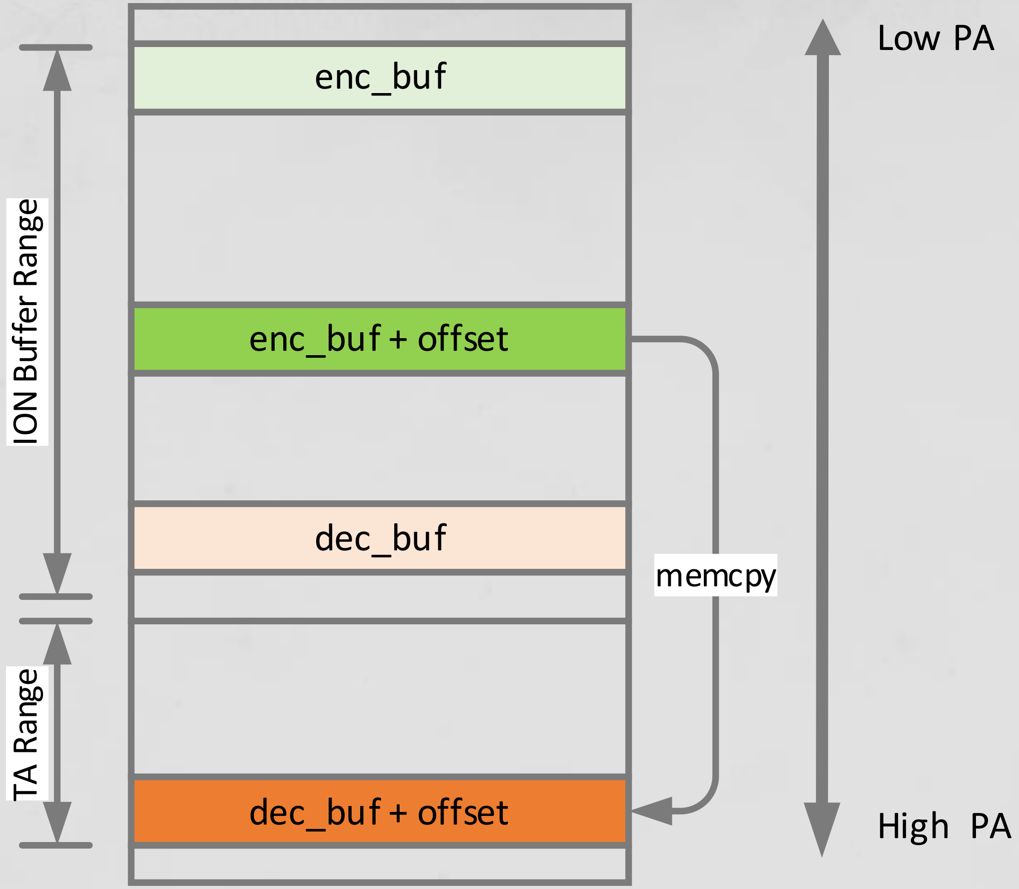
What we need

- ✓ Address of TA in memory
- ✓ Address of user controlled enc_buf and dec_buf in TA's view
- □ Delicate layout that lets the memory corrupt reach TA

Reach TA Memory

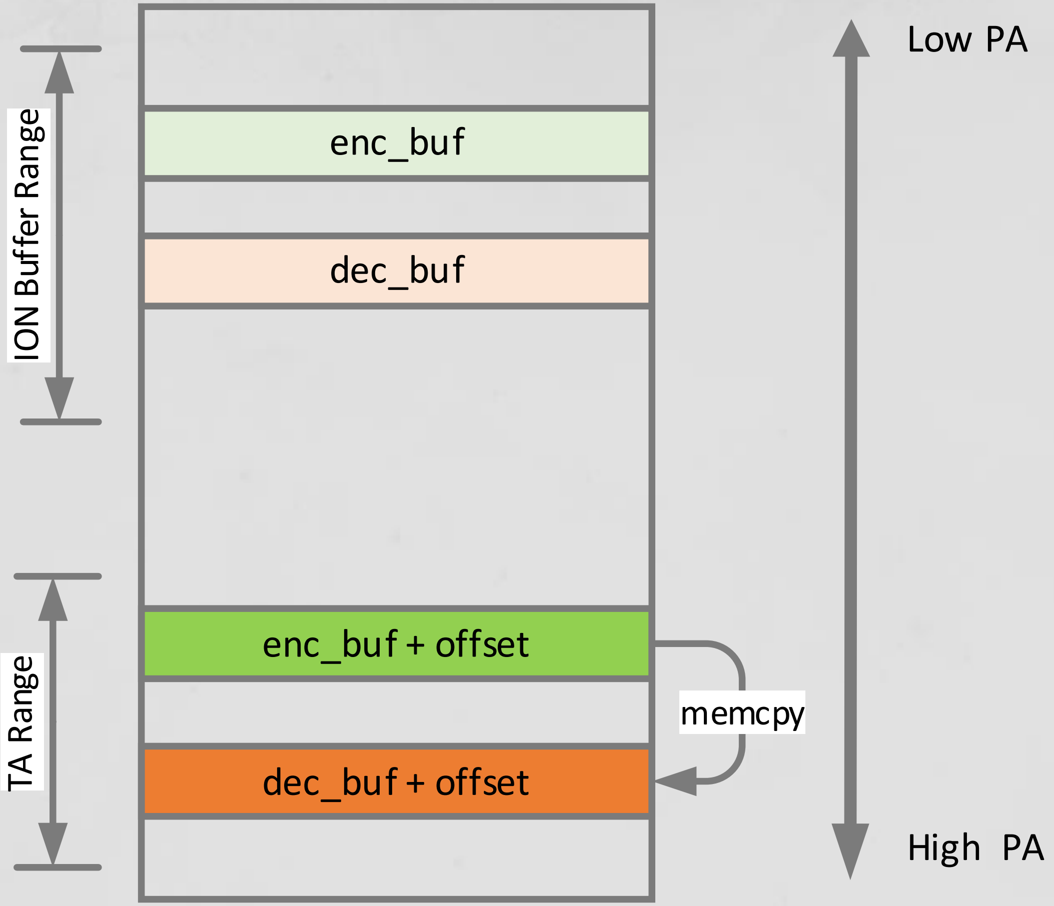
- We need to find a way to reach TA memory from enc_buf/dec_buf
- We have tried many approaches, each with its own limitation
- Here are some of the failed attempts

Plan 1: Huge ION Buffer Range



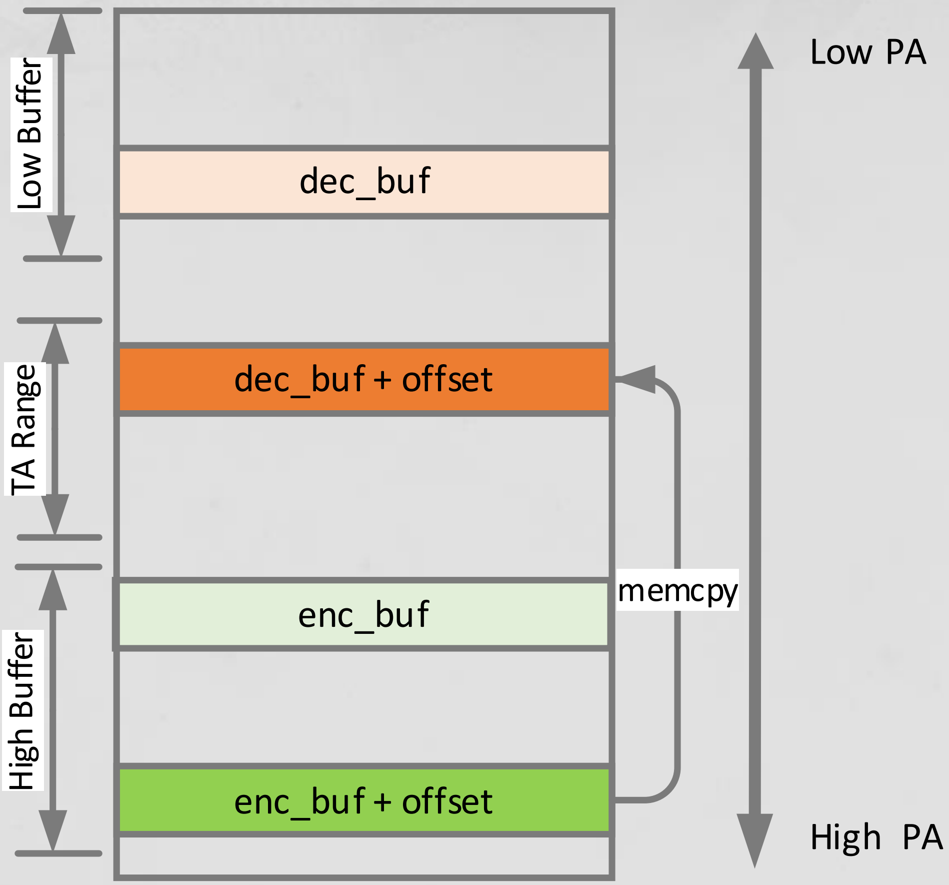
- Can copy arbitrary user-controlled data to TA
- Demand 3 buffers covering large range of memory

Plan 2: TA to TA memcpy



- Only need two buffers
- Copied content is hard to control, may need up to 256 variants to write an arbitrary byte

Plan 3: Sandwich Layout

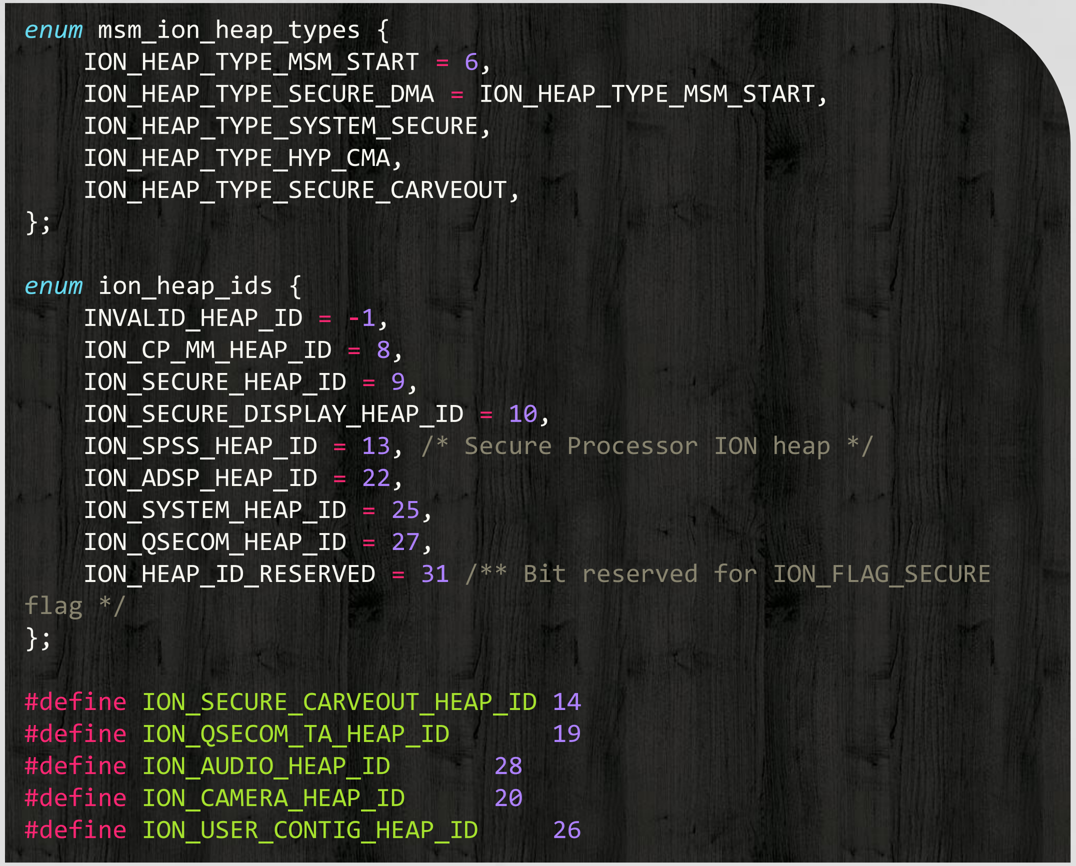


- Need the ability to allocate buffer in both higher and lower regions
- Need 4 buffers

Why They Fail?

- Shared buffers should be mapped to QTEE before using. In CENC command handler, only 2 buffers are mapped
- ION can only allocate buffers in certain regions, each with its own limitations:
  - Preserved DMA region, limited size
  - Not accepted by QTEE while sharing
  - Not physically contiguous or no fixed physical address
  - Unable to hold addresses higher than TA's region

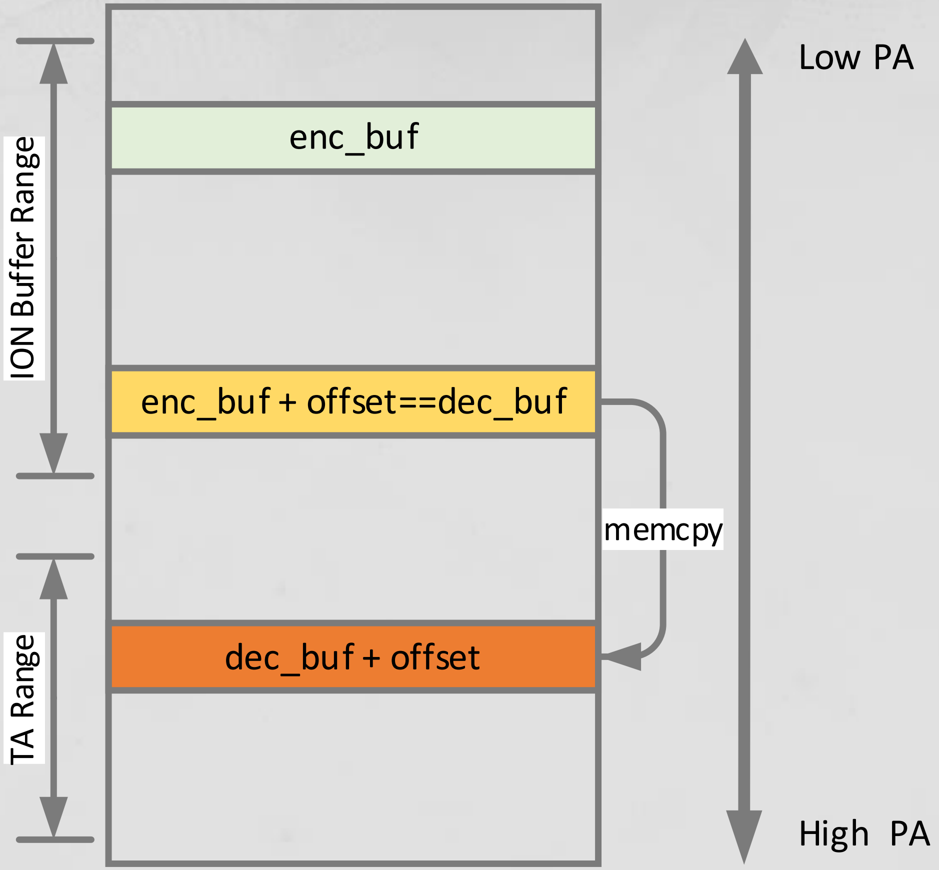
Possible ION Heaps



```
enum msm_ion_heap_types {
    ION_HEAP_TYPE_MSM_START = 6,
    ION_HEAP_TYPE_SECURE_DMA = ION_HEAP_TYPE_MSM_START,
    ION_HEAP_TYPE_SYSTEM_SECURE,
    ION_HEAP_TYPE_HYP_CMA,
    ION_HEAP_TYPE_SECURE_CARVEOUT,
};

enum ion_heap_ids {
    INVALID_HEAP_ID = -1,
    ION_CP_MM_HEAP_ID = 8,
    ION_SECURE_HEAP_ID = 9,
    ION_SECURE_DISPLAY_HEAP_ID = 10,
    ION_SPSS_HEAP_ID = 13, /* Secure Processor ION heap */
    ION_ADSP_HEAP_ID = 22,
    ION_SYSTEM_HEAP_ID = 25,
    ION_QSECOM_HEAP_ID = 27,
    ION_HEAP_ID_RESERVED = 31 /** Bit reserved for ION_FLAG_SECURE
flag */
};

#define ION_SECURE_CARVEOUT_HEAP_ID 14
#define ION_QSECOM_TA_HEAP_ID       19
#define ION_AUDIO_HEAP_ID       28
#define ION_CAMERA_HEAP_ID      20
#define ION_USER_CONTIG_HEAP_ID     26
```

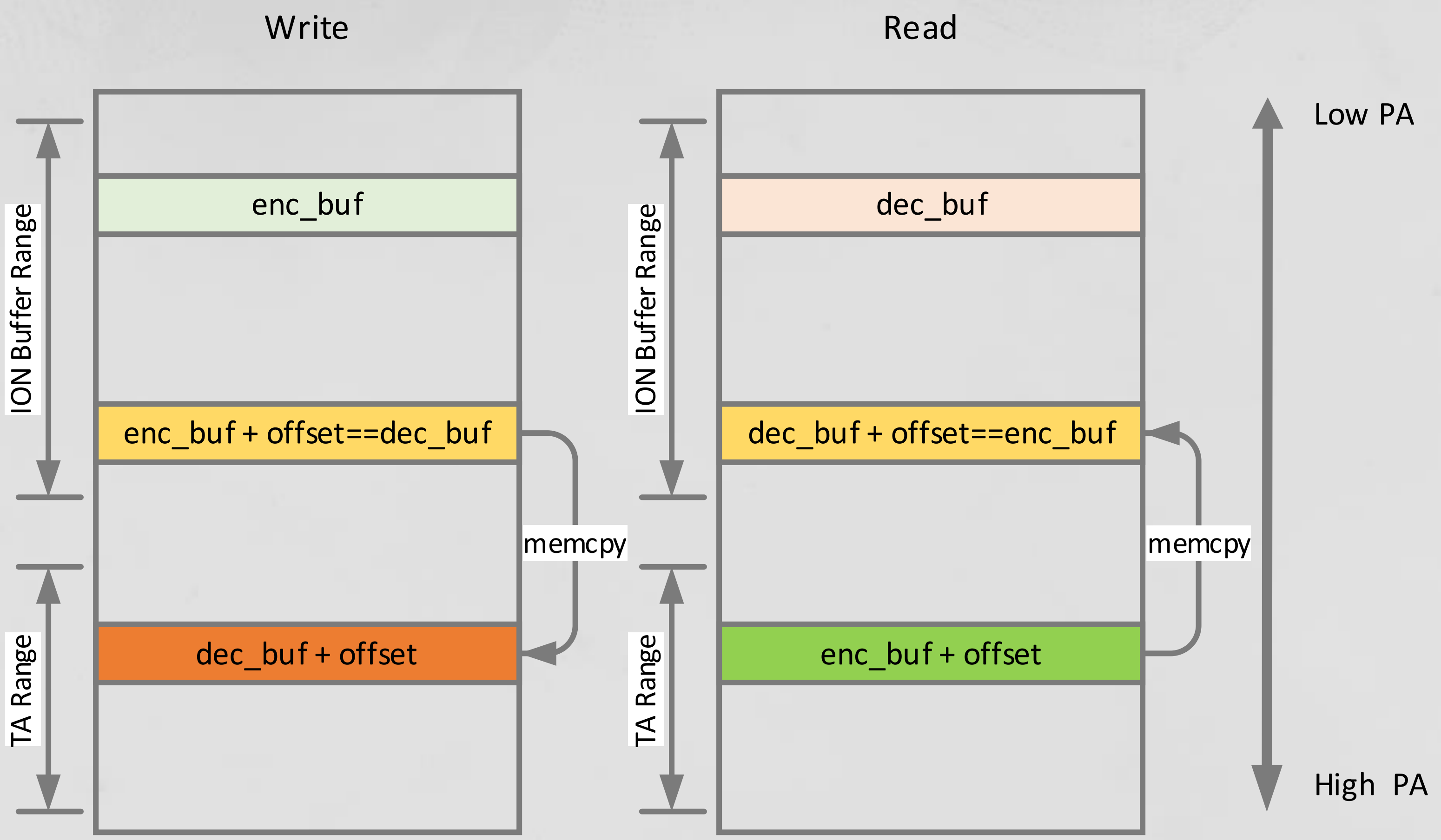
- In practice, only 19, 22, 25, 26, 27 are accepted by QTEE

Plan N: Overlapping Layout



- enc_buf + offset==dec_buf
- Only need 2 buffers
- Smaller memory range that can fit in the scarce memory space

R/W Primitives



We've Got Everything!

What we have

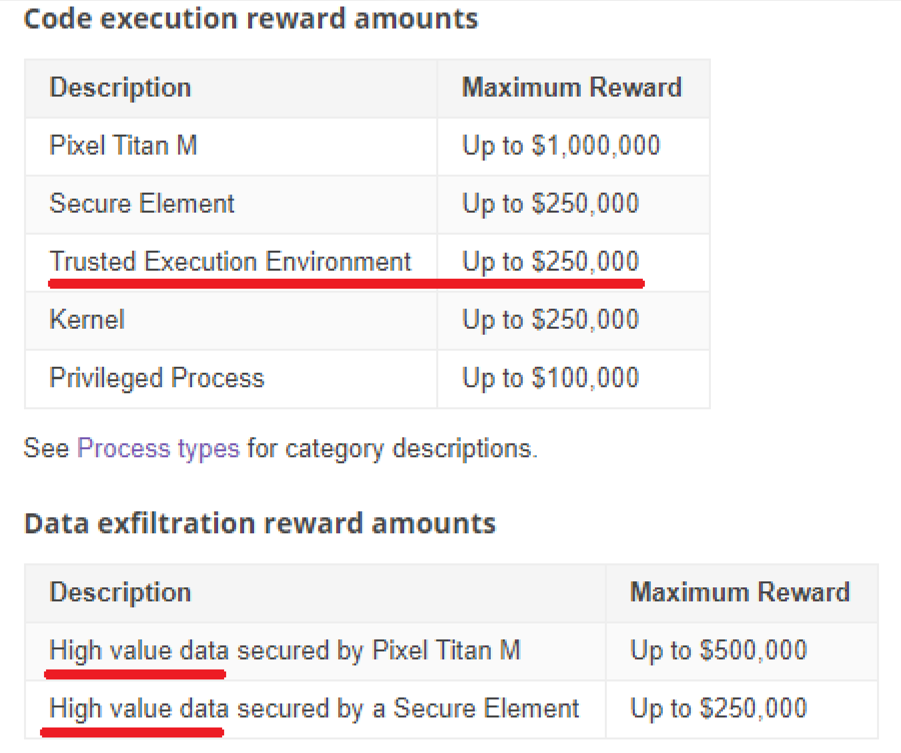
- ✓ Accurate memcpy() to single byte
- ✓ subsample_offset is a 32-bit value, not enough to cause integer overflow on 64-bit system

What we need

- ✓ Address of TA in memory
- ✓ Address of user controlled enc_buf and dec_buf in TA's view
- ✓ Delicate layout that lets the memory corrupt reach TA

Steal the Key

- Time to pop a shell!
- But TEEs have no shell/calculator to pop!



Code execution reward amounts

Description	Maximum Reward
Pixel Titan M	Up to $1,000,000
Secure Element	Up to $250,000
Trusted Execution Environment	Up to $250,000
Kernel	Up to $250,000
Privileged Process	Up to $100,000

See Process types for category descriptions.

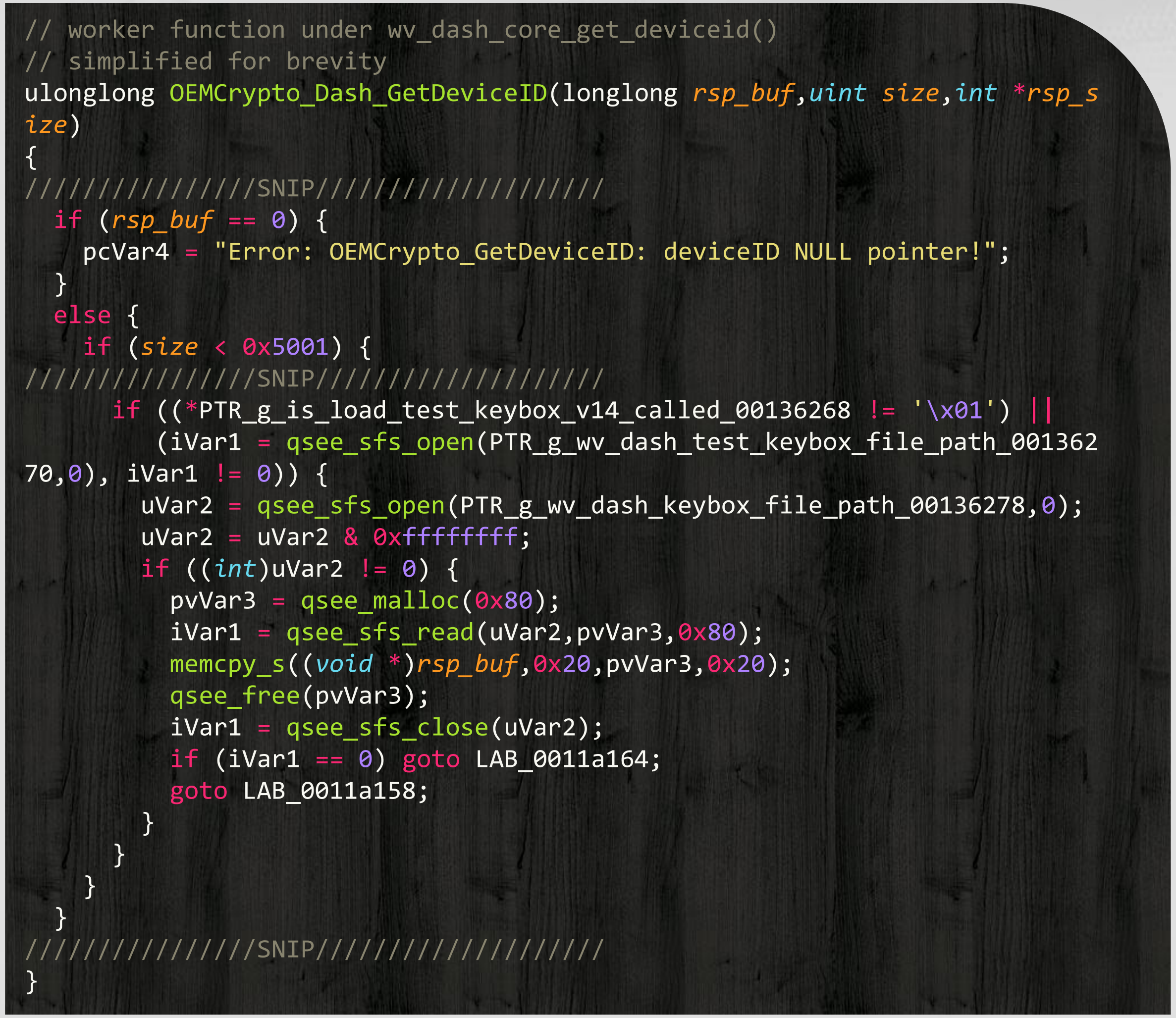
Data exfiltration reward amounts

Description	Maximum Reward
High value data secured by Pixel Titan M	Up to $500,000
High value data secured by a Secure Element	Up to $250,000

https://www.google.com/about/appsecurity/android-rewards/

- It seems code execution and high value data exfiltration are valued
- Let's combine them to exfriltrate the DRM keybox used by Widevine

The Victim

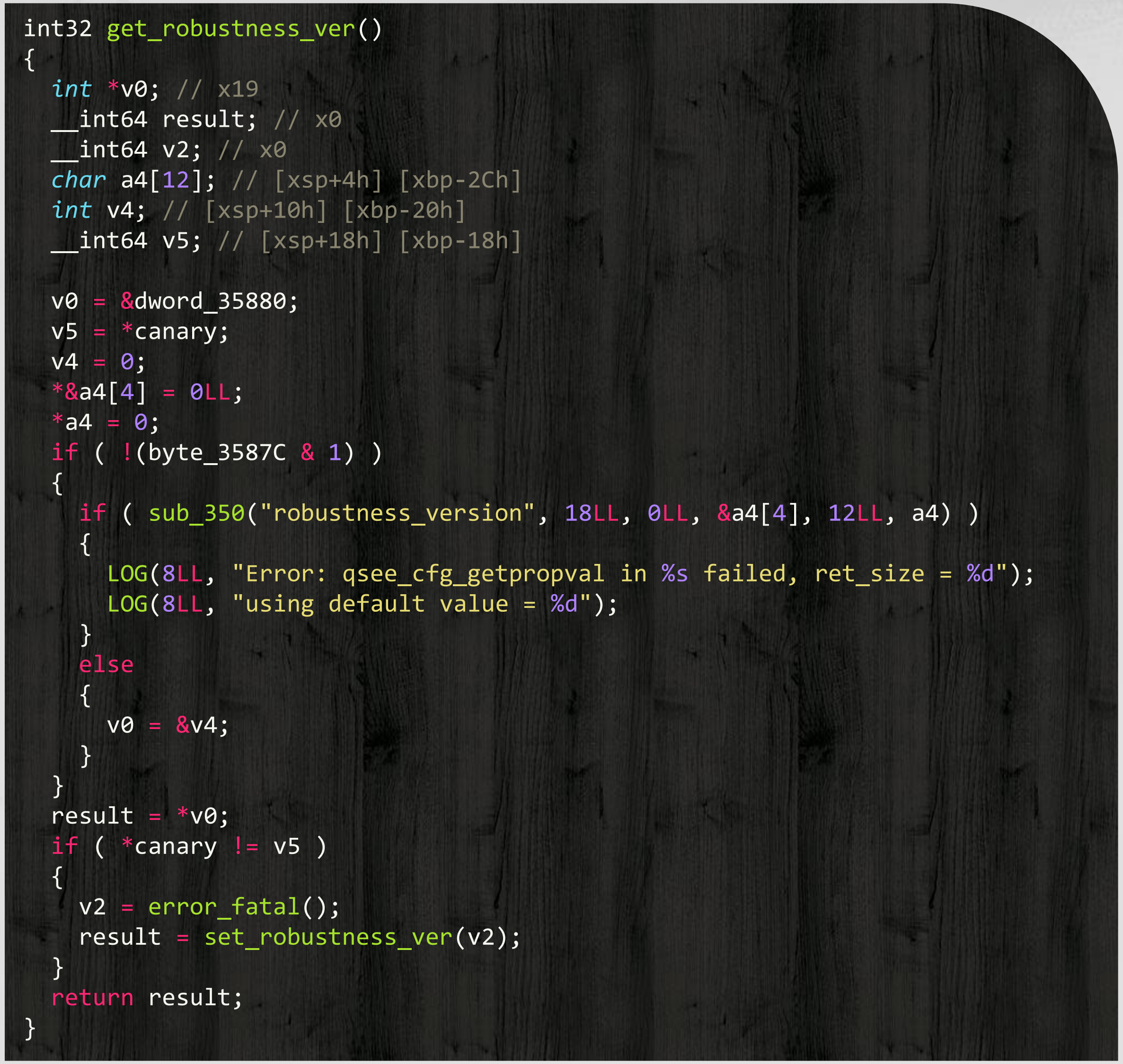


```
// worker function under wv_dash_core_get_deviceid()
// simplified for brevity
ulonglong OEMCrypto_Dash_GetDeviceID(longlong rsp_buf,uint size,int *rsp_s
ize)
{
////////////////SNIP////////////////////
  if (rsp_buf == 0) {
    pcVar4 = "Error: OEMCrypto_GetDeviceID: deviceID NULL pointer!";
  }
  else {
    if (size < 0x5001) {
////////////////SNIP////////////////////
      if ((*PTR_g_is_load_test_keybox_v14_called_00136268 != '\x01') ||
         (iVar1 = qsee_sfs_open(PTR_g_wv_dash_test_keybox_file_path_001362
70,0), iVar1 != 0)) {
        uVar2 = qsee_sfs_open(PTR_g_wv_dash_keybox_file_path_00136278,0);
        uVar2 = uVar2 & 0xffffffff;
        if ((int)uVar2 != 0) {
          pvVar3 = qsee_malloc(0x80);
          iVar1 = qsee_sfs_read(uVar2,pvVar3,0x80);
          memcpy_s((void *)rsp_buf,0x20,pvVar3,0x20);
          qsee_free(pvVar3);
          iVar1 = qsee_sfs_close(uVar2);
          if (iVar1 == 0) goto LAB_0011a164;
          goto LAB_0011a158;
        }
      }
    }
  }
////////////////SNIP////////////////////
}
```

- Contains open, read, return operations to SFS
- Modify g_wv_dash_keybox_file_path to exfriltrate other files

*SFS(Secure File System) is Qualcomm's trusted storage system protected by QTEE

Hijack qsee_malloc()



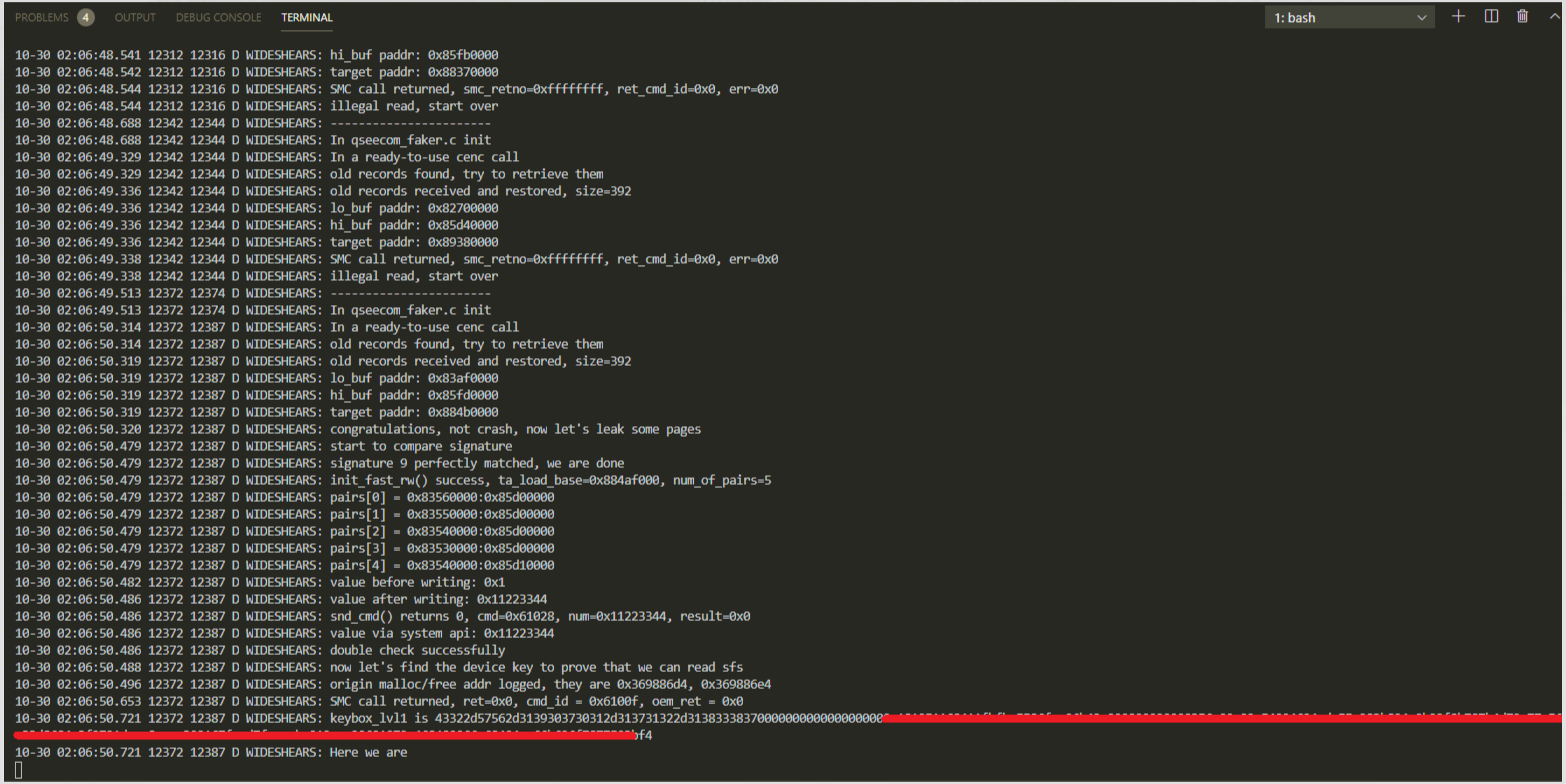
```
int32 get_robustness_ver()
{
  int *v0; // x19
  __int64 result; // x0
  __int64 v2; // x0
  char a4[12]; // [xsp+4h] [xbp-2Ch]
  int v4; // [xsp+10h] [xbp-20h]
  __int64 v5; // [xsp+18h] [xbp-18h]

  v0 = &dword_35880;
  v5 = *canary;
  v4 = 0;
  *&a4[4] = 0LL;
  *a4 = 0;
  if ( !(byte_3587C & 1) )
  {
    if ( sub_350("robustness_version", 18LL, 0LL, &a4[4], 12LL, a4) )
    {
      LOG(8LL, "Error: qsee_cfg_getpropval in %s failed, ret_size = %d");
      LOG(8LL, "using default value = %d");
    }
    else
    {
      v0 = &v4;
    }
  }
  result = *v0;
  if ( *canary != v5 )
  {
    v2 = error_fatal();
    result = set_robustness_ver(v2);
  }
  return result;
}
```

- GOT hijacking
- Replace qsee_malloc() with get_robustness_ver()
- Relocate qsee_malloc()'s buffer to controlled global buffer

Leak the Keybox

- Also hijack qsee_free() to avoid crashes
- After invoking OEMCrypto_Dash_GetDeviceID(), the keybox will be left on the global region
- Use the read primitive to retrieve the keybox contents

Demo



```
10-30 02:06:48.541 12312 12316 D WIDESHEARS: hi_buf paddr: 0x85fb0000
10-30 02:06:48.542 12312 12316 D WIDESHEARS: target paddr: 0x88370000
10-30 02:06:48.544 12312 12316 D WIDESHEARS: SMC call returned, smc_retno=0xffffffff, ret_cmd_id=0x0, err=0x0
10-30 02:06:48.544 12312 12316 D WIDESHEARS: illegal read, start over
10-30 02:06:48.688 12342 12344 D WIDESHEARS: ----------------------
10-30 02:06:48.688 12342 12344 D WIDESHEARS: In qseecom_faker.c init
10-30 02:06:49.329 12342 12344 D WIDESHEARS: In a ready-to-use cenc call
10-30 02:06:49.329 12342 12344 D WIDESHEARS: old records found, try to retrieve them
10-30 02:06:49.336 12342 12344 D WIDESHEARS: old records received and restored, size=392
10-30 02:06:49.336 12342 12344 D WIDESHEARS: lo_buf paddr: 0x82700000
10-30 02:06:49.336 12342 12344 D WIDESHEARS: hi_buf paddr: 0x85d40000
10-30 02:06:49.336 12342 12344 D WIDESHEARS: target paddr: 0x89380000
10-30 02:06:49.338 12342 12344 D WIDESHEARS: SMC call returned, smc_retno=0xffffffff, ret_cmd_id=0x0, err=0x0
10-30 02:06:49.338 12342 12344 D WIDESHEARS: illegal read, start over
10-30 02:06:49.513 12372 12374 D WIDESHEARS: ----------------------
10-30 02:06:49.513 12372 12374 D WIDESHEARS: In qseecom_faker.c init
10-30 02:06:50.314 12372 12387 D WIDESHEARS: In a ready-to-use cenc call
10-30 02:06:50.314 12372 12387 D WIDESHEARS: old records found, try to retrieve them
10-30 02:06:50.319 12372 12387 D WIDESHEARS: old records received and restored, size=392
10-30 02:06:50.319 12372 12387 D WIDESHEARS: lo_buf paddr: 0x83af0000
10-30 02:06:50.319 12372 12387 D WIDESHEARS: hi_buf paddr: 0x85fd0000
10-30 02:06:50.319 12372 12387 D WIDESHEARS: target paddr: 0x884b0000
10-30 02:06:50.320 12372 12387 D WIDESHEARS: congratulations, not crash, now let's leak some pages
10-30 02:06:50.479 12372 12387 D WIDESHEARS: start to compare signature
10-30 02:06:50.479 12372 12387 D WIDESHEARS: signature 9 perfectly matched, we are done
10-30 02:06:50.479 12372 12387 D WIDESHEARS: init_fast_rw() success, ta_load_base=0x884af000, num_of_pairs=5
10-30 02:06:50.479 12372 12387 D WIDESHEARS: pairs[0] = 0x83560000:0x85d00000
10-30 02:06:50.479 12372 12387 D WIDESHEARS: pairs[1] = 0x83550000:0x85d00000
10-30 02:06:50.479 12372 12387 D WIDESHEARS: pairs[2] = 0x83540000:0x85d00000
10-30 02:06:50.479 12372 12387 D WIDESHEARS: pairs[3] = 0x83530000:0x85d00000
10-30 02:06:50.479 12372 12387 D WIDESHEARS: pairs[4] = 0x83540000:0x85d10000
10-30 02:06:50.482 12372 12387 D WIDESHEARS: value before writing: 0x1
10-30 02:06:50.486 12372 12387 D WIDESHEARS: value after writing: 0x11223344
10-30 02:06:50.486 12372 12387 D WIDESHEARS: snd_cmd() returns 0, cmd=0x61028, num=0x11223344, result=0x0
10-30 02:06:50.486 12372 12387 D WIDESHEARS: value via system api: 0x11223344
10-30 02:06:50.486 12372 12387 D WIDESHEARS: double check successfully
10-30 02:06:50.488 12372 12387 D WIDESHEARS: now let's find the device key to prove that we can read sfs
10-30 02:06:50.496 12372 12387 D WIDESHEARS: origin malloc/free addr logged, they are 0x369886d4, 0x369886e4
10-30 02:06:50.653 12372 12387 D WIDESHEARS: SMC call returned, ret=0x0, cmd_id = 0x6100f, oem_ret = 0x0
10-30 02:06:50.721 12372 12387 D WIDESHEARS: keybox_lvl1 is 43322d57562d3139303730312d313731322d31383338370000000000000000000[redacted]
f4
10-30 02:06:50.721 12372 12387 D WIDESHEARS: Here we are
```

Closing Thoughts

As a developer:

- Separated data/metadata is difficult to trace and error-prone
- Don't use buffers returning to user as a transient storage

As a security researcher:

- Explore blackbox system with a hypotheis-verification workflow

Acknowledgments

- @oldfresher for the opportunity & guidance
- @_2freeman for the teaching on kernel

Thanks