

# Exploiting Qualcomm WLAN And Modem Over-The-Air

Xiling Gong, Peter Pi  
Tencent Blade Team



# About Us

## **Xiling Gong (@GXiling)**

Senior security researcher at Tencent Blade Team.

Focus on Android Security, Qualcomm Firmware Security.

Speaker of BlackHat, CanSecWest.

## **Peter Pi(@tencent\_blade)**

Senior security researcher at Tencent Blade Team.

Find many vulnerabilities of vendors like Google, Microsoft, Apple, Qualcomm, Adobe and Tesla.

The #1 Researcher of Google Android VRP in year 2016.

Speaker of BlackHat, CanSecWest, HITB, GSEC and Hitcon.

# About Tencent Blade Team



- Founded by Tencent Security Platform Department in 2017
- Focus on security research in the areas of AIoT, Mobile devices, Cloud virtualization, Blockchain, etc
- Report 200+ vulnerabilities to vendors such as Google, Apple, Microsoft, Amazon
- We talked about how to break Amazon Echo at DEFCON26
- Blog: <https://blade.tencent.com>

# Agenda

- Introduction and Related Work
- The Debugger
- Reverse Engineering and Attack Surface
- Vulnerability and Exploitation
- Escaping into Modem
- Escaping into Kernel
- Stability of Exploitation
- Conclusions

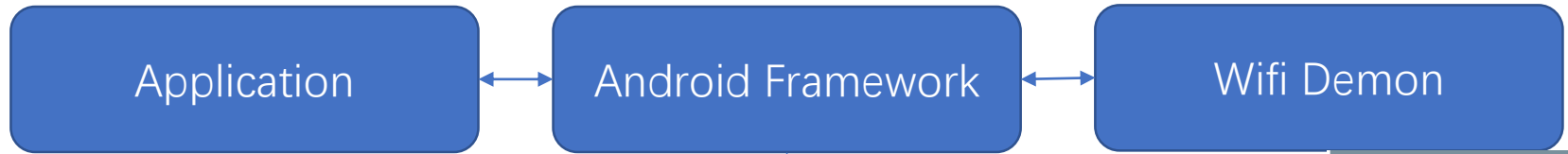
# Agenda

- **Introduction and Related Work**
- The Debugger
- Reverse Engineering and Attack Surface
- Vulnerability and Exploitation
- Escaping into Modem
- Escaping into Kernel
- Stability of Exploitation
- Conclusions

# Introduction

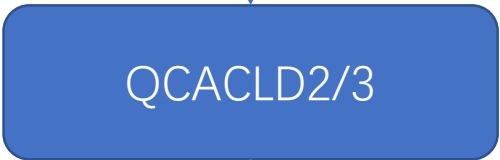
- Broadcom WIFI Chip
  - 2017, Gal Beniamini
    - Over The Air: Exploiting Broadcom's Wi-Fi Stack
  - 2017, Nitay Artenstein, BlackHat USA 2017
    - BROADPWN: REMOTELY COMPROMISING ANDROID AND IOS VIA A BUG IN BROADCOM'S WI-FI CHIPSETS
- Marvel WIFI Chip
  - 2019, Denis Selyanin
    - Zero Nights 2018 , Researching Marvell Avastar Wi-Fi: from zero knowledge to over-the-air zero-touch RCE
    - Blog 2019, Remotely compromise devices by using bugs in Marvell Avastar Wi-Fi: from zero knowledge to zero-click RCE
- **How about Qualcomm WIFI?**

# Qualcomm WLAN (MSM8998)



Linux Kernel

Full MAC Layer



Baseband Subsystem



CVE	References	Type	Severity	Component
CVE-2018-11940	A-79377832 <a href="#">QC-CR#2254946</a>	N/A	Critical	WLAN HOST
CVE-2017-17772	A-72957385 <a href="#">QC-CR#2153003 [2]</a>	N/A	High	WLAN HOST
CVE-2018-11294	A-109741680 <a href="#">QC-CR#2197481</a>	N/A	High	WLAN HOST
CVE-2018-5855	A-77527719 <a href="#">QC-CR#2193421</a>	N/A	High	WLAN HOST
CVE-2018-11299	A-109741946 <a href="#">QC-CR#2186953</a>	N/A	High	WLAN HOST
CVE-2018-11826	A-111127853 <a href="#">QC-CR#2205957</a>	N/A	High	WLAN HOST

# Agenda

- Introduction and Related Work
- **The Debugger**
- Reverse Engineering and Attack Surface
- Vulnerability and Exploitation
- Escaping into Modem
- Escaping into Kernel
- Stability of Exploitation
- Conclusions

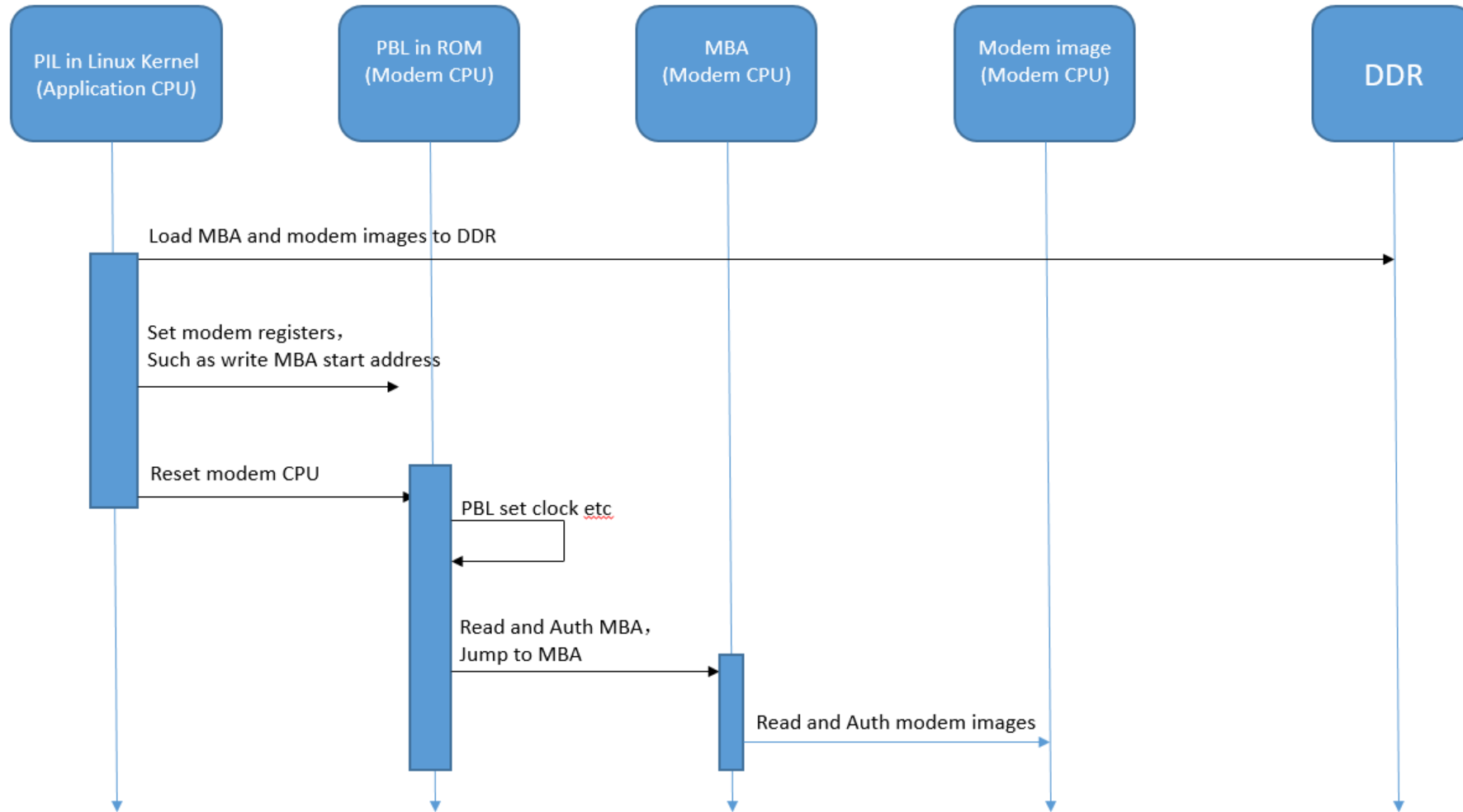


# MBA and Modem images

- Modem Boot Authenticator
- mba.mbn
- modem.mdt
- modem.b00 – modem.b20
- [Image format](#)

```
sailfish:/firmware/radio $ ls -l
total 55824
-r--r----- 1 system root      244 1980-01-01 00:00 mba.b00
-r--r----- 1 system root    4584 1980-01-01 00:00 mba.b01
-r--r----- 1 system root  154064 1980-01-01 00:00 mba.b02
-r--r----- 1 system root   10144 1980-01-01 00:00 mba.b03
-r--r----- 1 system root   31588 1980-01-01 00:00 mba.b04
-r--r----- 1 system root    896 1980-01-01 00:00 mba.b05
-r--r----- 1 system root  213888 1980-01-01 08:00 mba.mbn
-r--r----- 1 system root    4828 1980-01-01 00:00 mba.mdt
-r--r----- 1 system root    884 1980-01-01 00:00 modem.b00
-r--r----- 1 system root    5224 1980-01-01 00:00 modem.b01
-r--r----- 1 system root    5460 1980-01-01 08:00 modem.b02
-r--r----- 1 system root   196608 1980-01-01 08:00 modem.b03
-r--r----- 1 system root  2816788 1980-01-01 08:00 modem.b04
-r--r----- 1 system root  3288155 1980-01-01 08:00 modem.b05
-r--r----- 1 system root   163280 1980-01-01 08:00 modem.b06
-r--r----- 1 system root   735936 1980-01-01 08:00 modem.b07
-r--r----- 1 system root  2081092 1980-01-01 08:00 modem.b08
-r--r----- 1 system root 15348816 1980-01-01 08:00 modem.b09
-r--r----- 1 system root   343648 1980-01-01 08:00 modem.b10
-r--r----- 1 system root   488448 1980-01-01 08:00 modem.b11
-r--r----- 1 system root 11529496 1980-01-01 08:00 modem.b12
-r--r----- 1 system root  7234272 1980-01-01 08:00 modem.b13
-r--r----- 1 system root   82368 1980-01-01 08:00 modem.b15
-r--r----- 1 system root   529821 1980-01-01 08:00 modem.b16
-r--r----- 1 system root  9789440 1980-01-01 08:00 modem.b17
-r--r----- 1 system root   77824 1980-01-01 08:00 modem.b18
-r--r----- 1 system root 13230008 1980-01-01 08:00 modem.b19
-r--r----- 1 system root   406932 1980-01-01 08:00 modem.b20
-r--r----- 1 system root    6108 1980-01-01 08:00 modem.mdt
dr-xr-x--- 4 system root    16384 2011-07-06 01:30 modem_pr
-r--r----- 1 system root    472 1980-01-01 00:00 qdsp6m.qdb
-r--r----- 1 system root     27 1980-01-01 00:00 radiover.cfg
-r--r----- 1 system root     27 1980-01-01 00:00 version.cfg
sailfish:/firmware/radio $
```

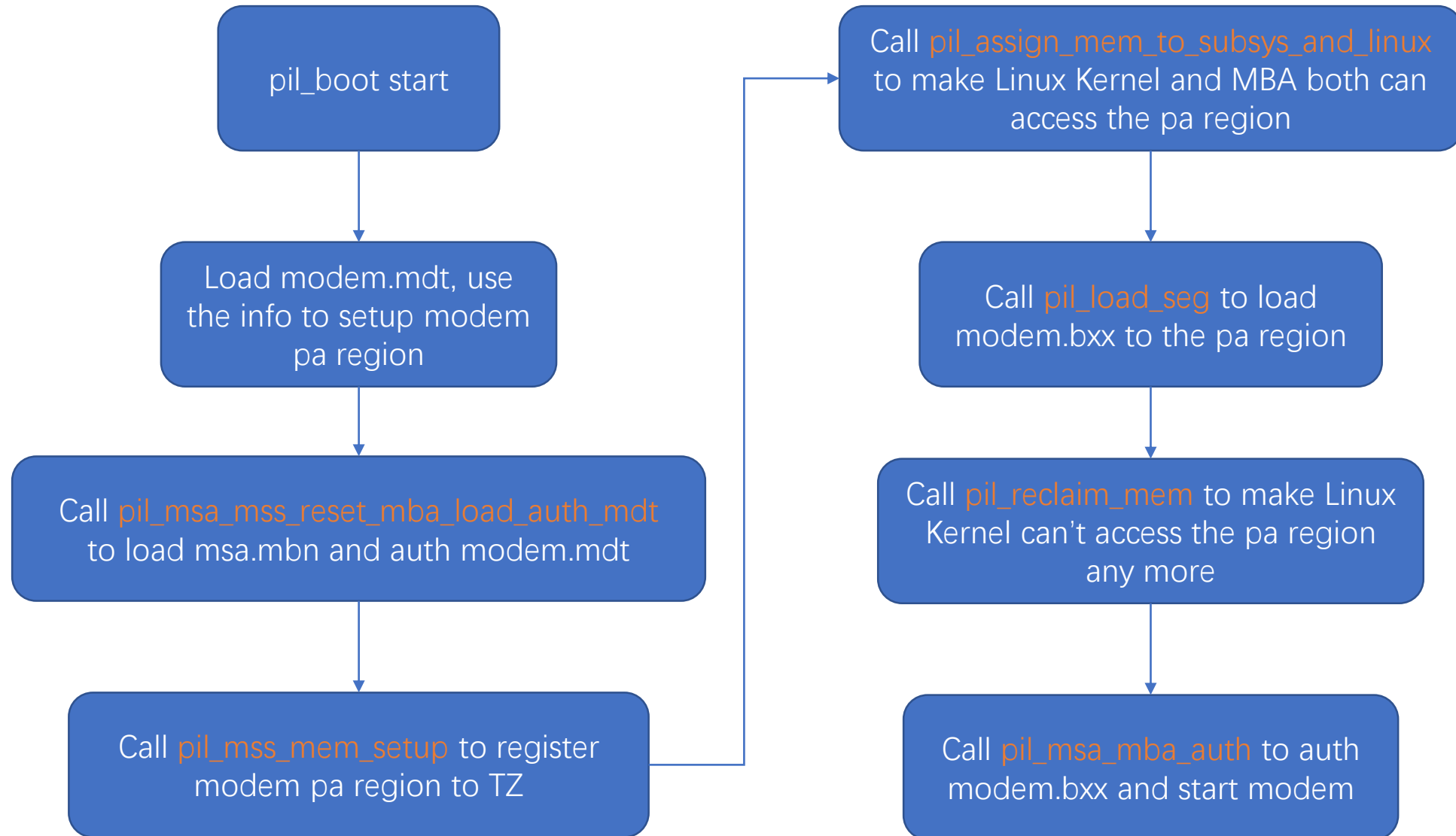
# Modem Secure Boot



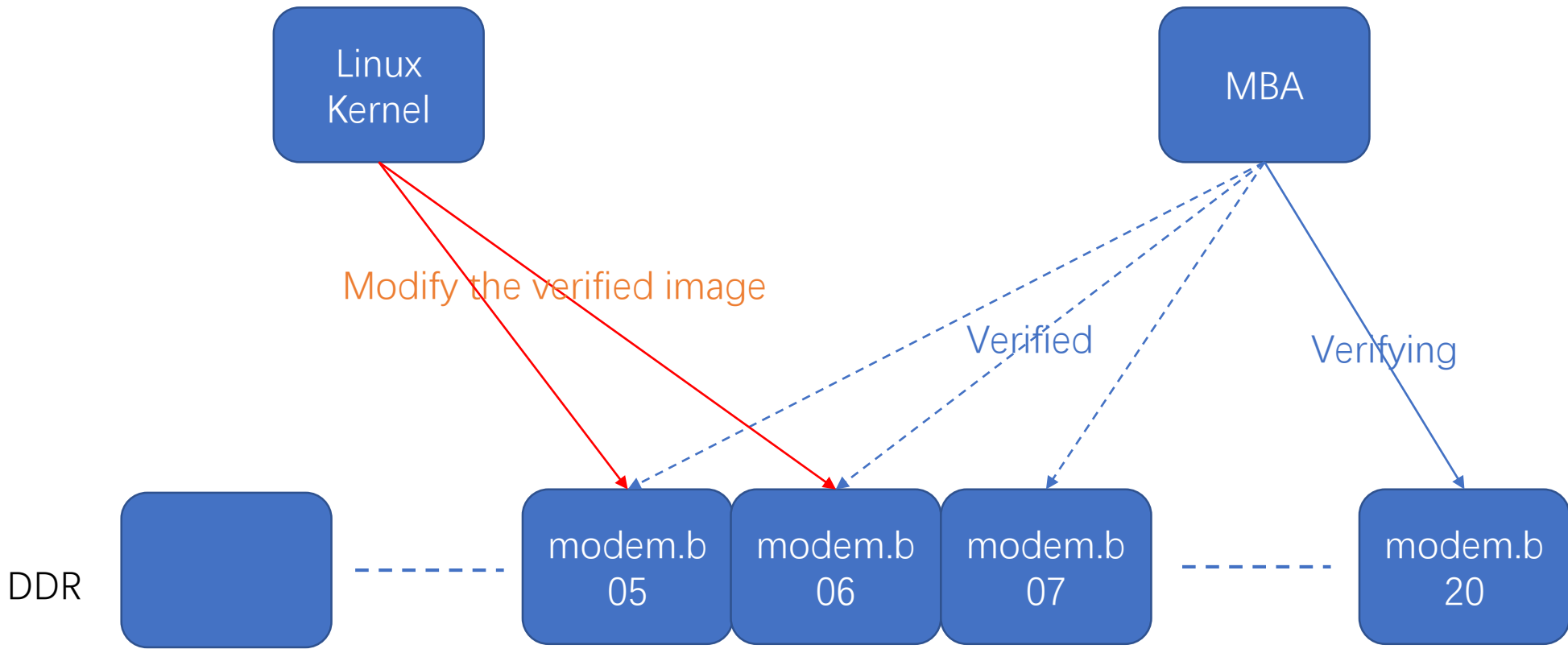
# pil\_boot

- The pil\_boot function in Linux Kernel describes the boot flow of modem.
- Load mba.mbn, modem.mdt and modem.bxx to physical memory.
- Trigger MBA and modem images to be verified and run in Modem Processor.
- Linux Kernel can restart Modem Processor at any time, will hit pil\_boot each time when restart.

# pil\_boot



# TOCTOU Vulnerability

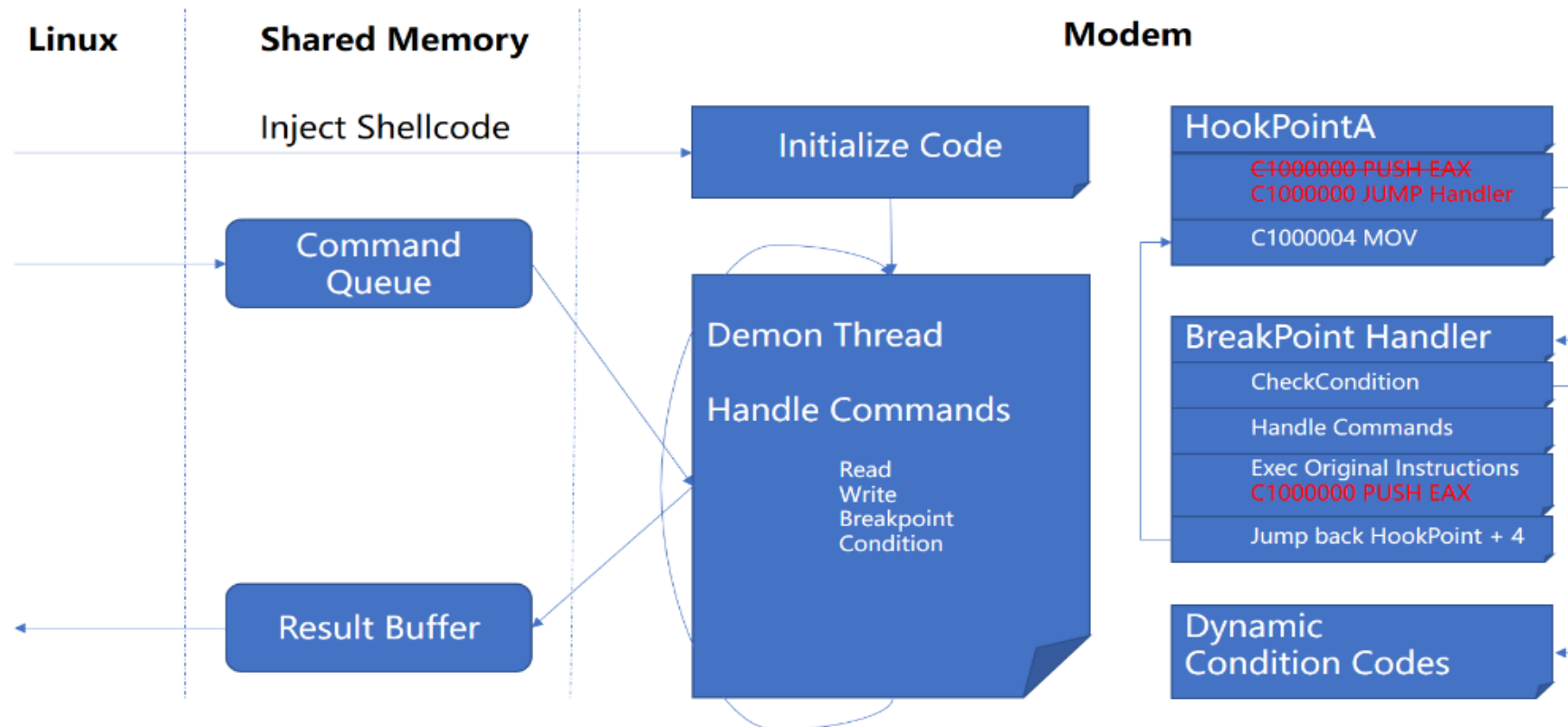


# POC

```
@@ -840,24 +868,33 @@ int pil_boot(struct pil_desc *desc)
|
|         goto err_deinit_image;
|     }
|
-     if (desc->subsys_vmid > 0) {
-         ret = pil_reclaim_mem(desc, priv->region_start,
-             (priv->region_end - priv->region_start),
-             desc->subsys_vmid);
-         if (ret) {
-             pil_err(desc, "Failed to assign %s memory, ret - %d\n",
-                 desc->name, ret);
-             goto err_deinit_image;
-         }
-         hyp_assign = false;
-     }
-
    ret = desc->ops->auth_and_reset(desc);
    if (ret) {
        pil_err(desc, "Failed to bring out of reset\n");
        goto err_auth_and_reset;
    }
    pil_info(desc, "Brought out of reset\n");
+
+     if (modem_dbg_cfg) { // just a switch can be set in userspace to enable our test
+         list_for_each_entry(seg, &desc->priv->segs, list) {
+             pil_modify_seg(desc, seg); // self defined function to modify segments
+         }
+     }
+
+     if (modem_dbg_cfg == 0) {
+         if (desc->subsys_vmid > 0) {
+             ret = pil_reclaim_mem(desc, priv->region_start,
+                 (priv->region_end - priv->region_start),
+                 desc->subsys_vmid);
+             if (ret) {
+                 pil_err(desc, "Failed to assign %s memory, ret - %d\n",
+                     desc->name, ret);
+                 goto err_deinit_image;
+             }
+             hyp_assign = false;
+         }
+     }
+ }
```

# Debug Server Injection

## Implementation

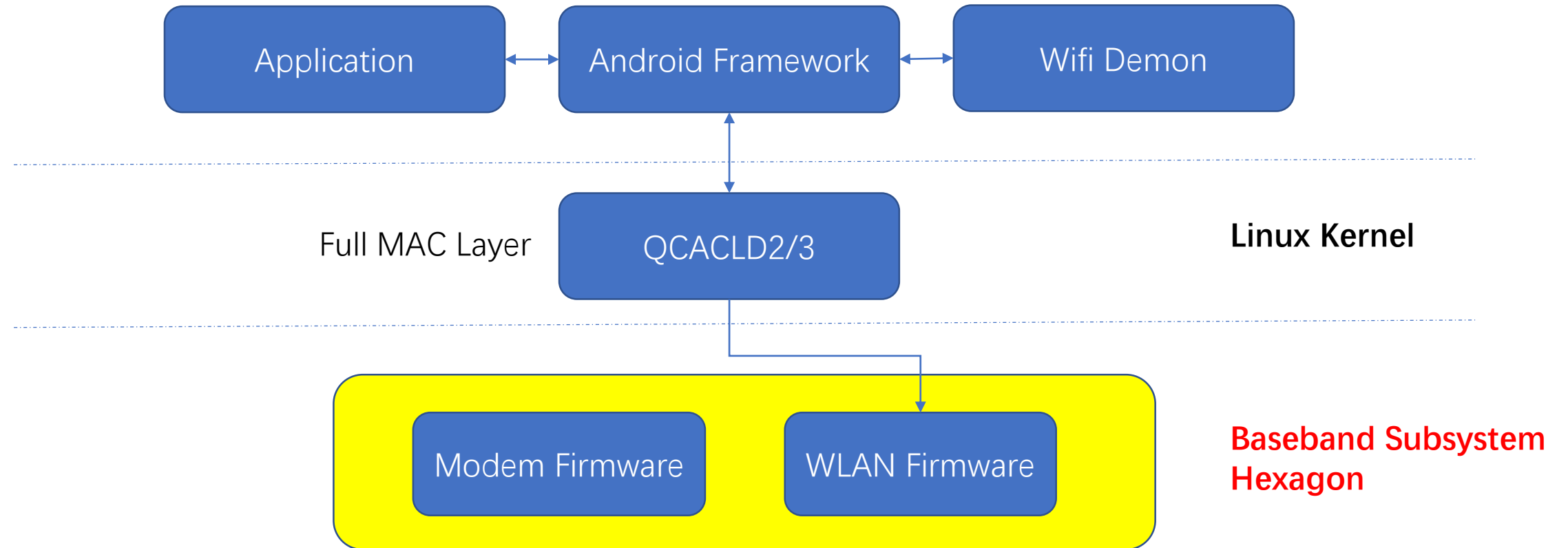


# Agenda

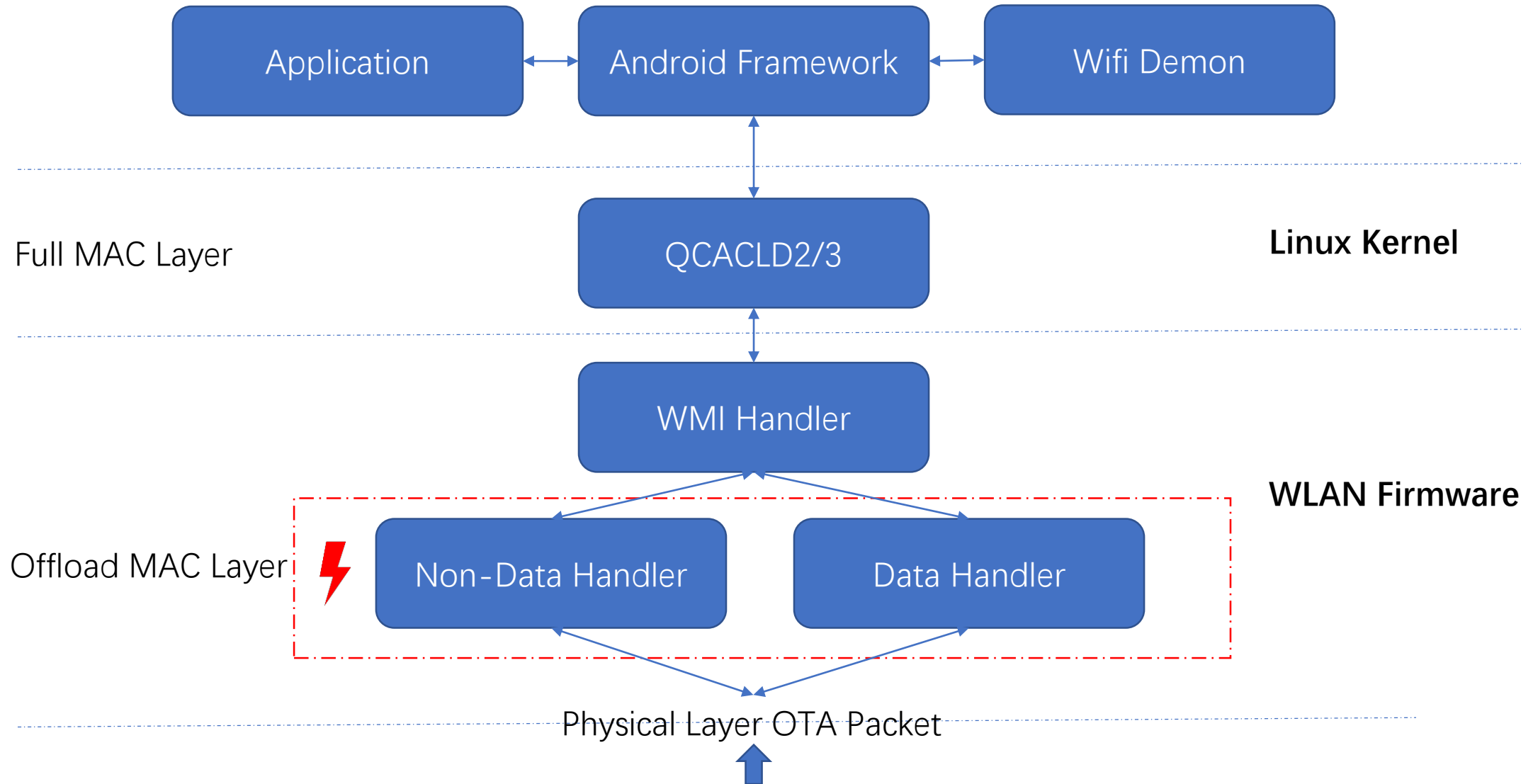
- Introduction and Related Work
- The Debugger
- **Reverse Engineering and Attack Surface**
- Vulnerability and Exploitation
- Escaping into Modem
- Escaping into Kernel
- Stability of Exploitation
- Conclusions



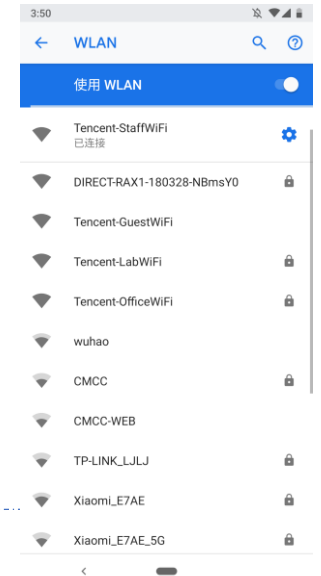
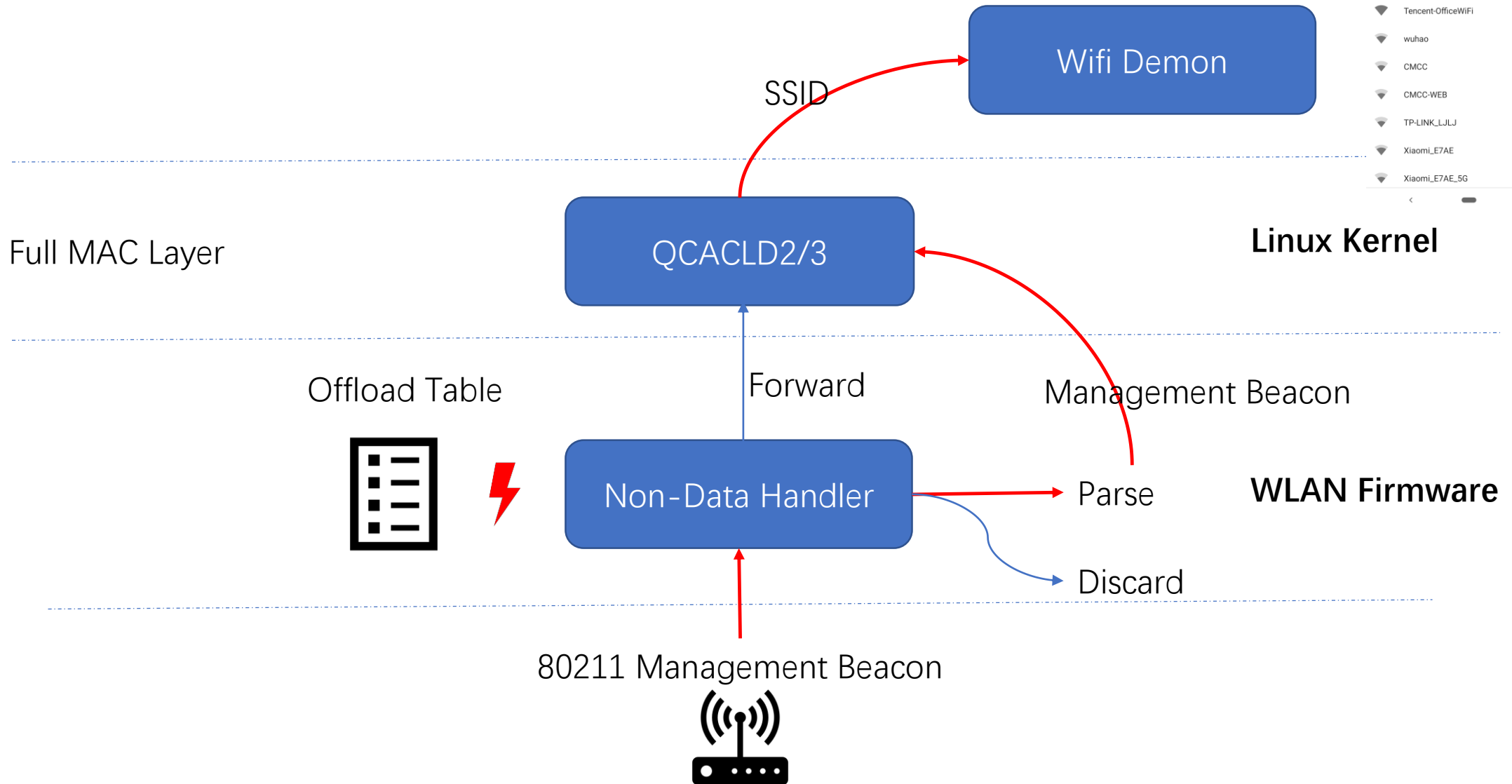
# Qualcomm WLAN



# Qualcomm WLAN Architecture



# Example - Management Beacon



# Firmware

- Modem load WLAN Firmware from  
    /vendor/firmware/wlanmdsp.mbn
- IDA Disassembler
  - <https://github.com/programa-stic/hexagon/tree/master/ida>
  - <https://github.com/gsmk/hexagon>
- Qualcomm SDK
  - <https://developer.qualcomm.com/software/hexagon-dsp-sdk/tools>
- Instruction Reference
  - <https://developer.qualcomm.com/download/hexagon/hexagon-v5x-programmers-reference-manual.pdf?referrer=node/6116>

# Reverse Engineering – Hint From Qualcomm

## String Table

Address	Length	Type	String
LOAD:B0287...	00000023	C	wlan pdev host configured oui init
LOAD:B0287...	0000001D	C	validate oui cmd buff length
LOAD:B0287...	0000001E	C	wlan config vendor oui action
LOAD:B0287...	00000014	C	wlan pdev set param
LOAD:B0287...	0000000F	C	wmi fips event
LOAD:B0287...	00000028	C	wlan vdev get active vdev count per mac
LOAD:B0287...	00000027	C	wlan pdev get hw mode transition event
LOAD:B0287...	00000017	C	dispatch wlan init cmd
LOAD:B0287...	00000014	C	memstats timer func
LOAD:B0287...	00000018	C	dispatch wlan pdev cmds
LOAD:B0287...	0000001A	C	wlan pdev resume send evt

## Import Function

```
f qurt exception raise nonfatal
f msg v2 send 2
f msg v2 send 3
f MMPM Init Ext
f DALSYS Init
f Diag LSM Init
f dog hb task
f dog hb init
f msg v2 send var
f hexagon udivsi3
```

## WMI Handler

drivers/staging/fw-api-fw/wmi\_unified.h

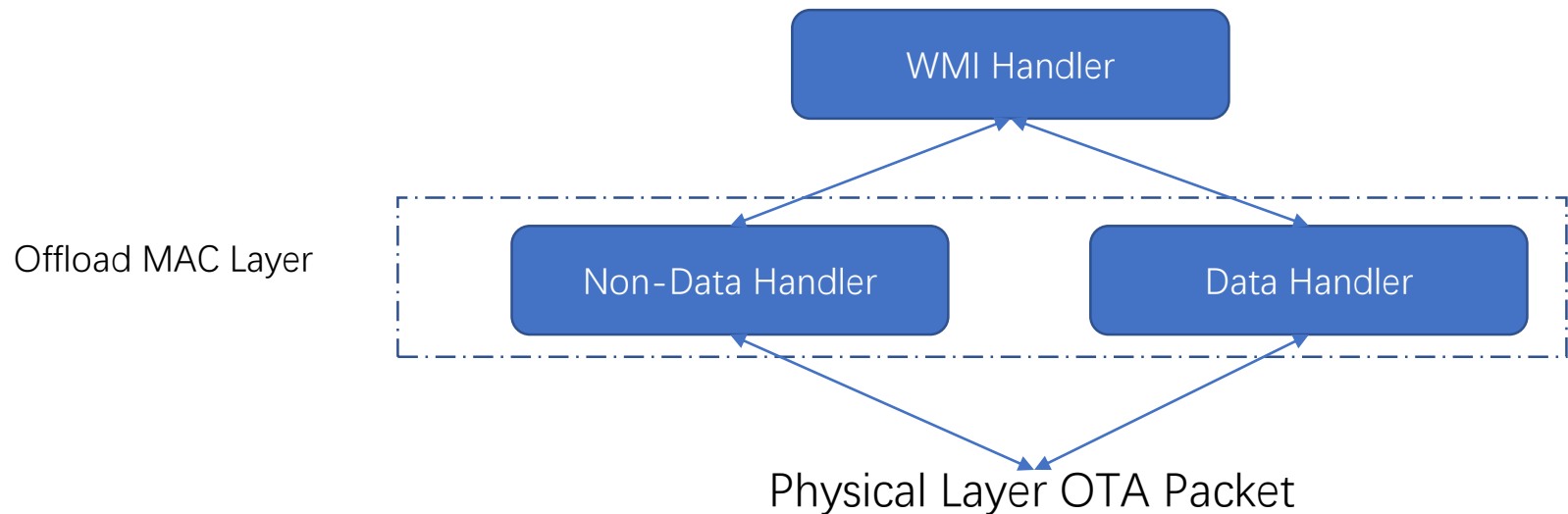
```
/**
 * Command IDs and commange events
 */
typedef enum {
    /** initialize the wlan sub system */
    WMI_INIT_CMDID = 0x1,

    /** Scan specific commands */

    /** start scan request to FW */
    WMI_START_SCAN_CMDID = WMI_CMD_GRP_START_ID(WMI_GRP_SCAN),
    /** stop scan request to FW */
    WMI_STOP_SCAN_CMDID,
    /** full list of channels as defined by the regulatory that will be used by scanner
    WMI_SCAN_CHAN_LIST_CMDID,
    /** overwrite default priority table in scan scheduler */
    WMI_SCAN_SCH_PRIO_TBL_CMDID,
    /** This command to adjust the priority and min.max_rest_time
    * of an on ongoing scan request.
    */
    WMI_SCAN_UPDATE_REQUEST_CMDID,
```

# Reverse Engineering

- Targets To Reverse
  - WMI Handlers
    - Handle WMI commands from Linux Kernel
    - Send back WMI indication to Linux Kernel
  - Offload Handlers
    - Handle OTA Packets



# WMI Handlers

drivers/staging/fw-api-fw/wmi\_unified.h

```
/**
 * Command IDs and commange events
 */
typedef enum {
    /** initialize the wlan sub system */
    WMI_INIT_CMDID = 0x1,

    /** Scan specific commands */

    /** start scan request to FW */
    WMI_START_SCAN_CMDID = WMI_CMD_GRP_START_ID(WMI_GRP_SCAN),
    /** stop scan request to FW */
    WMI_STOP_SCAN_CMDID,
    /** full list of channels as defined by the regulatory that will be used by scanner
    WMI_SCAN_CHAN_LIST_CMDID,
    /** overwrite default priority table in scan scheduler */
    WMI_SCAN_SCH_PRIO_TBL_CMDID,
    /** This command to adjust the priority and min.max_rest_time
    * of an on ongoing scan request.
    */
    WMI_SCAN_UPDATE_REQUEST_CMDID,
```

0x03001

```
LOAD:B0301D00 F4 69 02 B0
LOAD:B0301D04 01 30
LOAD:B0301D06 00 00 00 00
```

```
dd sub_B00269F4
dw 0x3001
dd 0
```

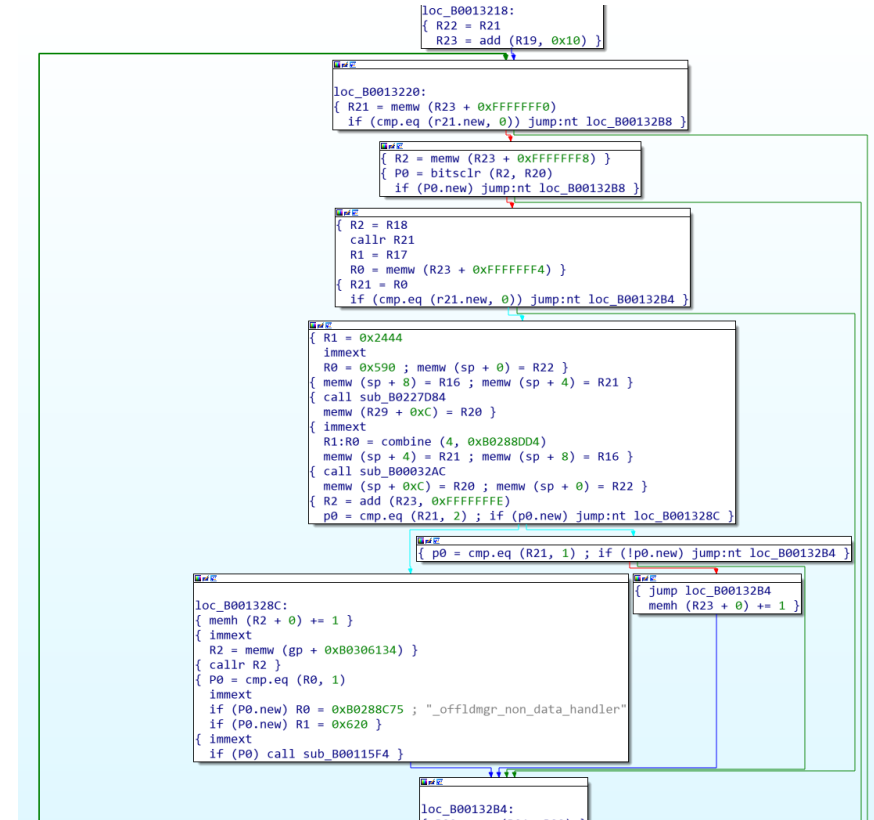
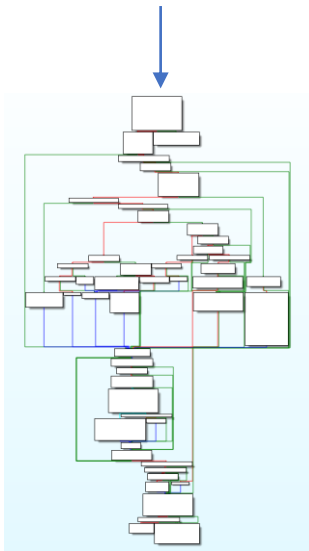
# Offload Handlers

```

LOAD:B0288... 00000020 C offldmgr protocol data handler
LOAD:B0288... 0000001B C offldmgr non data handler
    
```

```

LOAD:B0288C55 5F 6F 66 66+a_offldmgr_prot:db "_offldmgr_protocol_data_handler"
LOAD:B0288C55 6C 64 6D 67+ ; DATA XREF: _offldmgr_protocol_data_handler+B0↑r
LOAD:B0288C55 72 5F 70 72+ db 0
LOAD:B0288C75 5F 6F 66 66+a_offldmgr_non_:db "_offldmgr_non_data_handler"
LOAD:B0288C75 6C 64 6D 67+ ; DATA XREF: _offldmgr_non_data_handler+32C↑r
LOAD:B0288C75 72 5F 6E 6F+ ; _offldmgr_non_data_handler+3C8↑r
LOAD:B0288C75 6E 5F 64 61+ db 0
    
```





# Sample Offload Handler

```
sub_B0004C2C:  
DataPtr = R17  
DataPtr1 = R21  
{ call sub_B02859C4  
  allocframe (0x30) }  
{ R16 = R2 ; R19 = R0 }  
{ R2 = memw (R16 + 0) }  
{ R2 = memw (R2 + 0x10) }  
{ DataPtr = memub (R2 + 0x58)  
  R3 = memub (R2 + 0x59) }  
{ DataPtr |= asl (R3, 8)  
  R4 = memub (R2 + 0x5B)  
  R5 = memub (R2 + 0x5A) }  
{ R5 |= asl (R4, 8) }  
{ DataPtr |= asl (R5, 0x10) }  
{ R3 = memub (DataPtr + 0) }  
{ R18 = and (R3, 0xF0)  
  R1 = and (R3, 0xC) ; management  
  if (!cmp.eq (r1.new, 0)) jump:t loc_B0004DF4 }
```

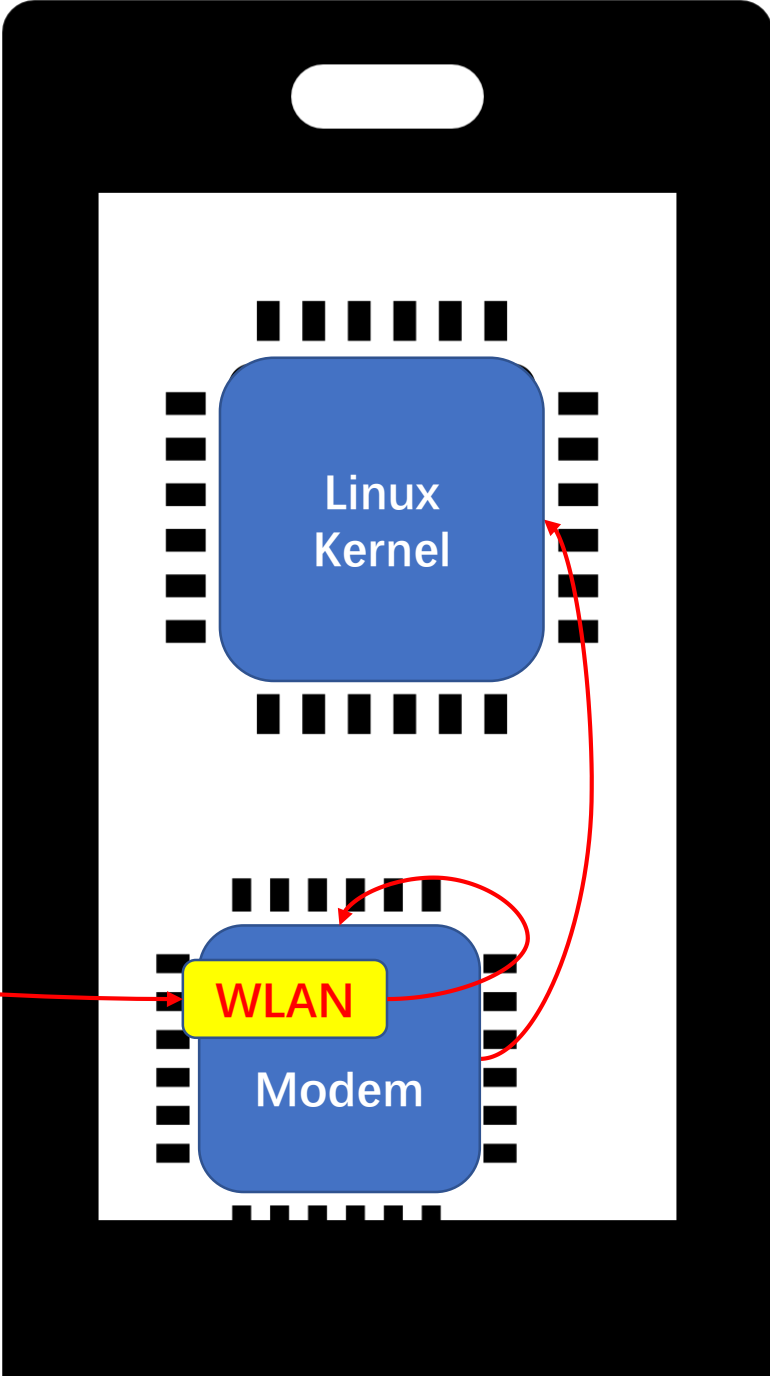
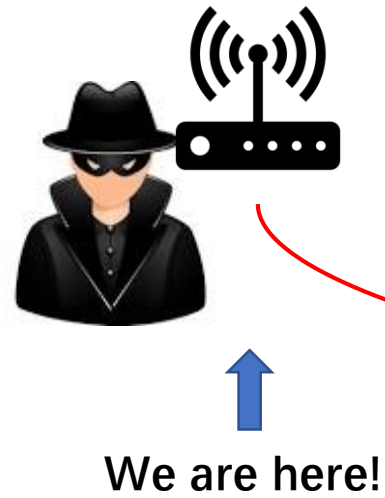
```
{ P0 = cmp.eq (R18, 0x80)  
  DataPtr1 = memub (R2 + 0x5C)  
  R22 = memub (R2 + 0x5D) }  
{ if (P0) jump loc_B0004C84 }
```

OTA Packet Data Pointer  
=  
[0x5B | 0x5A | 0x59 | 0x58]

# Agenda

- Introduction and Related Work
- The Debugger
- Reverse Engineering and Attack Surface
- **Vulnerability and Exploitation**
- Escaping into Modem
- Escaping into Kernel
- Stability of Exploitation
- Conclusions

# The Roadmap



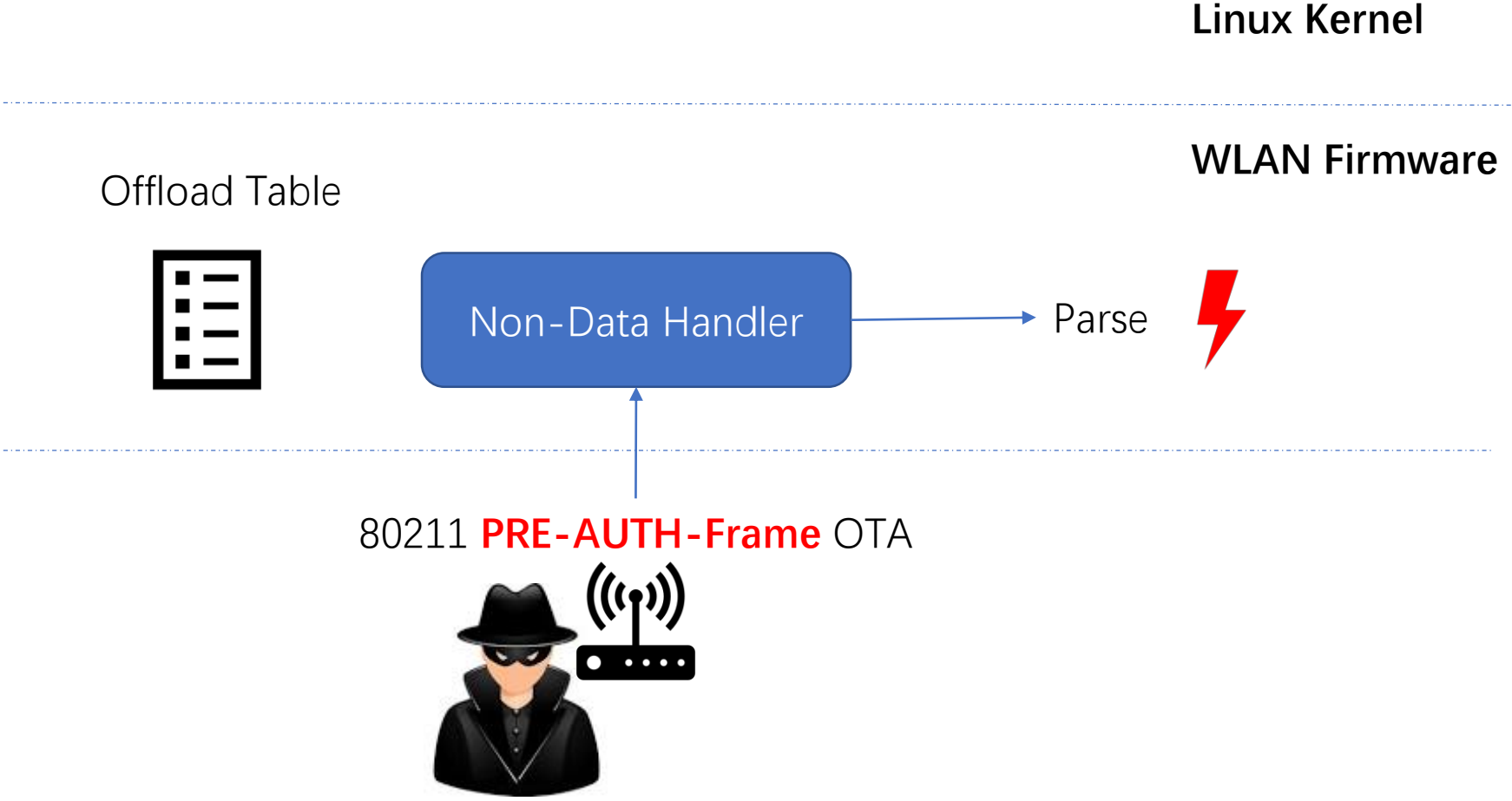
# Mitigation Table (WLAN & Modem)

Mitigation	Status
Heap ASLR	Y
Heap Cookie	Y
Stack Cookie	Y
W^X	Y
FRAMELIMIT*	Y
FRAMEKEY**	Y
Code & Global Data ASLR	N
CFI	N

\***FRAMELIMIT Register** - If SP < FRAMELIMIT throw exception

\*\***FRAMEKEY Register** - Return Address XOR FRAMEKEY. A random integer different for every thread

# The Vulnerability (CVE-2019-10540)



# The Vulnerability (CVE-2019-10540)

Copy items from packet into Global Static Buffer.

Max Item Count = 10

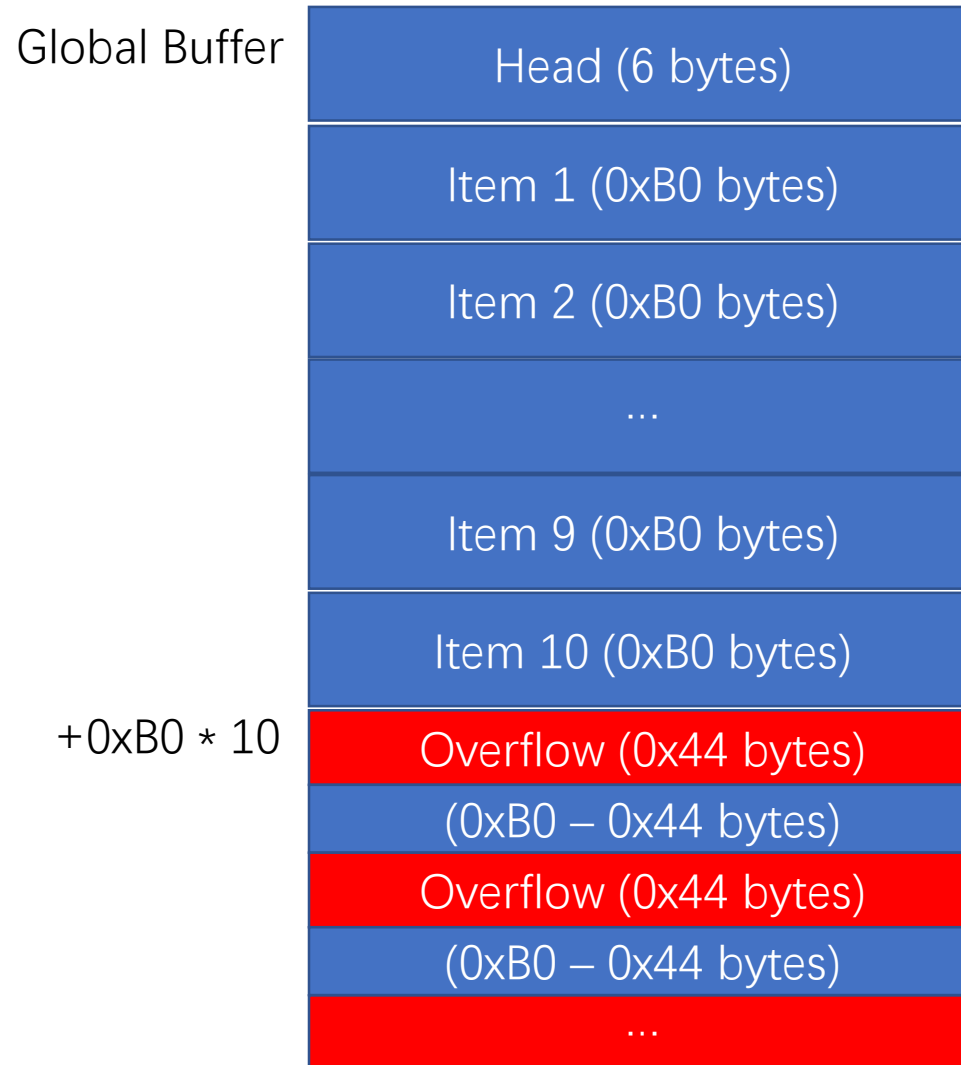
Send 11 items -> Overflow!

```
[GLOBAL] char *GlobalBuffer[10 * 0xB0 + 6];

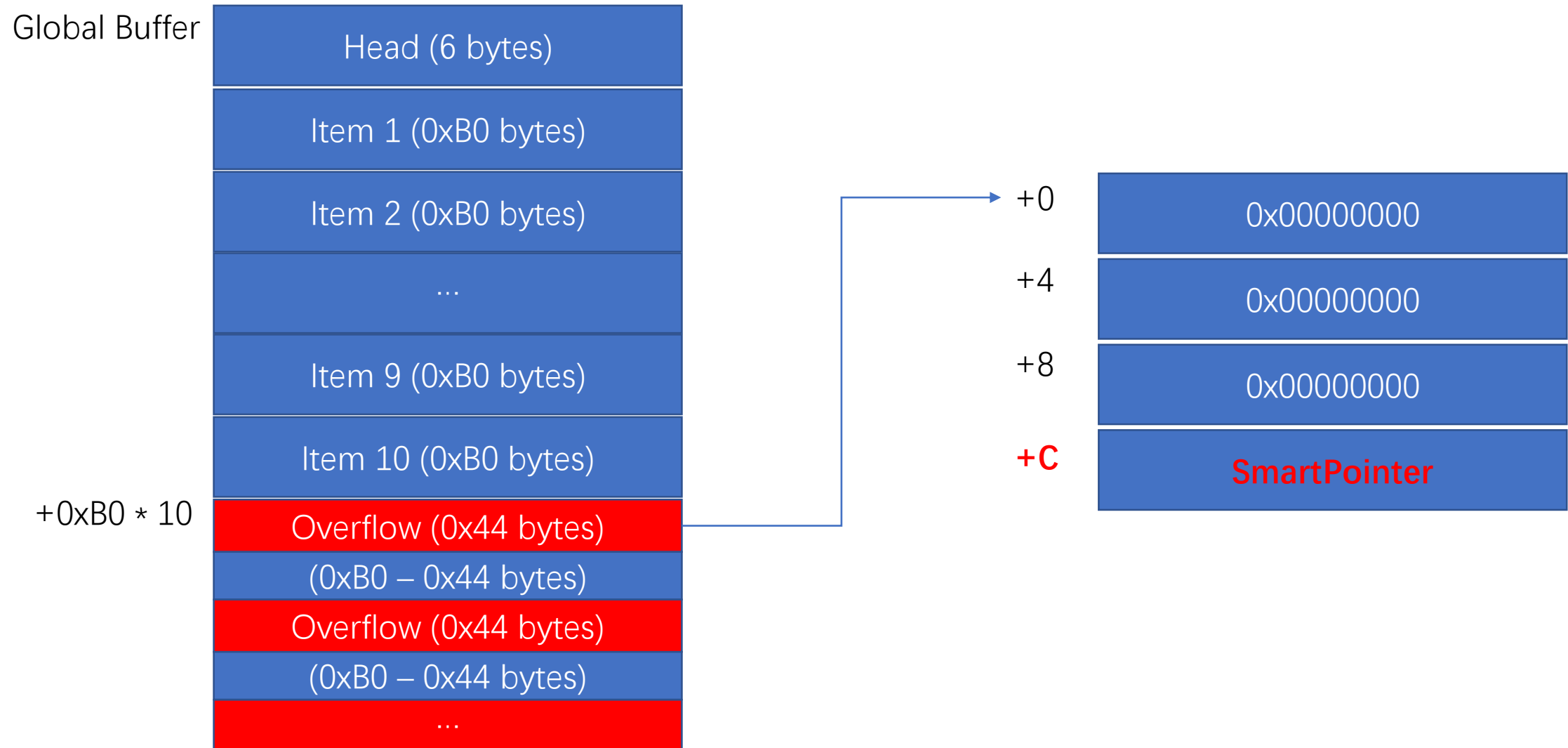
unsigned int itemCount = 0;
for (unsigned int i = 0; i < Length; i += 0x44) {
    memcpy (GlobalBuffer + 6 + itemCount * 0xB0,
            OTA_DataPtr + i,
            0x44);
    itemCount++;
}
```

\* Translated and simplified the code flow

# Data & Address of Overflow



# Smart Pointer Around Overflow Memory





# Usage Of Smart Pointer

```
Char **AddressOfSmartPointer = GlobalBuffer + 6 + 0xB0 * 11 + 0xC;
char *SmartPointer = *AddressOfSmartPointer;
char *MacAddress = OTA_DataPtr + 0x10;
char *BYTE_C = OTA_DataPtr + 0x10 + 0x20;
char *BYTE_D = OTA_DataPtr + 0x10 + 0x21;
char *BYTE_14 = OTA_DataPtr + 0x10 + 0x22;
if (TestBit(SmartPointer, 0) == 1) {
    if (memcmp(SmartPointer + 6, MacAddress, 6) == 0) {
        *(SmartPointer + 0xC) = *BYTE_C;
        *(SmartPointer + 0xD) = *BYTE_D;
        *(SmartPointer + 0x14) = *BYTE_14;
    }
}
```

\* Translated and simplified the code flow

# Usage Of Smart Pointer

```
Char **AddressOfSmartPointer = GlobalBuffer + 6 + 0xB0 * 11 + 0xC;
char *SmartPointer = *AddressOfSmartPointer; // ← Overwrite with vulnerability
char *MacAddress = OTA_DataPtr + 0x10;
char *BYTE_C = OTA_DataPtr + 0x10 + 0x20;
char *BYTE_D = OTA_DataPtr + 0x10 + 0x21;
char *BYTE_14 = OTA_DataPtr + 0x10 + 0x22;
if (TestBit(SmartPointer, 0) == 1) { // ← The only constraint, Bit0 == 1
    if (memcmp(SmartPointer + 6, MacAddress, 6) == 0) { // ← From OTA Data, could be bypass
        *(SmartPointer + 0xC) = *BYTE_C; // ← Overwrite 0xC
        *(SmartPointer + 0xD) = *BYTE_D; // ← Overwrite 0xD
        *(SmartPointer + 0x14) = *BYTE_14;
    }
}
```

\* Translated and simplified the code flow

# Global Write With Constraint

Step 1 Overwrite SmartPointer



SmartPointer

0XXXXXXXXX



0XXXXXXXXX

00 00 00 01

+4

00 00 00 00

+8

00 00 00 00

+C

12 34 56 78

Step 2 Global Write (Using SmartPointer)



+0

00 00 00 01

Bit 0

+4

MA CA 00 00

MAC

+8

AD DR ES SS

+C

12 34 ?? ??

Write



# Global Write With Constraint

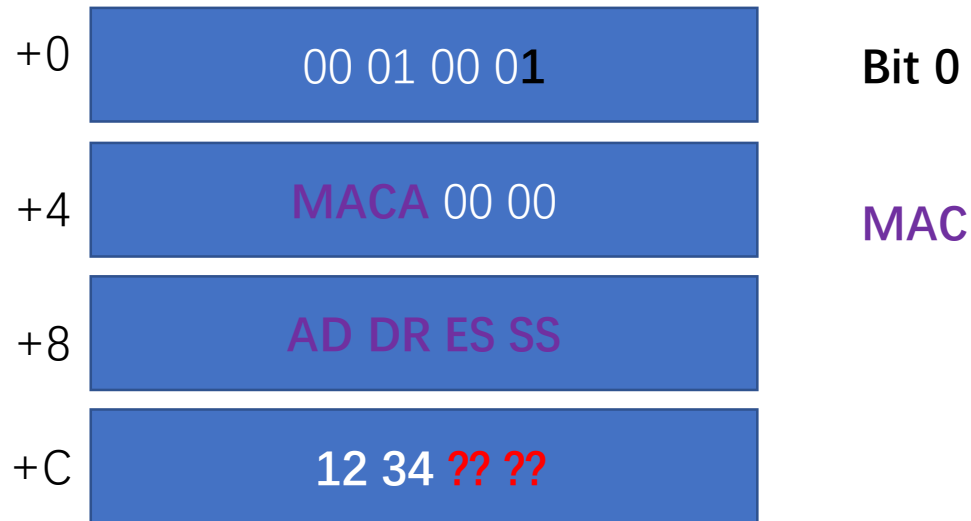
## How to write 4 bytes?

Step 1 Overwrite SmartPointer

SmartPointer 0XXXXXXXXX



Step 2 Global Write (Using SmartPointer)

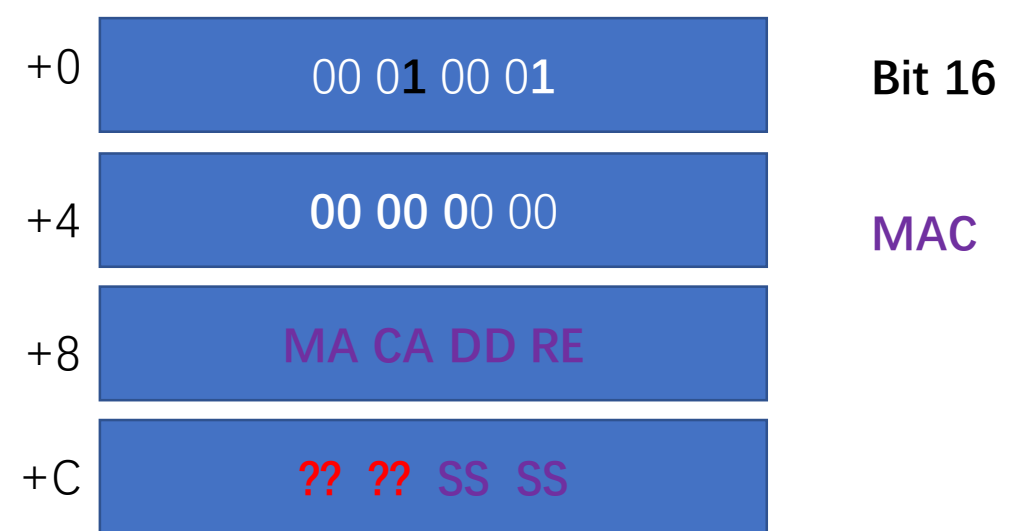


Step 3 Overflow SmartPointer

SmartPointer 0XXXXXXXXX+2

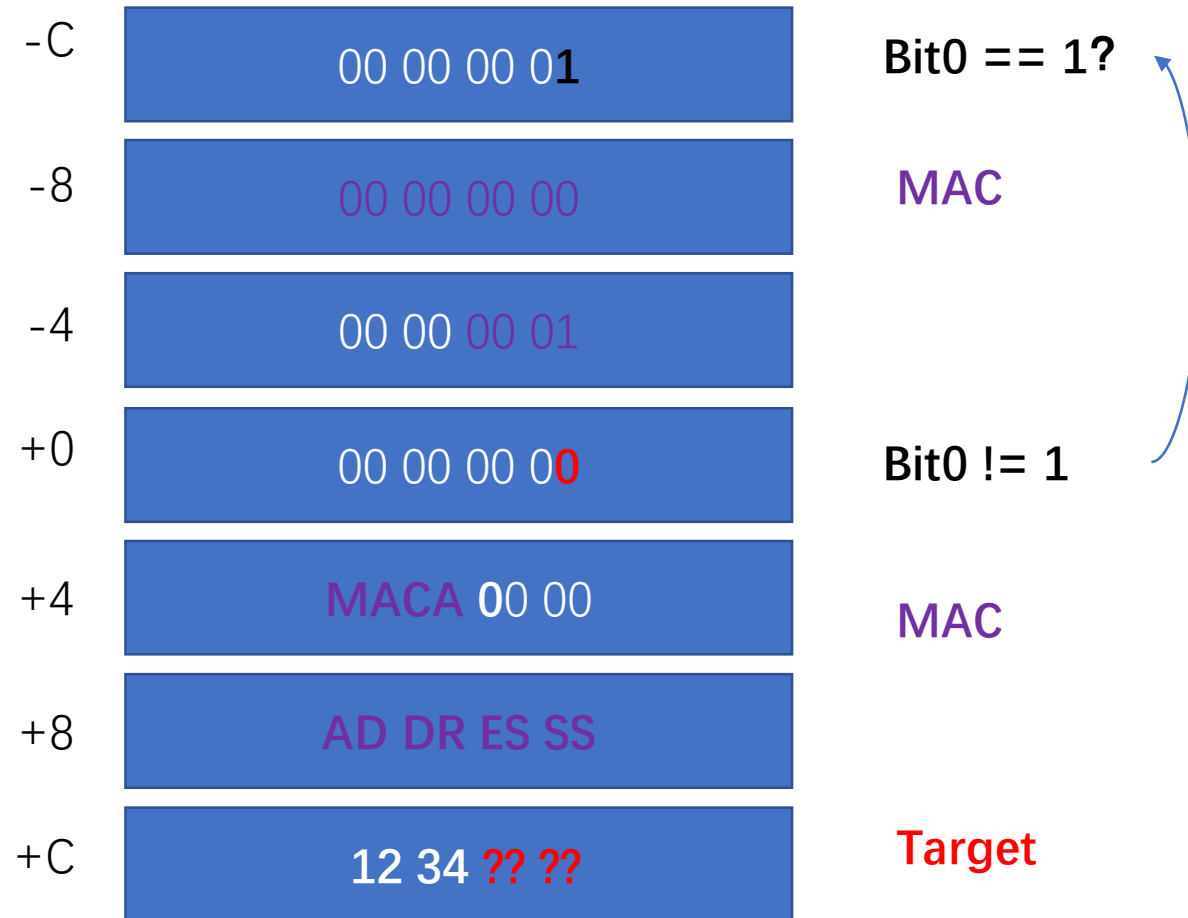


Step 4 Global Write (Using SmartPointer)



# Global Write With Constraint

## The Bit0 != 1?



# Control PC & R0

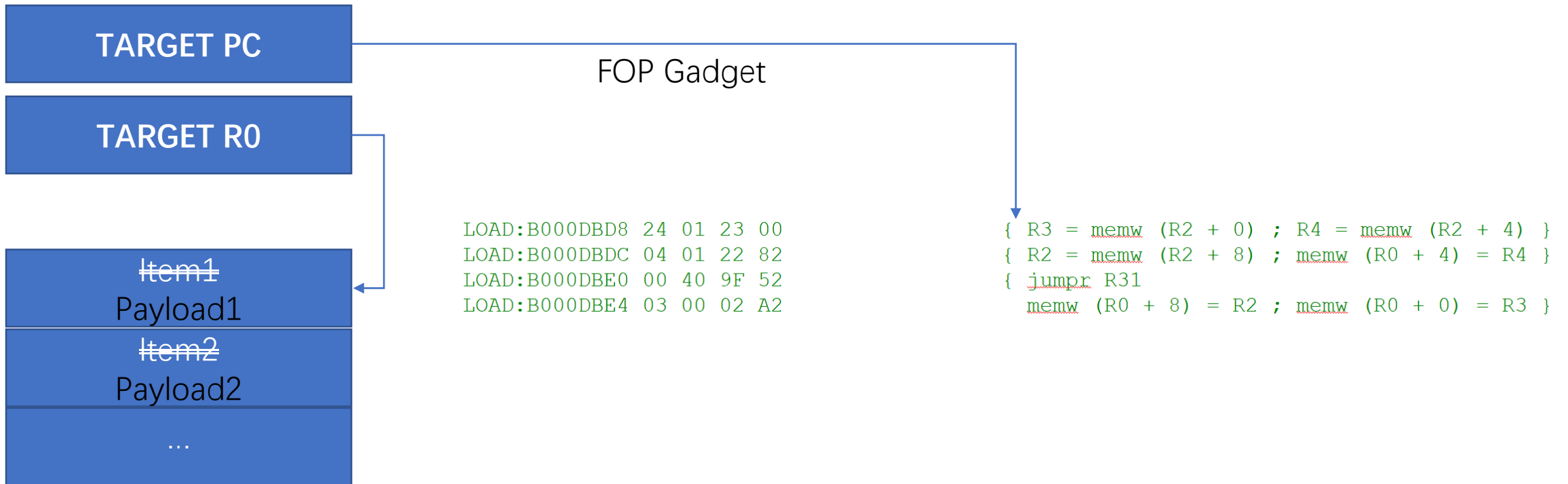
Address	Value
00	0x00010000
+04	0x00010001
+08	0x00000000
+0C	0x00000001
+10	0x00000000
+14	0x00000000
+18	0x00000000
+1C	0x00000000
+20	0x00000000
+24	0x12345678(PC)
+28	0x87654321(R0)



Address	Value
+00	0x00010000
+04	0x00010001
+08	0x00000000
+0C	0x000 <b>1</b> 0001
+10	0x000 <b>1</b> 000 <b>1</b>
+14	0x00000000
+18	0x000 <b>1</b> 000 <b>1</b>
+1C	0x000 <b>1</b> 000 <b>1</b>
+20	0x00000000
+24	<b>TARGET PC</b>
+28	<b>TARGET R0</b>

SmartPointer

# Transform To Arbitrary Write



# Run Useful FOP Gadget

**Step 1 Arbitrary Write** Overwrite function pointer →

**Function Pointer(PC)**

**Step 2 Arbitrary Write** Overwrite data pointer →

**Data Pointer (R0)**

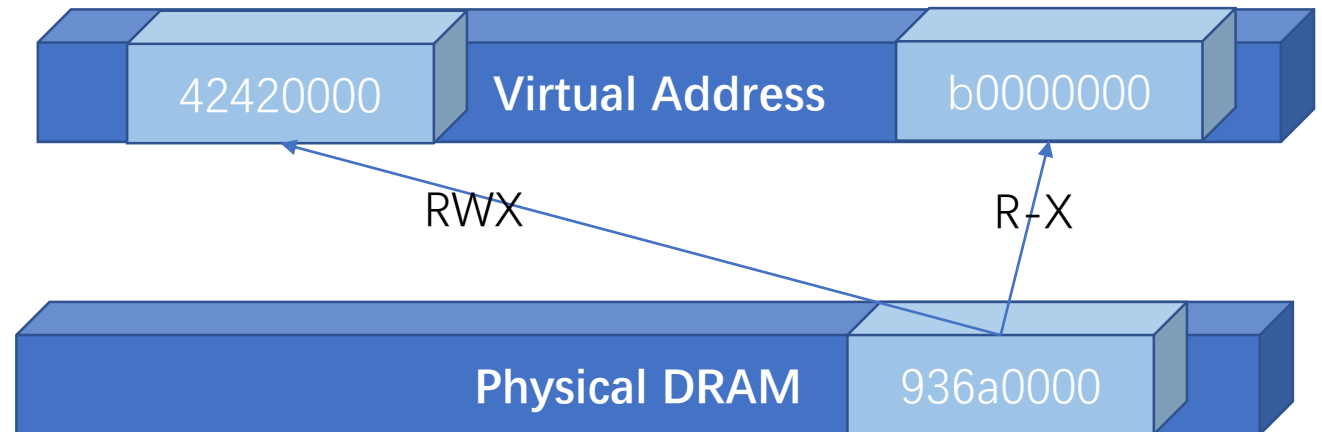
**Step 3 Send payload packet and trigger the PC** →



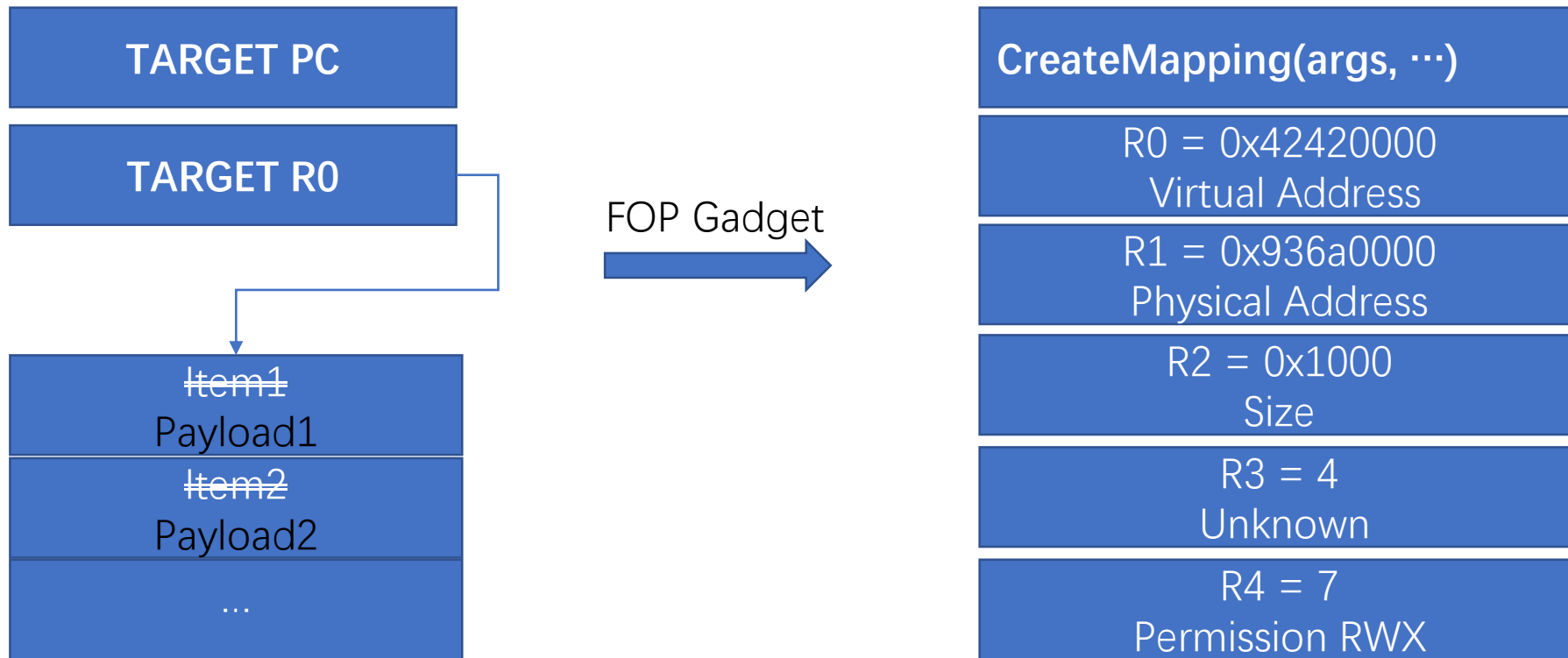


# Memory Mapping RWX

<b>CreateMapping(args, ...)</b>
R0 = 0x42420000 Virtual Address
R1 = 0x936a0000 Physical Address
R2 = 0x1000 Size
R3 = 4 Unknown
R4 = 7 Permission RWX



# Memory Mapping RWX



# Copy Shellcode to 0x42420000

**Step 1 Arbitrary Write** Overwrite function pointer →

memcpy(PC)

← Step 3 Trigger

**Step 2 Arbitrary Write** Overwrite data pointer →

0x42420000(R0)

OTA Packet(R1)

Packet Len(R2)

0x42420000

Shellcode

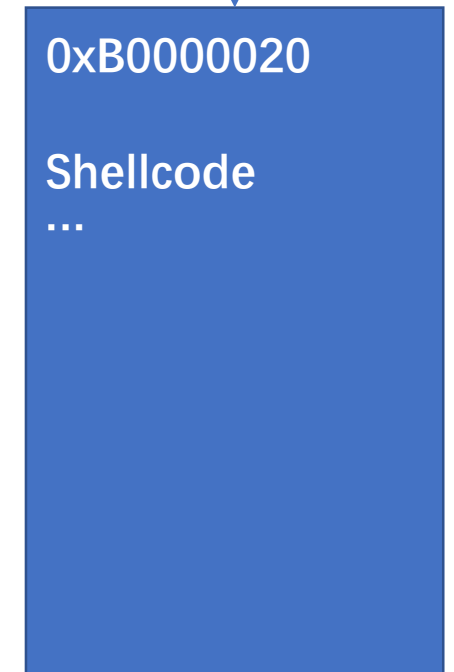
...

# Trigger Shellcode

**Step 1 Arbitrary Write** Overwrite function pointer →



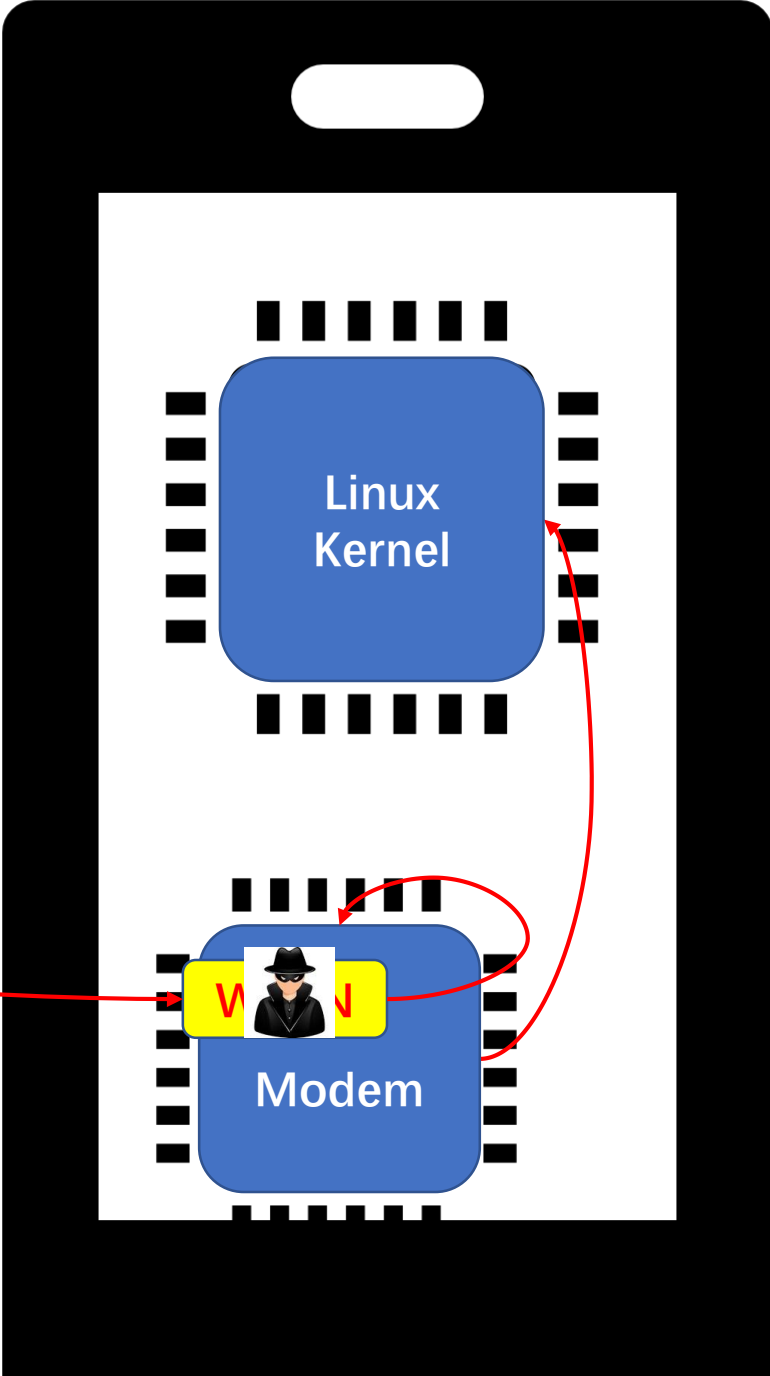
← **Step 2 Trigger**



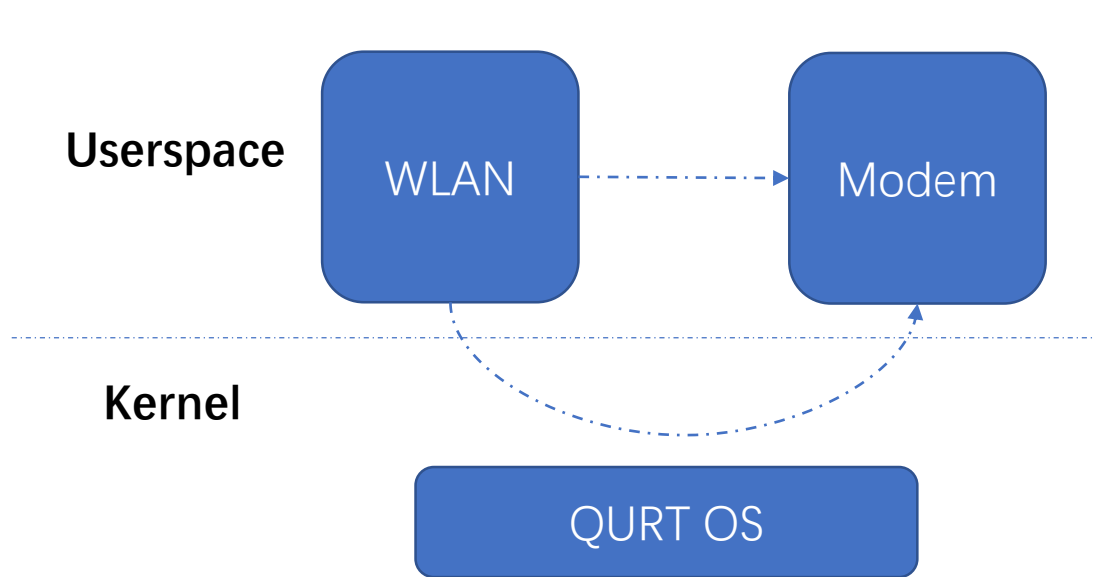
# Agenda

- Introduction and Related Work
- The Debugger
- Reverse Engineering and Attack Surface
- Vulnerability and Exploitation
- **Escaping into Modem**
- Escaping into Kernel
- Stability of Exploitation
- Conclusions

# The Roadmap



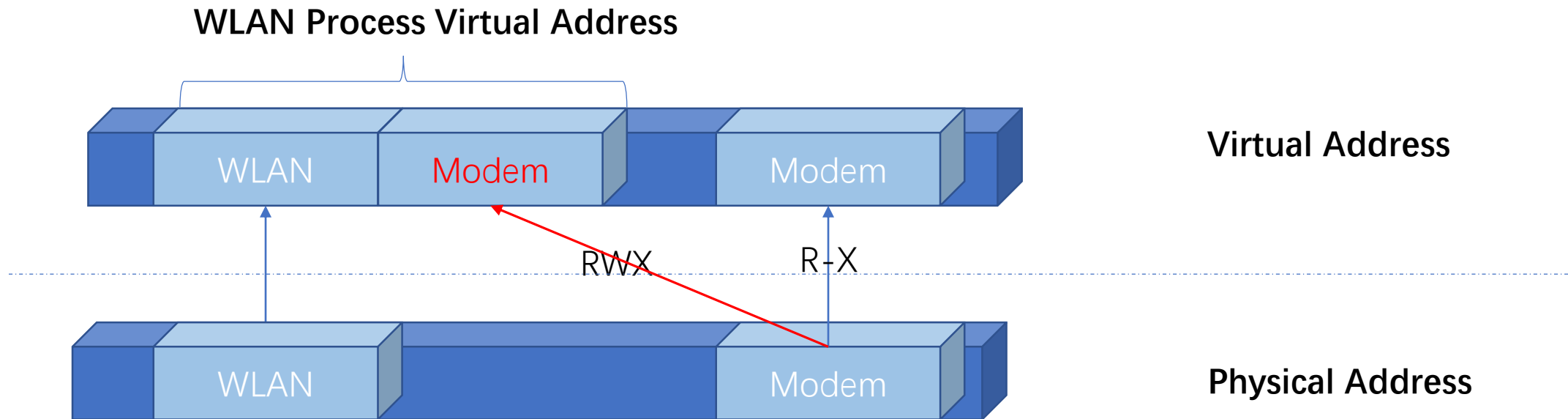
# From WLAN to Modem



Actions From WLAN	Eligible?
TLB Set*	N
Write Modem Data	N
Call Modem Complex Function**	N
Call Modem Simple Code Snippet***	Y
<b>Map Modem Memory</b>	<b>Y</b>

- \* TLB is a Hexagon Instruction to modify the Memory Page Attribute
- \*\* Complex Function uses the resource of Modem, or calls System Call
- \*\*\* Simple Code Snippet mean code has only register operation

# Map Modem Memory into WLAN

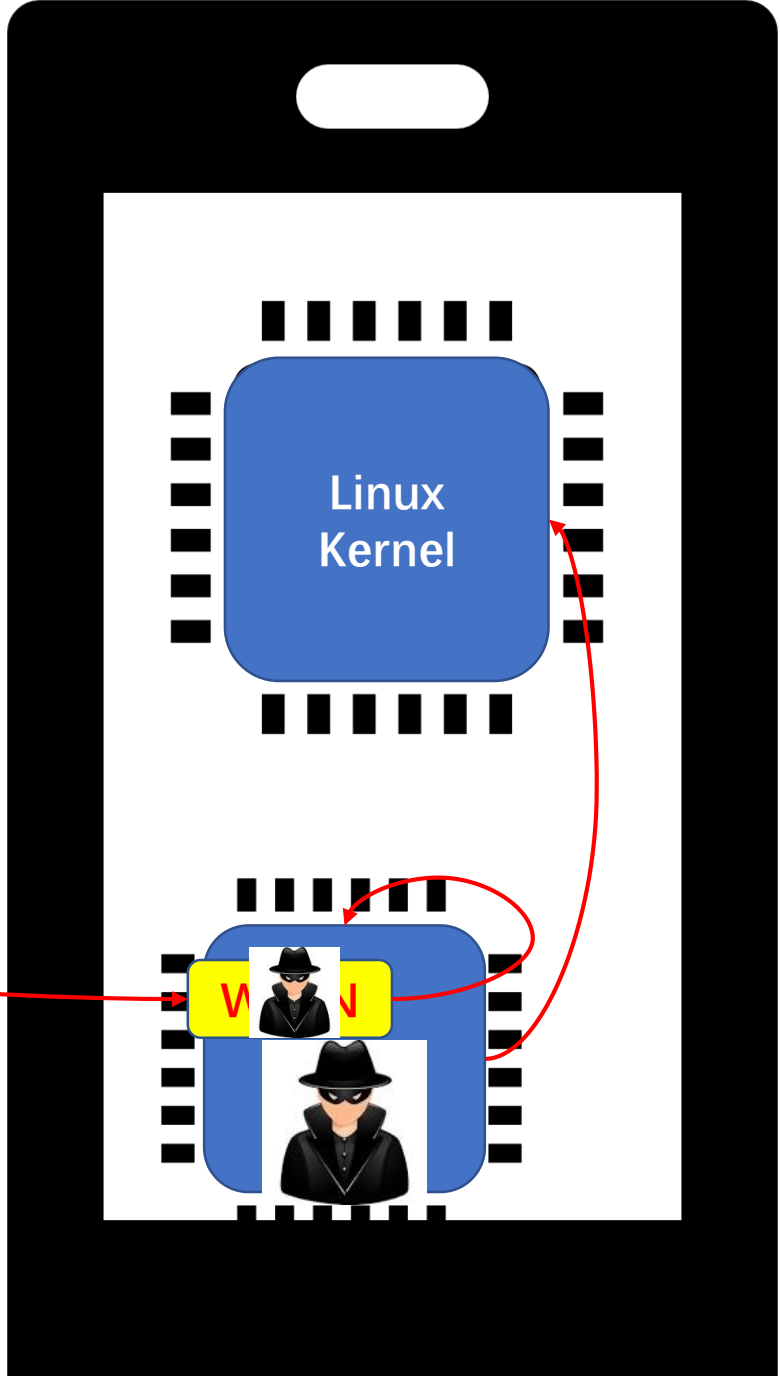




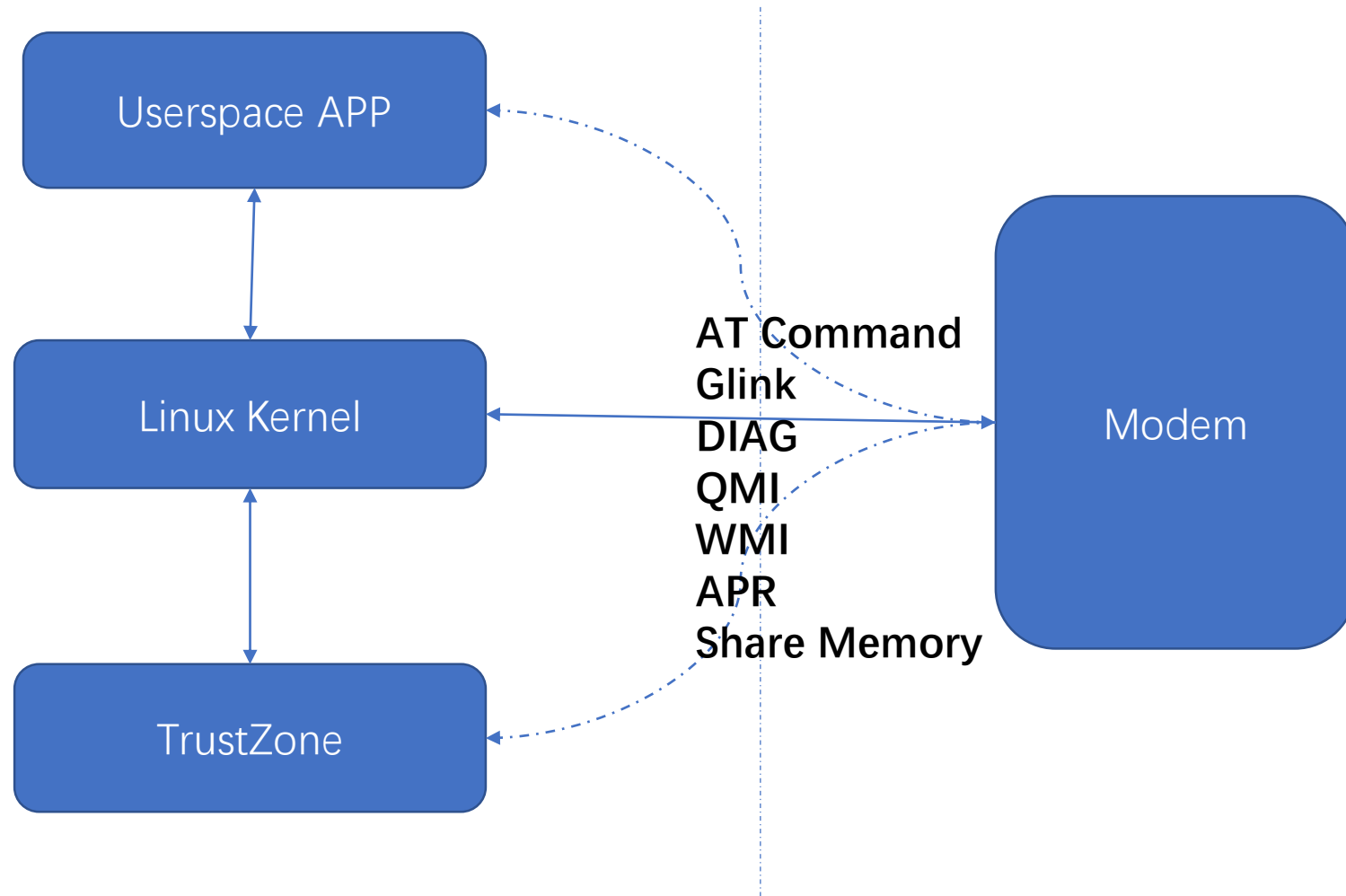
# Agenda

- Introduction and Related Work
- The Debugger
- Reverse Engineering and Attack Surface
- Vulnerability and Exploitation
- Escaping into Modem
- **Escaping into Kernel**
- Stability of Exploitation
- Conclusions

# The Roadmap



# The Attack Surfaces



- We've found

An arbitrary memory read/write vulnerability  
Could bypass all the mitigations of Linux Kernel  
From Modem into Linux Kernel

- In these attack surfaces
- But we are unable to disclose the detail now

# Agenda

- Introduction and Related Work
- The Debugger
- Reverse Engineering and Attack Surface
- Vulnerability and Exploitation
- Escaping into Modem
- Escaping into Kernel
- **Stability of Exploitation**
- Conclusions

# Deliver the Payload Over-The-Air

Pixel 2XL



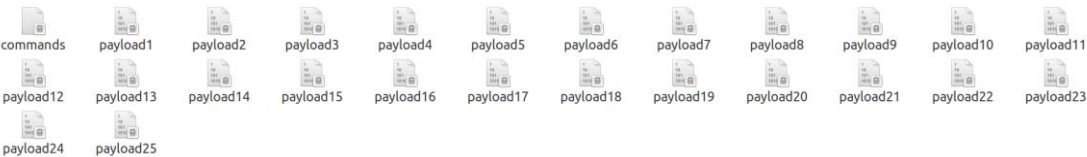
```
      aSPY//YASa
    apyyyyCY/////////YCa
  sY////////YSpcs  scpCY//Pp
ayp ayyyyyySCP//Pp      syY//C
AYAsAYYYYYYYY//Ps      cY//S
      pCCCCY//p      cSSps y//Y
    SPPPP//a      pP//AC//Y
      A//A      cyP///C
      p//Ac      sC///a
      P///YCpc      A//A
    sccccp///pSP///p      p//Y
  sY/////////y caa      S//P
cayCyayP//Ya      pY/Ya
sY/PsY///YCc      aC//Yp
  sc  sccaCY//PCyapaPY//YSs
      spCPY/////////YPSps
      ccaacs
```

```
Welcome to Scapy
Version 2.4.2.dev172

https://github.com/secdev/scapy


Have fun!

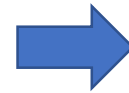
To craft a packet, you have to be a
packet, and learn how to swim in
the wires and in the waves.
-- Jean-Claude Van Damme
```



Packet Losing Rate **90%+!**

# Deliver the Payloads Using Pixel2

Pixel 2 

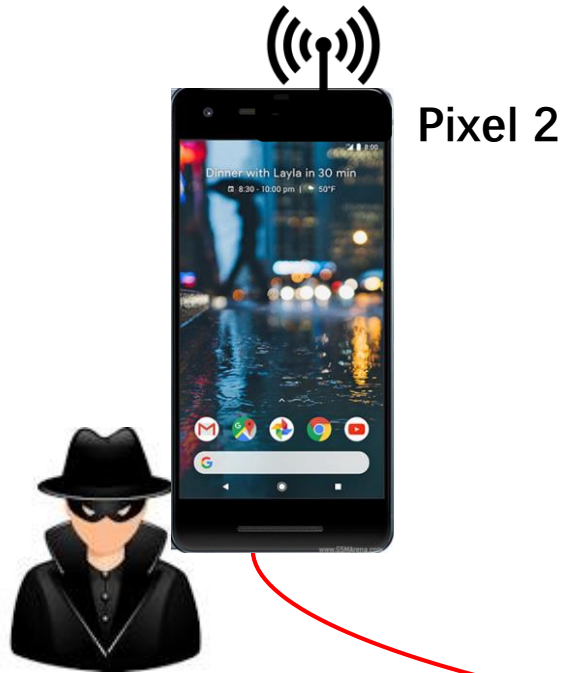


Pixel 2XL

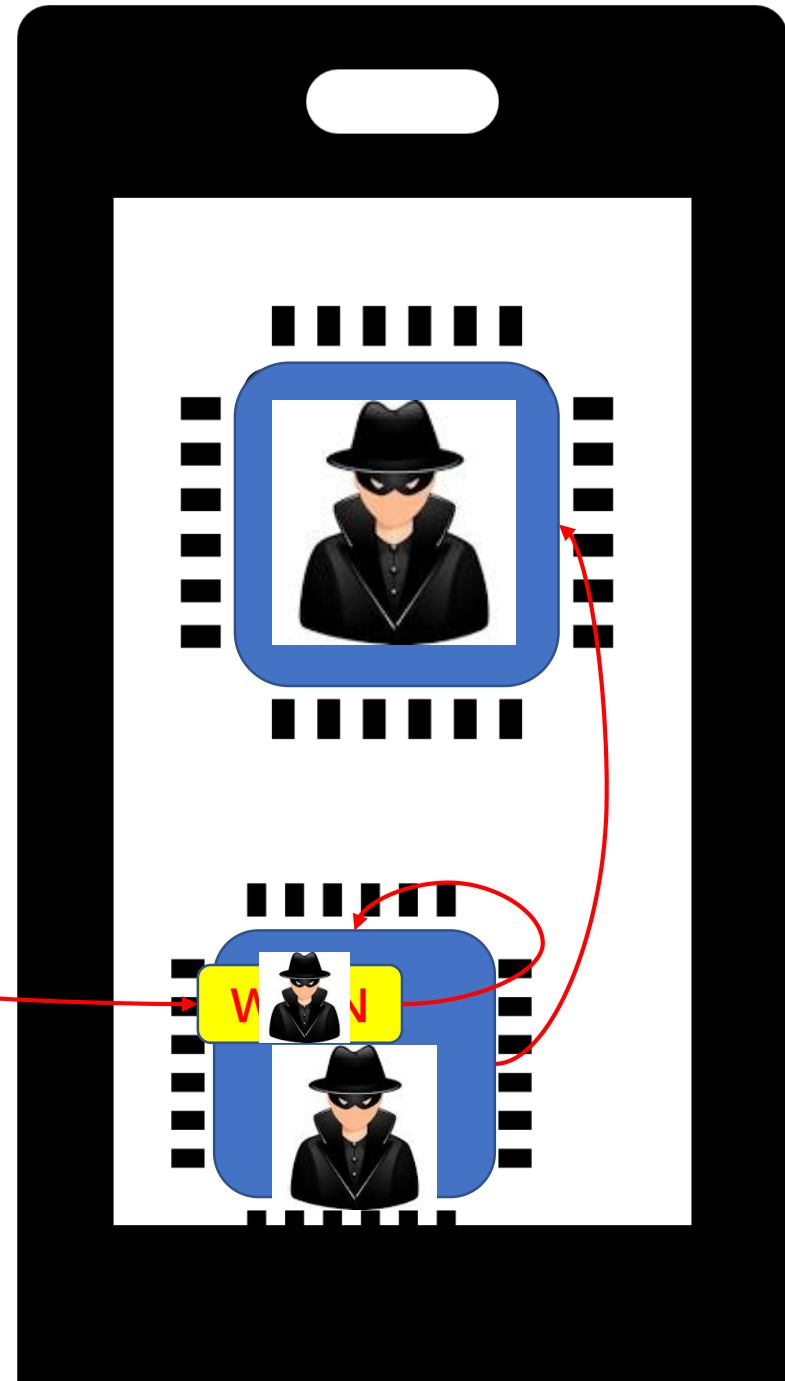


- commands
- payload1
- payload2
- payload3
- payload4
- payload5
- payload6
- payload7
- payload8
- payload9
- payload10
- payload11
- payload12
- payload13
- payload14
- payload15
- payload16
- payload17
- payload18
- payload19
- payload20
- payload21
- payload22
- payload23
- payload24
- payload25

# The Roadmap



- commands
- payload1
- payload2
- payload3
- payload4
- payload5
- payload6
- payload7
- payload8
- payload9
- payload10
- payload11
- payload12
- payload13
- payload14
- payload15
- payload16
- payload17
- payload18
- payload19
- payload20
- payload21
- payload22
- payload23
- payload24
- payload25



Pixel 2XL



Demo

# Future Works

- There are still lots of mystery in the WLAN.
  - We were only reversed a small part of the code
  - Lots of functions are unknown
- How to fuzz the WLAN Firmware?
  - Reverse engineering is quite...
  - How to fuzz closed source target and Hexagon architecture effectively?
- Translate Hexagon Instruction to C?
  - IDA/Ghidra F5 plugin?

# Timeline

- 2019-2-14 Find the Modem debug vulnerability on MSM8998
- 2019-3-24 Find the WLAN issue and report to Google
- 2019-3-28 Google forwards the issue to Qualcomm
- 2019-4-24 Google confirms the WLAN issue as Critical
- 2019-5-08 Report the WLAN into Linux Kernel issue to Google
- 2019-5-24 Google confirms the WLAN into Linux Kernel issue
- 2019-5-28 Submit the full exploit chain to Google
- 2019-6-04 Google reply unable to reproduce the full exploit chain
- 2019-6-17 Improve the stability and submit to Google
- 2019-7-19 CVE Assigned by Google
- 2019-7-20 Qualcomm confirms issues will be fixed before October
- 2019-8-0? Google release the fix for Google Pixel2/Pixel3

THANK YOU

**BLADE** Tencent  
Blade

<https://blade.tencent.com>