The Baseband Basics:

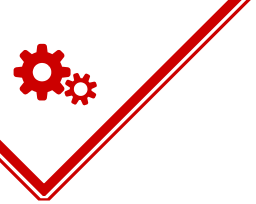
Understanding, Debugging and Attacking the Mediatek Communication Processor

xbase team

Charles Muiruri
@icrackthecode

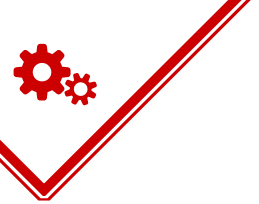
Nitay Artenstein
@nitayart

Anna Dorfman
@___IgniS



In this talk

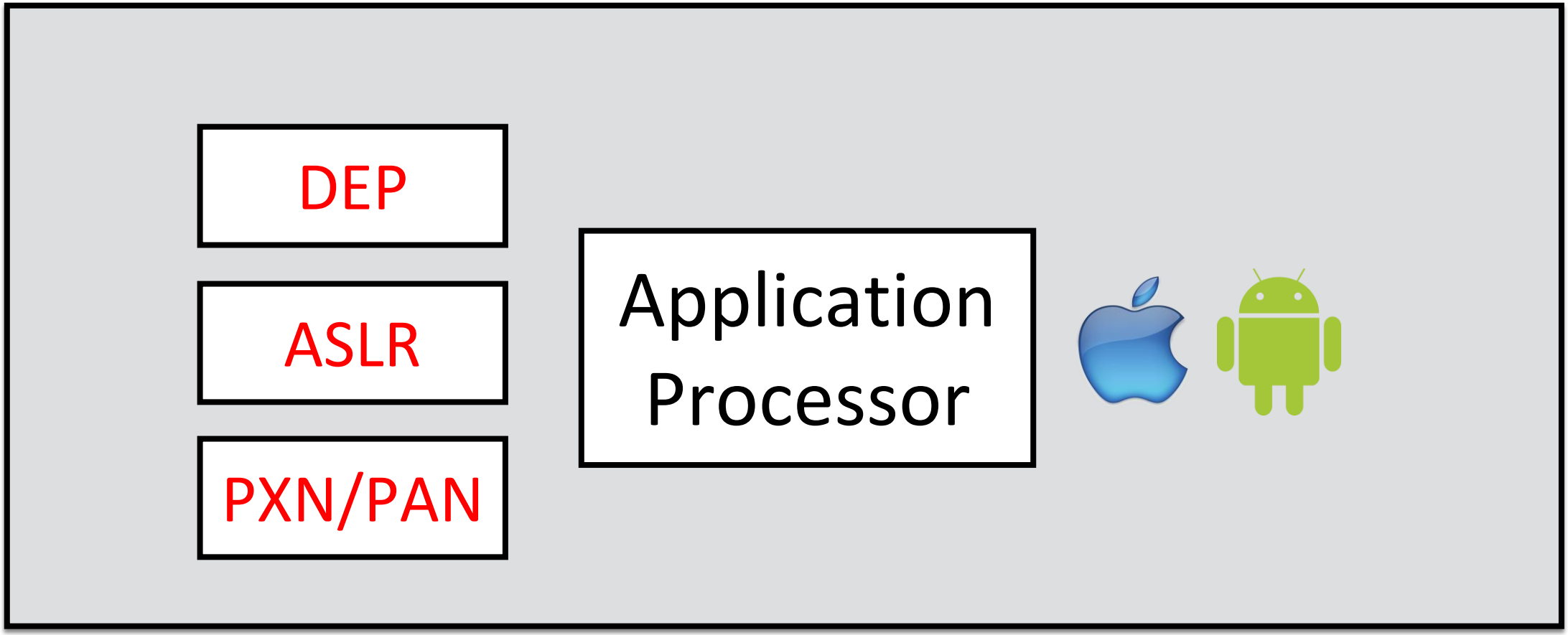
- This is the first public research targeting the Mediatek baseband platform
- Mediatek powers most popular phones in Africa
- Disclaimer: Not ready to disclose 0-days yet
- We will show a DOS though



THE GOAL: A FULLY REMOTE ATTACK

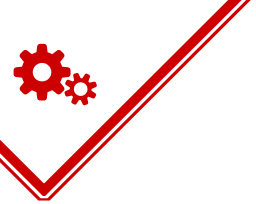
- Fully remote attacks do not involve any interaction on behalf of the victim
- Trigger silently without external indication to the victim
- Will lead (possibly as part of an exploit chain) to full device compromise

ATTACKING ANDROID/IOS



Data Execution Prevention (DEP)

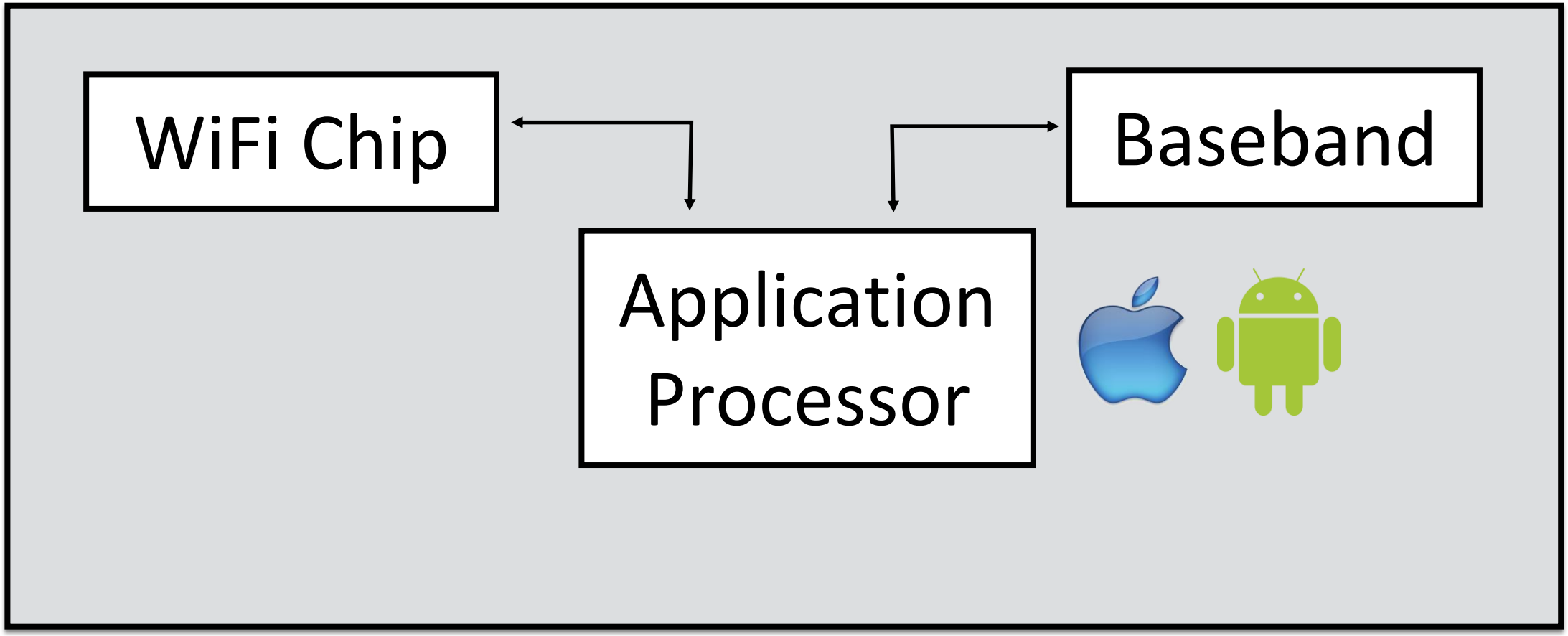
- Prevents certain memory sectors, e.g. the stack, from being executed.
- Hardware-enforced DEP works in conjunction with the NX (Never eXecute) bit on compatible CPUs.



Address Space Layout Randomization (ASLR)

- Randomizes the address space positions of key data areas of a process, including the base of the executable and the positions of the stack, heap and libraries
- Makes guessing the location of ROP gadgets and APIs very difficult.

ATTACKING ANDROID/IOS



BASEBANDS

iPhone	QUALCOMM, intel
Samsung Galaxy and Note	QUALCOMM, SAMSUNG
Google Nexus	QUALCOMM
Some LGs and HTCs	QUALCOMM, MEDIATEK

Popular phones in Africa

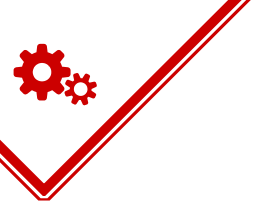
Alcatel	MEDIATEK
Oppo	MEDIATEK
Tecno	MEDIATEK
iTel	MEDIATEK

Attacking Basebands: Introduction

CC

SMS

GPRS

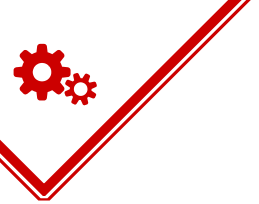


The Challenges

- Closed source, no specs available
- Extensive reversing required
- No debugging interfaces whatsoever, need to build from scratch

MediaTek MTK Kernel Source Code Leaked! Download it from HERE!

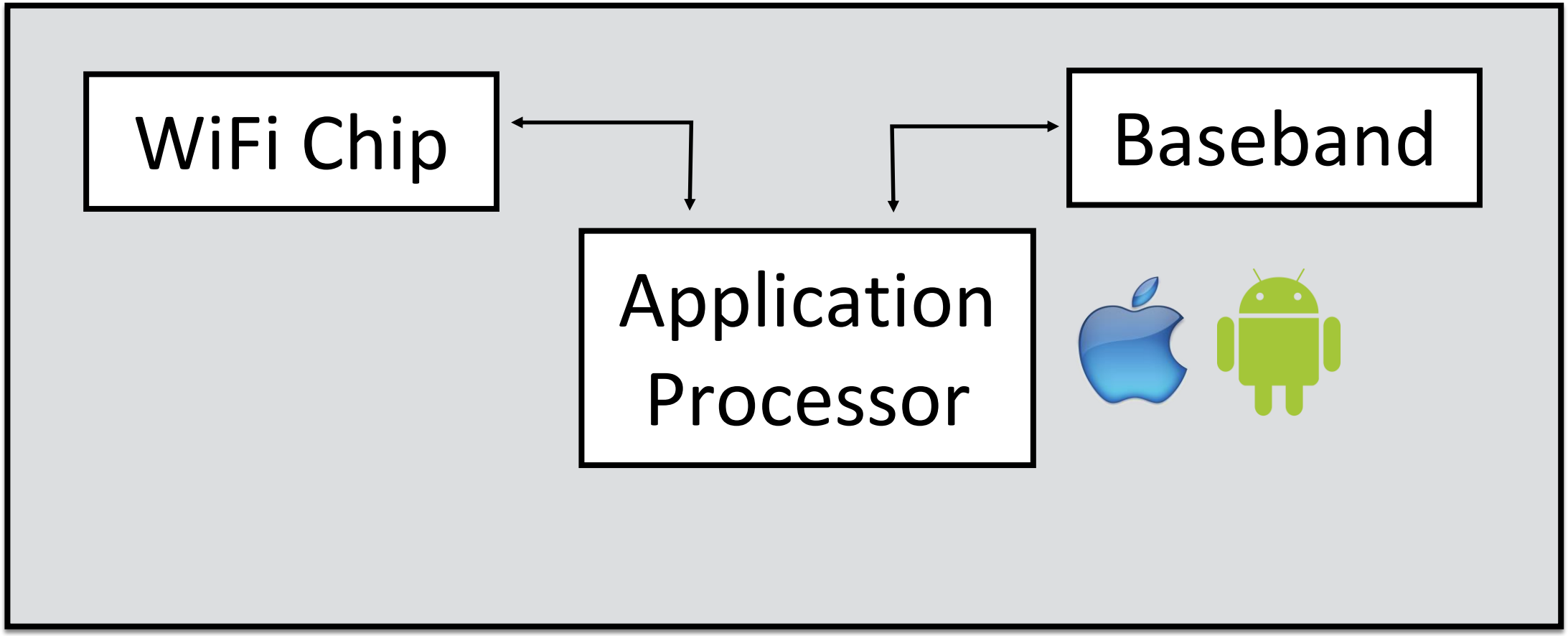
MediaTek's full source code has been leaked! Yeah! now you guys can also build Android Kit Kat and other OS for your MediaTek devices.



Common Attack Vector

- Attacking a communication processor (either baseband or Wi-Fi) allows an attacker to form a bridgehead within a system
- Further escalation is possible and has been done in the past: either through writing directly to the kernel (Broadcom Wi-Fi) or exploiting bugs in the RIL (basebands)

EMBEDDED BRIDGEHEAD DEVICES

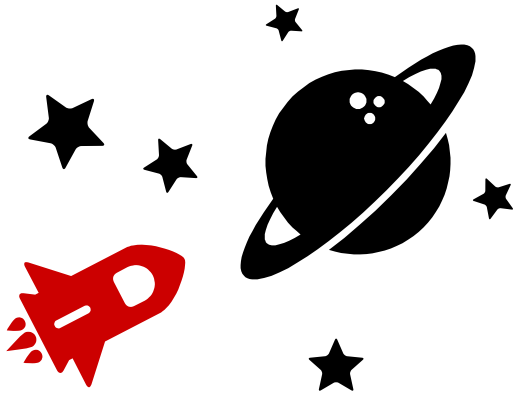


EBDF: Embedded Bridgehead Devices Framework

A little bit history…

- Framework was developed for Nitay's WiFi project

- **Shannon**: the same concept was found to be useful for baseband research
- **Mediatek:** worked after some adaptations



The Heart of the Framework: ***Hooks***

Hooks. General Idea

- Intercept function call
- Execute our piece of code before any function logic is executed
- Trace \ change function data

- Hooks logic

1

Finding a target function

```
RAM:0017EBC0                    ; void __cdecl wlc_bss_mac_event(wlc_info_t *wlc, wlc_bsscfg_t *bsscfg,
RAM:0017EBC0                    wlc_bss_mac_event                       ; CODE XREF: j_wlc_bss_mac_event
RAM:0017EBC0                                                            ; sub_17C7C0+40↑p ...
RAM:0017EBC0
RAM:0017EBC0                    a5              = -0x20
RAM:0017EBC0                    var_1C          = -0x1C
RAM:0017EBC0                    var_18          = -0x18
RAM:0017EBC0                    var_14          = -0x14
RAM:0017EBC0                    var_10          = -0x10
RAM:0017EBC0                    rxframe_data    = -0xC
RAM:0017EBC0                    result          =  0
RAM:0017EBC0                    status          =  4
RAM:0017EBC0                    auth_type       =  8
RAM:0017EBC0                    data            =  0xC
RAM:0017EBC0                    datalen         =  0x10
RAM:0017EBC0
RAM:0017EBC0 000 10 B5                          PUSH            {R4,LR}
RAM:0017EBC2 008 86 B0                          SUB             SP, SP, #0x18
RAM:0017EBC4 020 13 2A                          CMP             R2, #0x13
RAM:0017EBC6 020 08 9C                          LDR             R4, [SP,#0x20+result]
RAM:0017EBC8 020 01 D1                          BNE             loc_17EBCE
RAM:0017EBCA 020 03 2C                          CMP             R4, #3
RAM:0017EBCC 020 0C D0                          BEQ             loc_17EBE8
```

- Hooks logic

2

Hooked function

```
ROM:0017EBBC                    B.W             sub_45CBC
ROM:0017EBC0 ; ---------------------------------------------------------------------------
ROM:0017EBC0                    B.W             loc_1D5254
ROM:0017EBC4 ; ---------------------------------------------------------------------------
ROM:0017EBC4 ; START OF FUNCTION CHUNK FOR sub_1B694A
ROM:0017EBC4
ROM:0017EBC4 loc_17EBC4                                 ; CODE XREF: sub_1B694A+1E46A↓j
ROM:0017EBC4                                            ; ROM:001D52AC↓j
ROM:0017EBC4                                            ; DATA XREF: ...
ROM:0017EBC4                    CMP             R2, #0x13
ROM:0017EBC6                    LDR             R4, [SP,#0x20+arg_0]
ROM:0017EBC8                    BNE             loc_17EBCE
ROM:0017EBCA                    CMP             R4, #3
ROM:0017EBCC                    BEQ             loc_17EBE8
ROM:0017EBCE
ROM:0017EBCE loc_17EBCE                                 ; CODE XREF: sub_1B694A-37D82↑j
ROM:0017EBCE                    STR             R4, [SP,#0x20+var_20]
```

Jump to the hook

- Hooks logic

Trampoline

- Prepare function arguments for our usage
- Jump to our hook main logic
- Execute the original instructions that was overwritten
- Get back to the normal execution

- Hooks:

Trampoline

```
__attribute__((naked)) void
hook(void)
{
    asm(
        "push {r0-r12,lr}\n"
        "ldr r12, hook_info_addr\n"
        "str r0, [r12]\n"
        "str r1, [r12, #0x4]\n"
        "str r2, [r12, #0x8]\n"
        "str r3, [r12, #0xc]\n"
        "str r4, [r12, #0x10]\n"
        "str r5, [r12, #0x14]\n"
        "str r6, [r12, #0x18]\n"
        "str r7, [r12, #0x1c]\n"
        "str r8, [r12, #0x20]\n"
        "str r9, [r12, #0x24]\n"
        "str r10, [r12, #0x28]\n"
        "str fp, [r12, #0x2c]\n"
        "str ip, [r12, #0x30]\n"
        "str sp, [r12, #0x34]\n"
        "str lr, [r12, #0x38]\n"
        "str r0, [r12, #0x44]\n"
        "mov r0, r12\n"
        "ldr r12, entry_addr\n"
        "blx r12\n"
        "pop {r0-r12,lr}\n"
        "push {r4,lr}\n" // Replacement
        "sub  sp, sp, #0x18\n" // Replacement
        "ldr pc, ret_addr\n"
    "hook_info_addr:\n"
        ".long " HOOK_INFO_ADDR "\n"
    "entry_addr:\n"
        ".long " ENTRY_FUNC_ADDR "\n"
    "ret_addr:\n"
        ".long " RET_ADDR "\n"
    );
}
```

Prepare function arguments for the hook

Jmp to main hook logic

Original instructions

- Hooks logic

Addresses

```
HOOK_FUNC_ADDR = 0x1d5254
ENTRY_FUNC_ADDR = 0x161b00
```

- Hooks logic

4

Trampoline After applying

```
ROM:001D5254
ROM:001D5254 loc_1D5254                              ; CODE XREF: ROM:0017EBC0↑j
ROM:001D5254                 PUSH.W          {R0-R12,LR}
ROM:001D5258                 LDR.W           R12, =dword_1D7D94
ROM:001D525C                 STR.W           R0, [R12]
ROM:001D5260                 STR.W           R1, [R12,#(dword_1D7D98 - 0x1D7D94)]
ROM:001D5264                 STR.W           R2, [R12,#(dword_1D7D9C - 0x1D7D94)]
ROM:001D5268                 STR.W           R3, [R12,#(dword_1D7DA0 - 0x1D7D94)]
ROM:001D526C                 STR.W           R4, [R12,#(dword_1D7DA4 - 0x1D7D94)]
ROM:001D5270                 STR.W           R5, [R12,#(dword_1D7DA8 - 0x1D7D94)]
ROM:001D5274                 STR.W           R6, [R12,#(dword_1D7DAC - 0x1D7D94)]
ROM:001D5278                 STR.W           R7, [R12,#(dword_1D7DB0 - 0x1D7D94)]
ROM:001D527C                 STR.W           R8, [R12,#(dword_1D7DB4 - 0x1D7D94)]
ROM:001D5280                 STR.W           R9, [R12,#(dword_1D7DB8 - 0x1D7D94)]
ROM:001D5284                 STR.W           R10, [R12,#(dword_1D7DBC - 0x1D7D94)]
ROM:001D5288                 STR.W           R11, [R12,#(dword_1D7DC0 - 0x1D7D94)]
ROM:001D528C                 STR.W           R12, [R12,#(dword_1D7DC4 - 0x1D7D94)]
ROM:001D5290                 STR.W           SP, [R12,#(dword_1D7DC8 - 0x1D7D94)]
ROM:001D5294                 STR.W           LR, [R12,#(dword_1D7DCC - 0x1D7D94)]
ROM:001D5298                 STR.W           R0, [R12,#(dword_1D7DD8 - 0x1D7D94)]
ROM:001D529C                 MOV             R0, R12
ROM:001D529E                 LDR.W           R12, =(sub_1D5354+1)
ROM:001D52A2                 BLX             R12 ; sub_1D5354
ROM:001D52A4                 POP.W           {R0-R12,LR}
ROM:001D52A8                 PUSH            {R4,LR}
ROM:001D52AA                 SUB             SP, SP, #0x18
ROM:001D52AC                 LDR.W           PC, =(loc_17EBC4+1)
ROM:001D52AC ; ---------------------------------------------------------------------------
ROM:001D52B0 off_1D52B0      DCD dword_1D7D94        ; DATA XREF: ROM:001D5258↑r
ROM:001D52B4 off_1D52B4      DCD sub_1D5354+1        ; DATA XREF: ROM:001D529E↑r
ROM:001D52B8 off_1D52B8      DCD loc_17EBC4+1        ; DATA XREF: ROM:001D52AC↑r
ROM:001D52BC                 DCB 0xBC ; ¼
ROM:001D52BD                 DCB 0x3C ; <
```

Jmp to main hook logic

- Hooks logic

Hook_info structure

```
struct hook_info {
    struct orig_regs regs;
    void *hook_func;
    void *hook_output;
    unsigned int data_idx;
    unsigned int data[HOOK_DATA_SIZE];
    unsigned int lock;
};
```

- Hooks logic

Hook main logic

```
39  void
40  entry(struct hook_info *h)
41  {
42      unsigned int stub;
43      struct arg_info *args = (struct arg_info *) h->data;
44      unsigned int a1 = h->regs.ARM_r0;
45      unsigned int a2 = h->regs.ARM_r1;
46      unsigned int a3 = h->regs.ARM_r2;
47      unsigned int a4 = h->regs.ARM_r3;
48      unsigned int sp = h->regs.ARM_sp;
49      unsigned int lr = GET_OFFSET_UINT(sp, 0x34);
50
51      printf("wlc_bss_mac_event: a1: %08x | a2: %08x | a3: %08x | a4: %08x | lr: %08x\n", a1, a2, a3, a4, lr);
52
53      printf("Stack trace:\n");
54      print_stack(&stub, 0x100);
55
56      return;
57  }
```

- Hooks logic

Hook main
After applying

```
sub_1D5354

var_30= -0x30
var_2C= -0x2C
var_28= -0x28
var_20= -0x20
arg_120=  0x120

PUSH.W          {R4-R8,LR}
MOV             R7, R0
LDR.W           R8, =(off_1D5438 - 0x1D536C)
SUB             SP, SP, #0x18
LDR             R0, =(aWlc_bss_mac_ev - 0x1D5372)
MOV.W           R6, #0xFB00
LDR             R3, [R7,#4]
ADD             R8, PC ; off_1D5438
LDR.W           LR, [R7,#0x34]
ADD             R0, PC  ; "wlc_bss_mac_event: a1: %08x | a2: %08x "..
MOV             R2, R3
LDR             R3, [R7,#0xC]
ADD             R4, SP, #0x30+var_20
LDR.W           R12, [R7,#8]
ADD             R5, SP, #0x30+arg_120
LDR             R1, [R7]
MOVT.W          R6, #0x1F
STR             R3, [SP,#0x30+var_30]
MOV             R3, R12
LDR             R7, [R7,#0x14]
LDRB            R7, [R7,#8]
STR             R7, [SP,#0x30+var_2C]
LDR.W           R7, [LR,#0x34]
STR             R7, [SP,#0x30+var_28]
LDR.W           R7, [R8]
BLX             R7 ; sub_162BB8
LDR             R0, =(aStackTrace - 0x1D53A4)
LDR.W           R3, [R8] ; sub_162BB8
LDR             R7, =(aFn08x08x_0 - 0x1D53AC)
ADD             R0, PC  ; "Stack trace:\n"
LDR.W           R8, =(off_1D5438 - 0x1D53AE)
BLX             R3 ; sub_162BB8
ADD             R7, PC  ; "fn: %08x: %08x\n"
ADD             R8, PC ; off_1D5438
B               loc_1D53B2
```

Hooks condition

Read \ Write Primitive 🔨

Read \ Write Primitive

	How?	Signature check?
Wifi chip	**dhdutil**	**No ☺**
Shannon	**Bug + ATcommands**	**Yes ☹**
Mediatek	**Trick**	**No ☺**

Mediatek: Achieving Debugging Abilities

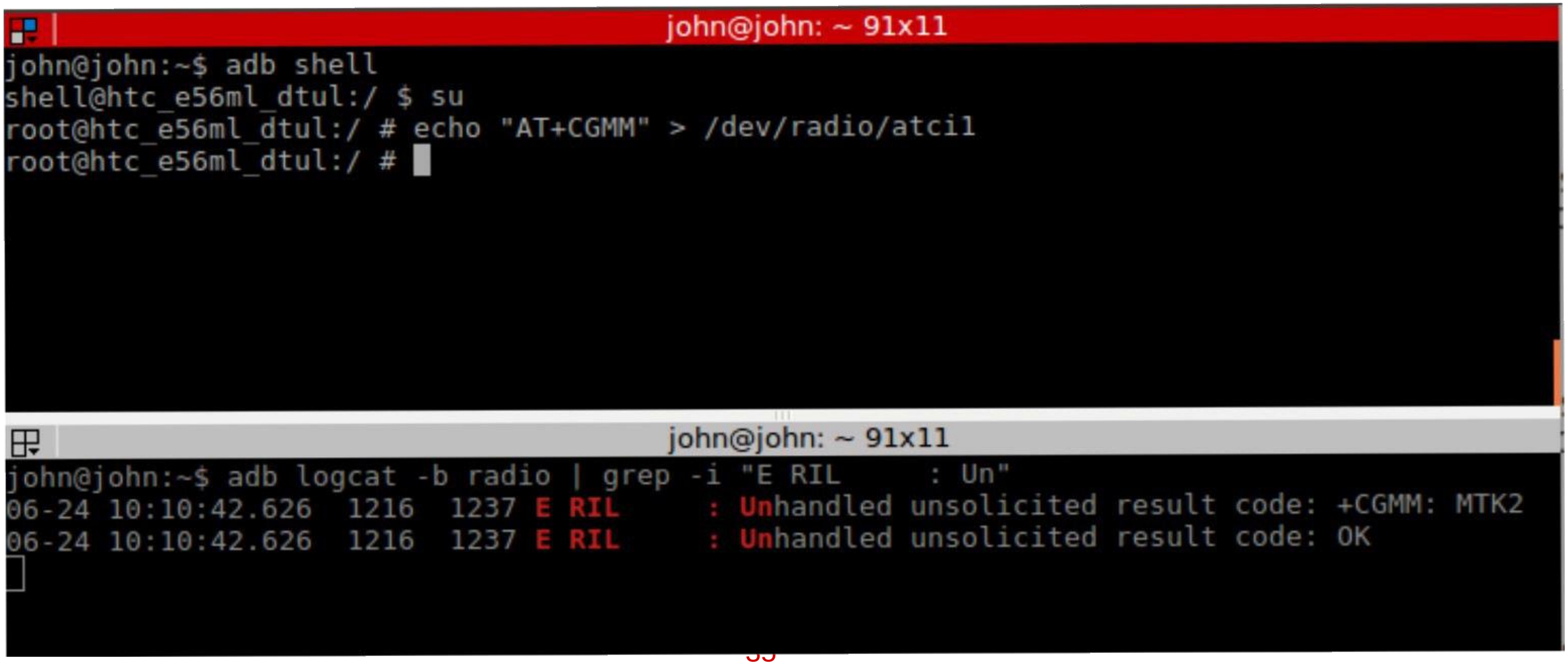
Conditions of a debugger

- Must have the ability allow reading and writing to memory.
- Must have the ability to change memory permissions of various regions
- Must have ability to change code on runtime

Modem / AT Commands

- Modems use a set of commands that enable them to do various tasks called Modem/AT commands
- AT commands take in a command and respond back to RILD which logs them to the the radio logs.
- Syntax AT+ Command=arguments (...mainly)

AT commands


john@john: ~ 91x11
john@john:~$ adb shell
shell@htc_e56ml_dtul:/ $ su
root@htc_e56ml_dtul:/ # echo "AT+CGMM" > /dev/radio/atci1
root@htc_e56ml_dtul:/ #
john@john: ~ 91x11
john@john:~$ adb logcat -b radio | grep -i "E RIL : Un"
06-24 10:10:42.626 1216 1237 E RIL : Unhandled unsolicited result code: +CGMM: MTK2
06-24 10:10:42.626 1216 1237 E RIL : Unhandled unsolicited result code: OK

AT commands function table

```
ROM:006BD89C E5 64 31 00                    DCD rmmi___hdlr+1
ROM:006BD8A0                        ; RMMI_EXT_CMD_FUNCTION rmmi_extended_validator_ft[]
ROM:006BD8A0 71 74 31 00            rmmi_extended_validator_ft DCD rmmi_cacm_hdlr+1
ROM:006BD8A0                                                   ; DATA XREF:
ROM:006BD8A0                                                   ; rmmi_exten
ROM:006BD8A4 11 75 31 00                    DCD rmmi_camm_hdlr+1
ROM:006BD8A8 15 76 31 00                    DCD rmmi_clcc_hdlr+1
ROM:006BD8AC 49 76 31 00                    DCD rmmi_vts_hdlr+1
ROM:006BD8B0 D9 76 31 00                    DCD rmmi_chup_hdlr+1
ROM:006BD8B4 C5 77 31 00                    DCD rmmi_chld_hdlr+1
ROM:006BD8B8 F5 78 31 00                    DCD rmmi_ecpi_hdlr+1
ROM:006BD8BC 4D 79 31 00                    DCD rmmi_eccp_hdlr+1
ROM:006BD8C0 8D 79 31 00                    DCD rmmi_caoc_hdlr+1
ROM:006BD8C4 8D 7A 31 00                    DCD rmmi_ccwe_hdlr+1
ROM:006BD8C8 09 7B 31 00                    DCD rmmi_ccug_hdlr+1
ROM:006BD8CC D9 7B 31 00                    DCD rmmi_cpas_hdlr+1
ROM:006BD8D0 51 7C 31 00                    DCD rmmi_cvhu_hdlr+1
ROM:006BD8D4 C1 7C 31 00                    DCD rmmi_ctfr_hdlr+1
```

```
1  void __fastcall rmmi_gmm_hdlr(rmmi_string_struct *source_string)
2  {
3    rmmi_string_struct *source_string_ptr; // r5
4    rmmi_context_struct *context; // r6
5    signed int v3; // r4
6    char *v4; // r1
7    kal_uint16 v5; // r3
8    int v6; // [sp+0h] [bp-88h]
9    char v7; // [sp+50h] [bp-38h]
10
11   source_string_ptr = source_string;
12   context = rmmi_ptr_g;
13   dhl_trace(0, 0, 246415620, 0);
14   if ( source_string_ptr->field_D == 4 )
15   {
16     v3 = sub_30DB66(source_string_ptr->src_id, 1, (int)&v7);
17     if ( v3 == 1 )
18     {
19       context->arg_list[0] = &v7;
20       if ( sbp_query_md_feature(38) == 1 )
21         v4 = "%s";
22       else
23         v4 = "+CGMM: %s";
24       v5 = rmmi_fast_string_print((kal_uint8 *)&v6, (kal_uint8 *)v4, context->arg_list, 1u);
25     }
26     else
27     {
28       v5 = 0;
29     }
30   }
31   else
32   {
33     v5 = 0;
34     v3 = 0;
35   }
36   rmmi_final_rsp_generator(source_string_ptr->src_id, (kal_bool)v3, (kal_uint8 *)&v6, v5);
37 }
```

Hooking the handlers (Strategy)

- **Read** => AT+command=read=size=address
- **Write** => AT+command=write=address=raw bytes
- **Memory allocation** => AT+command=alloc=size

```
john@john:~$ adb logcat -b radio | grep "unsoli"
06-22 07:06:41.612  1216  1243 E RIL     : Unhandled unsolicited
result code: 600021616019ee
06-22 07:06:41.612  1216  1243 E RIL     : Unhandled unsolicited
result code: [+] End of dump
06-22 07:06:41.612  1216  1243 E RIL     : Unhandled unsolicited
result code:  1000 bytes from 0x00000
06-22 07:06:41.613  1216  1243 E RIL     : Unhandled unsolicited
result code: [+] dff81cf0dff81cf0dff81cf0dff81cf0dff81cf0dff81cf0
dff81cf0dff81cf0892a00003b3259005732590 07f3259009d3259000000000a
40c0370dcbc100070b5054619ee1d4f0020664219e019ee1d0fa0422cbfc4eb00
003018e5f0f6fc09a3d3e9002316f383f109a3d3e90023e5f0b0fb08a3d3e9002
316f3a3f216f395f7a842e3d370bde0229ca98ec4f83fe0229ca98ec4e83f0000
000000408f402de9f0411746424a06460c46136833b9404bd8061cbf23f01f032
03313603e4b15681b68d3b914f01f0125f0704503d024f01f00103101e0204610
211f3121f01f0102f013fe04f07043b3f1704f05d124f0704420460121 08f11cf
9002110222846faf392ef002315f01f012b70284603d025f01f00103100e01021
1f3121f01f0102f0f3fd237814f01f0143f00103656023 7003d024f01f0010310
1e0204610211f3121f01f0102f0e0fd01231b4ab34013600c3a11680b42fcd116
4a12683ab9012106f10a029140154a10680142fcd1144a4ff47a7842f82640002
5124e08fb07f8013f53652021204602f084fd23780120d90709d5fff752ffb742
f3d801354545f0d90020bde8f081bde8f08140cab2f110cab2f1383c9bf15c010
e8000020e8000010e80358941002de9f0471746704a80460c46136833b96e4bd9
061cbf23f01f03203313606c4b
06-22 07:06:41.613  1216  1243 E RIL     : Unhandled unsolicited
result code: [+] End of dump
```

Changing memory permissions

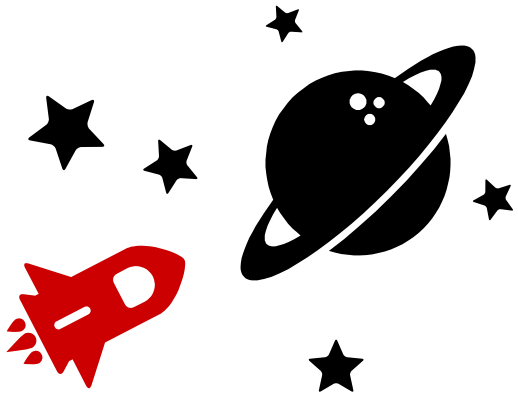
- The goal is to make the whole memory RWX to the MPU
- Make it possible to write code and execute it
- AT+command=permission=region start address

```
// Set Region;
asm volatile ("MCR p15, 0, %0, c6, c2, 0" : : "r"(region_number));

// Read Region Access Control Register
asm volatile ("MRC p15, 0, %0, c6, c1, 4" : "=r"(dracr));

// Region Base Address
asm volatile ("MRC p15, 0, %0, c6, c1, 0" : "=r"(drbar));
dracr &= ~0x1700;
dracr |= 0x300;

// Write Region Access Control Register
asm volatile ("MCR p15, 0, %0, c6, c1, 4" : : "r"(dracr));
```



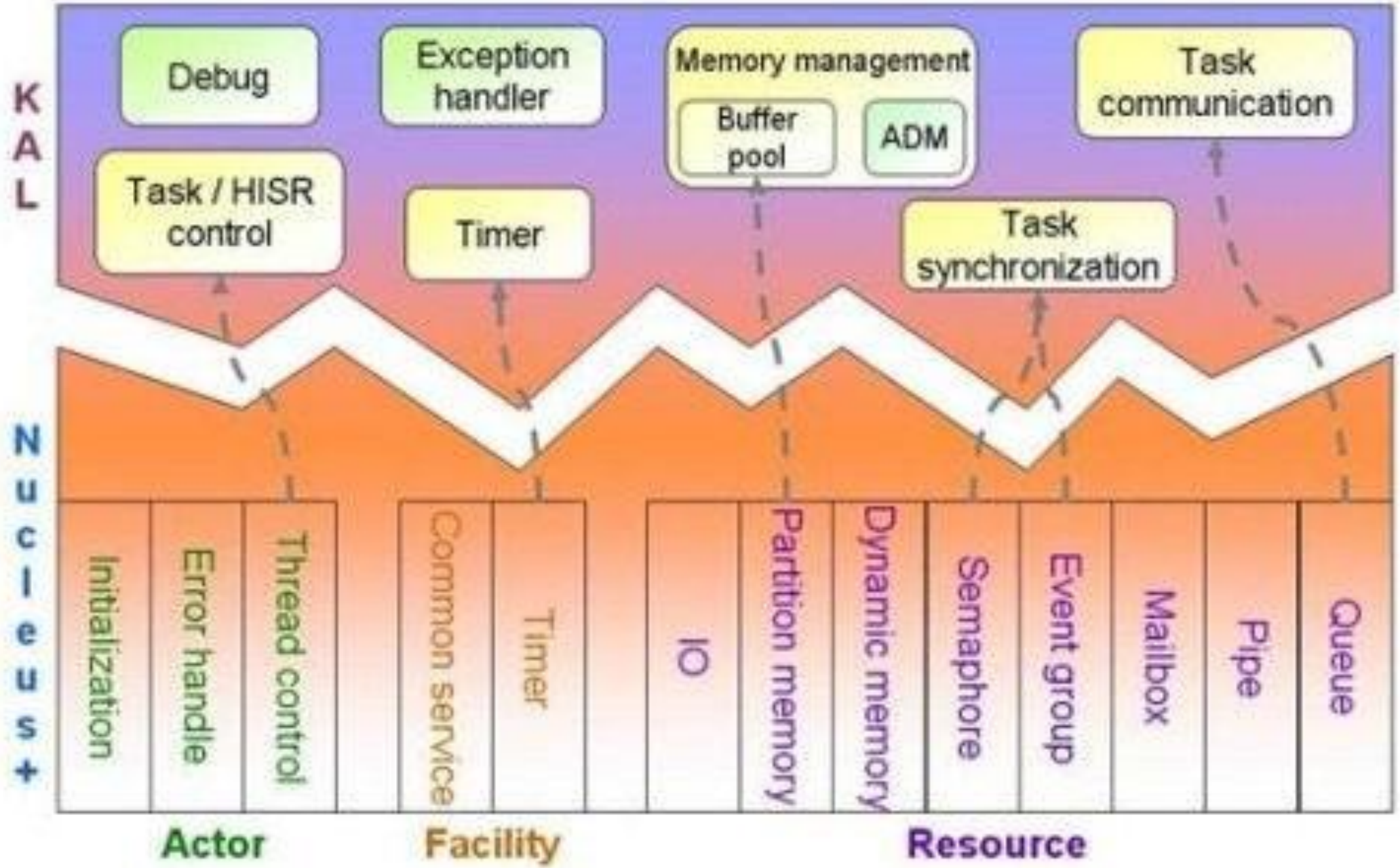
Architecture

MAUI Runtime Environment

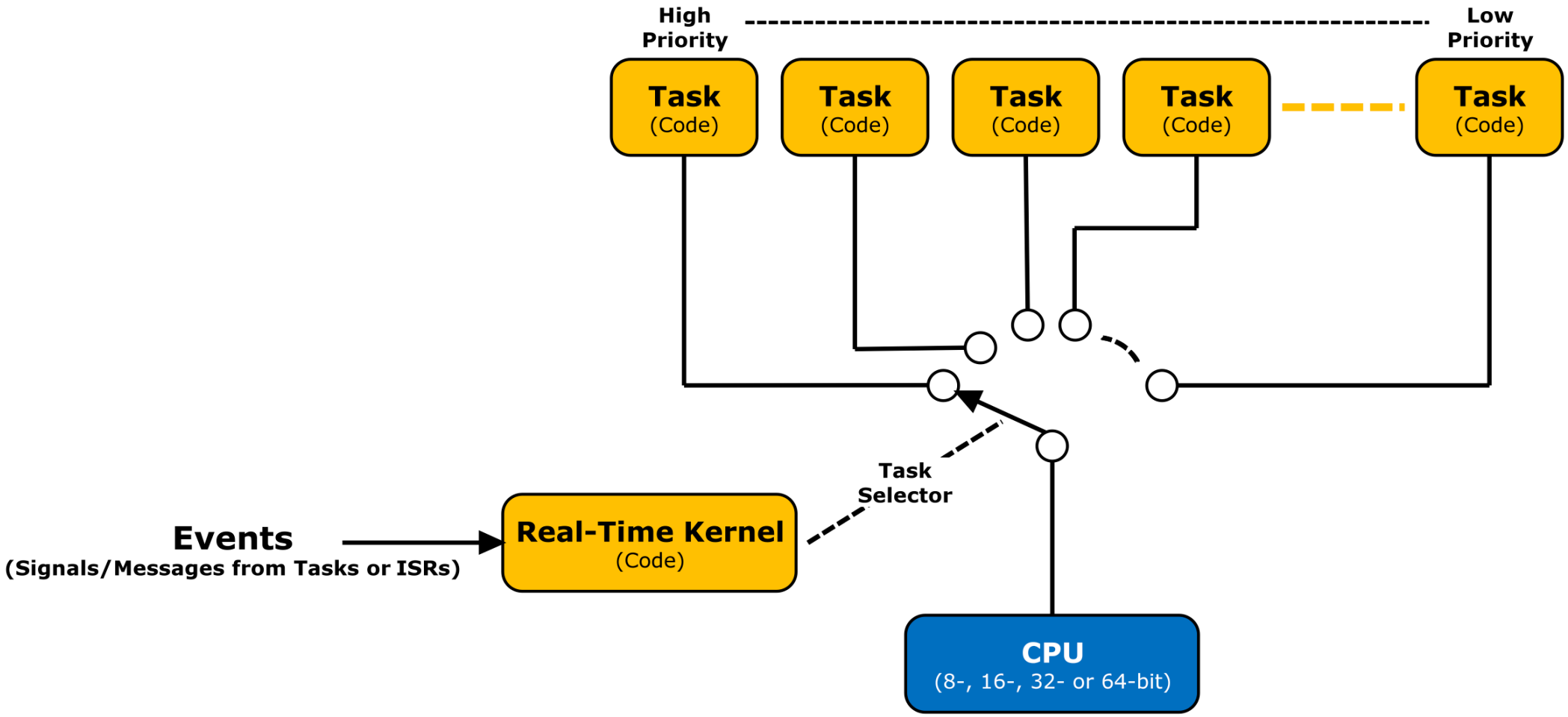
- Mediatek implements "standalone" applications for their MAUI/Nucleus OS.
- Use Kernel Adaptation Layer(KAL) between Real Time Operating System (RTOS) and upper layer applications

KAL API's

- Task management
- Task synchronization
- Task communication
- Timer Management
- Memory management


KAL
Debug
Exception handler
Memory management
Buffer pool
ADM
Task communication
Task / HISR control
Timer
Task synchronization
Nucleus+
Initialization
Error handle
Thread control
Common service
Timer
IO
Partition memory
Dynamic memory
Semaphore
Event group
Mailbox
Pipe
Queue
Actor
Facility
Resource

Task Management

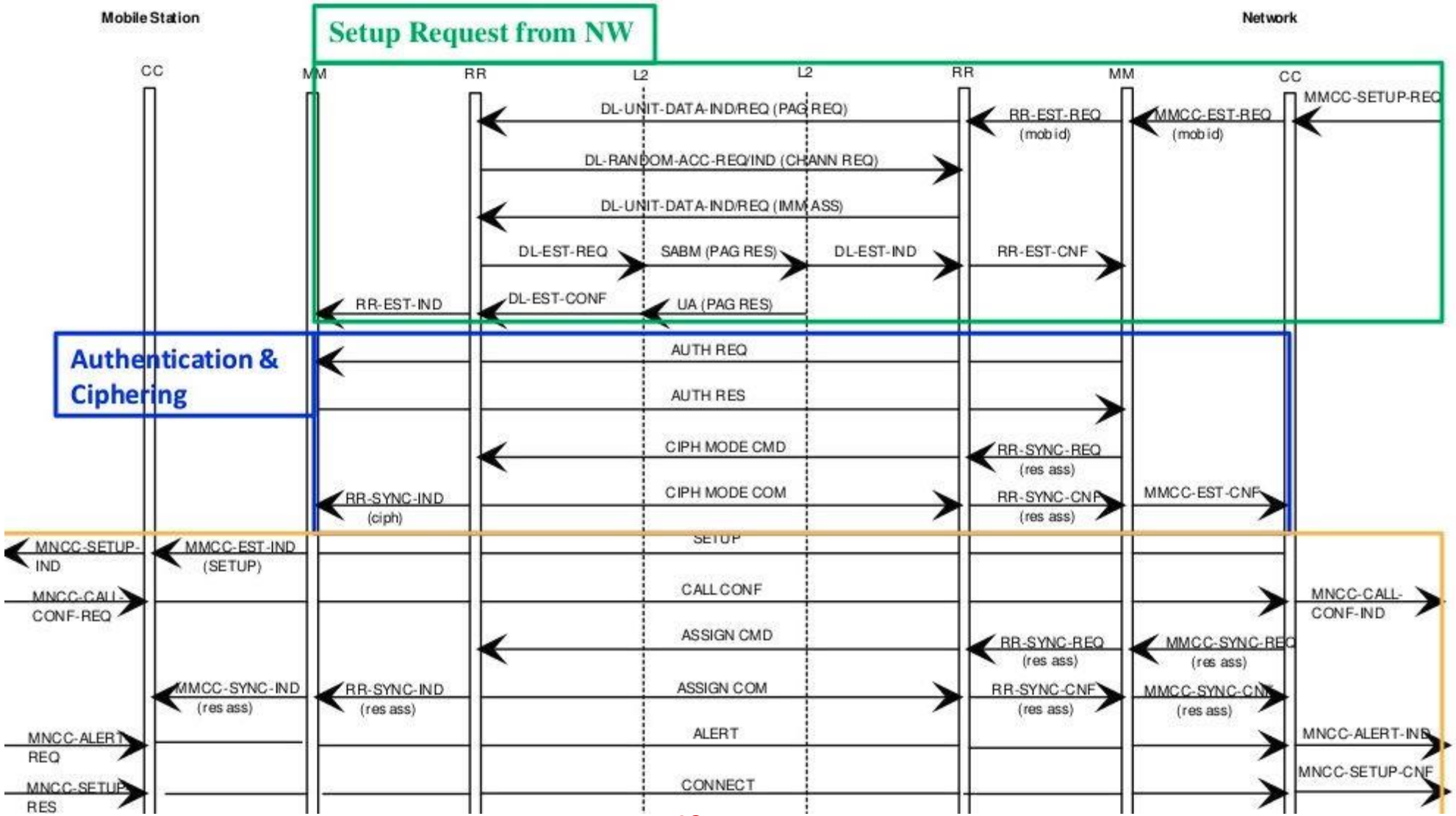


ILM_STRUCT

```
typedef struct ilm_struct {
   module_type       src_mod_id;
   module_type       dest_mod_id;
   sap_type          sap_id;
   msg_type          msg_id;
   local_para_struct *local_para_ptr;
   peer_buff_struct  *peer_buff_ptr;
} ilm_struct;
```

Reverse Engineering the CC Protocol

CC Protocol (Call Control)



CC task routing

```
void __fastcall cc_route_ilm(ilm_struct *a1)
{
  local_para_struct *v1; // r1
  ilm_struct *v2; // r5
  trace_class_enum v3; // r0
  kal_uint32 v4; // r2
  int v5; // r4
  unsigned __int8 v6; // r3
  int v7; // r0
  int v8; // r0
  int v9; // r0
  void (__fastcall *v10)(local_para_struct *, _DWORD); // [sp+8h] [bp-28h]
  char v11; // [sp+Ch] [bp-24h]
  unsigned __int8 v12; // [sp+Fh] [bp-21h]

  v1 = a1->local_para_ptr;
  v2 = a1;
  v10 = 0;
  v12 = 7;
  if ( !v1 )
  {
```

CC task routing

- Tasks are chosen depending on the destination module’s ID
- Each task has an array of messages. These are sent one by one to the relevant message handlers
- Some of these messages are under an attacker’s full control.

```
ROM:00710DB0 8D 05 60 00     cc_peer_message_handlers DCD cc_peer_alert_hdlr+1
ROM:00710DB0                                          ; DATA XREF: cc_get_peer_msg_hdlr:loc_60159A↑
ROM:00710DB0                                          ; ROM:off_601614↑o
ROM:00710DB4 51 06 60 00                 DCD cc_peer_call_proceeding_hdlr+1
ROM:00710DB8 89 04 60 00                 DCD cc_peer_connect_hdlr+1
ROM:00710DBC 51 0C 60 00                 DCD sub_600C50+1
ROM:00710DC0 B5 0E 60 00                 DCD cc_peer_progress_hdlr+1
ROM:00710DC4 00 00 00 00                 DCD 0
ROM:00710DC8 00 00 00 00                 DCD 0
ROM:00710DCC 59 0D 60 00                 DCD cc_peer_setup_hdlr+1
ROM:00710DD0 55 24 60 00                 DCD cc_peer_modify_hdlr+1
ROM:00710DD4 D1 24 60 00                 DCD cc_peer_modify_complete_hdlr+1
ROM:00710DD8 55 25 60 00                 DCD cc_peer_modify_reject_hdlr+1
ROM:00710DDC F1 25 60 00                 DCD sub_6025F0+1
ROM:00710DE0 D5 1A 60 00                 DCD sub_601AD4+1
ROM:00710DE4 65 1B 60 00                 DCD sub_601B64+1
ROM:00710DE8 51 1D 60 00                 DCD sub_601D50+1
ROM:00710DEC E1 1D 60 00                 DCD sub_601DE0+1
ROM:00710DF0 5D ED 5F 00                 DCD cc_peer_disconnect_hdlr+1
ROM:00710DF4 59 EF 5F 00                 DCD cc_peer_release_hdlr+1
ROM:00710DF8 69 F1 5F 00                 DCD cc_peer_release_complete_hdlr+1
ROM:00710DFC 09 26 60 00                 DCD sub_602608+1
ROM:00710E00 F1 C0 5F 00                 DCD sub_5FC0F0+1
ROM:00710E04 89 BE 5F 00                 DCD sub_5FBE88+1
ROM:00710E08 AD BF 5F 00                 DCD sub_5FBFAC+1
ROM:00710E0C 39 FC 5F 00                 DCD sub_5FFC38+1
ROM:00710E10 61 FD 5F 00                 DCD sub_5FFD60+1
ROM:00710E14 CD FC 5F 00                 DCD sub_5FFCCC+1
ROM:00710E18 41 A9 5F 00                 DCD sub_5FA940+1
ROM:00710E1C 6D 6F 64 65+    aModemNasCcSs_2 DCB "modem/nas/cc-ss/cc/src/cc_hold_proc.c",0
```

```
void *__fastcall cc_form_app_setup_ind(void *result, kal_uint32 *a2, int a3)
{
  _BYTE *setup_ind; // r5
  kal_uint32 *v4; // r4
  unsigned __int8 *buff; // r6

  setup_ind = result;
  v4 = a2;
  buff = (unsigned __int8 *)a3;
  if ( *((_BYTE *)a2 + 4) )
  {
    *((_BYTE *)result + 5) = 1;
    *((_BYTE *)result + 6) = *(_BYTE *)a2[2];
  }
  if ( *((_BYTE *)a2 + 12) )
  {
    *((_BYTE *)result + 7) = 1;
    result = cc_form_app_bc_from_peer((_BYTE *)result + 8, (void *)a2[4]);
  }
  if ( *((_BYTE *)v4 + 20) )
  {
    setup_ind[26] = 1;
    result = cc_form_app_bc_from_peer(setup_ind + 27, (void *)v4[6]);
  }
  if ( *((_BYTE *)v4 + 28) )
  {
    result = (void *)cc_form_app_fac_ie(setup_ind + 48, buff);
    setup_ind[45] = (_BYTE)result;
  }
  if ( *((_BYTE *)v4 + 36) )
  {
    setup_ind[56] = 1;
    result = cc_form_app_progress_indicator(setup_ind + 57, v4[10]);
  }
  if ( *((_BYTE *)v4 + 44) )
  {
    setup_ind[59] = 1;
    setup_ind[60] = *(_BYTE *)v4[12];
  }
  if ( *((_BYTE *)v4 + 52) )
```

Saving BC IE

```
void *__fastcall cc_form_app_bc_from_peer(_BYTE *a1, void *a2)
{
  unsigned int v2; // r2

  v2 = *(unsigned __int8 *)a2;
  *a1 = v2;
  return memcpy_2(a1 + 2, *((const void **)a2 + 1), v2);
}
```

Exploitation

DOS on the baseband

- The baseband allocates memory in the control buffer.
- Among the bugs we found was a DOS bug which involved memory allocation via **get_ctrl_buffer_ext**
- When no heap memory is available, **get_ctrl_buffer_ext** fails to handle the failure gracefully and crashes the whole system
- Causing large allocations via CC messages reliably caused a system crash

DEMO

Future Steps

- Currently working on a potential RCE bug
- If successful, will release around end of year
- Escalation to application processor: Previous research by Comsecuris showed this to be possible by exploiting bugs in the RIL daemon

QUESTIONS?