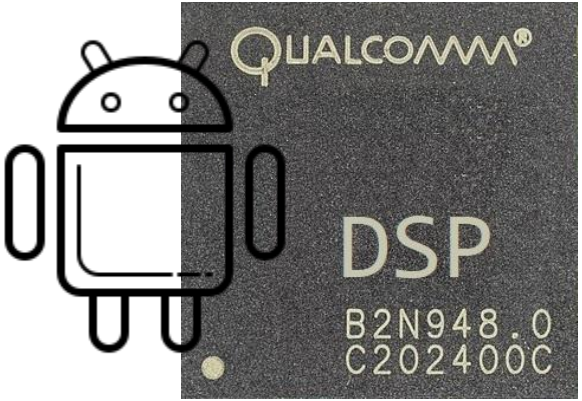
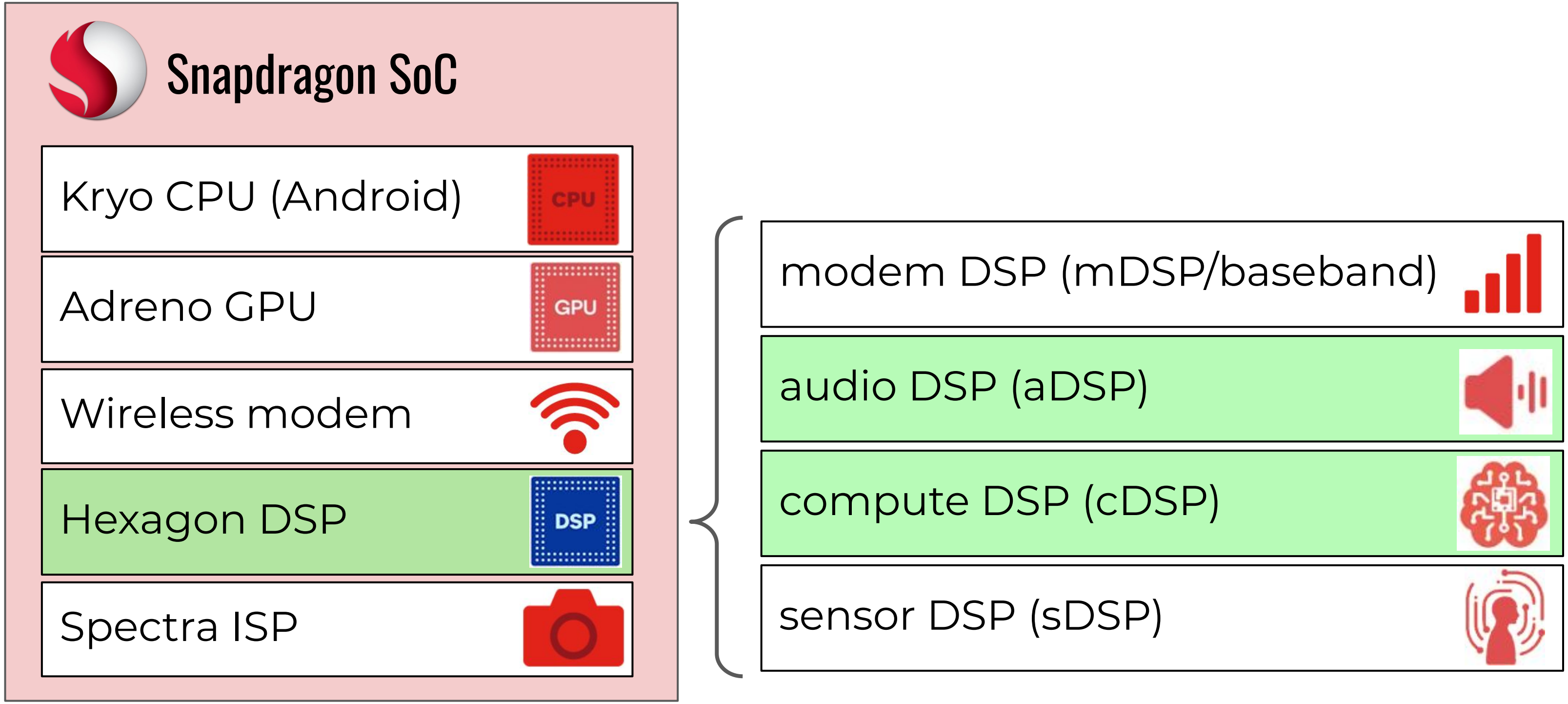
Pwn2Own Qualcomm cDSP



cp<r>

Slava Makkaveev

What processors are on your mobile phone?



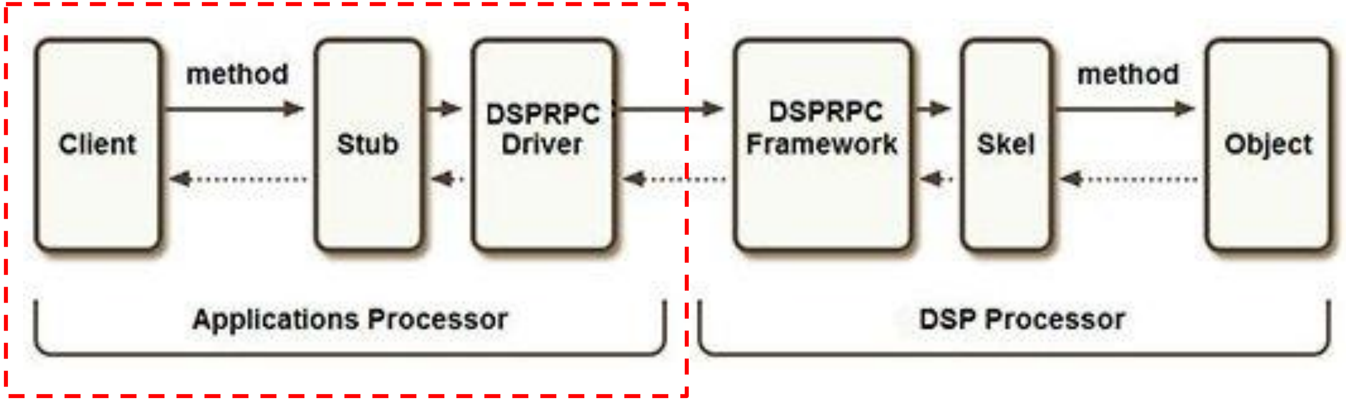
DSP assignment

- Low-power processing of audio and voice data
- Computer vision tasks
- Machine learning-related calculations
- Camera streaming
- Artificial intelligence
- …

aDSP is responsible for everything	Tasks are distributed between aDSP and cDSP
Snapdragon 835 (MSM8998): - Samsung S8 - OnePlus 5 - Sony Xperia XZ Premium	Snapdragon 855 (SM8150): - Google Pixel 4 - Samsung S10 - Xiaomi Mi9

Communication between the CPU and DSP

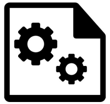
FastRPC mechanism (AP side)



Android
application

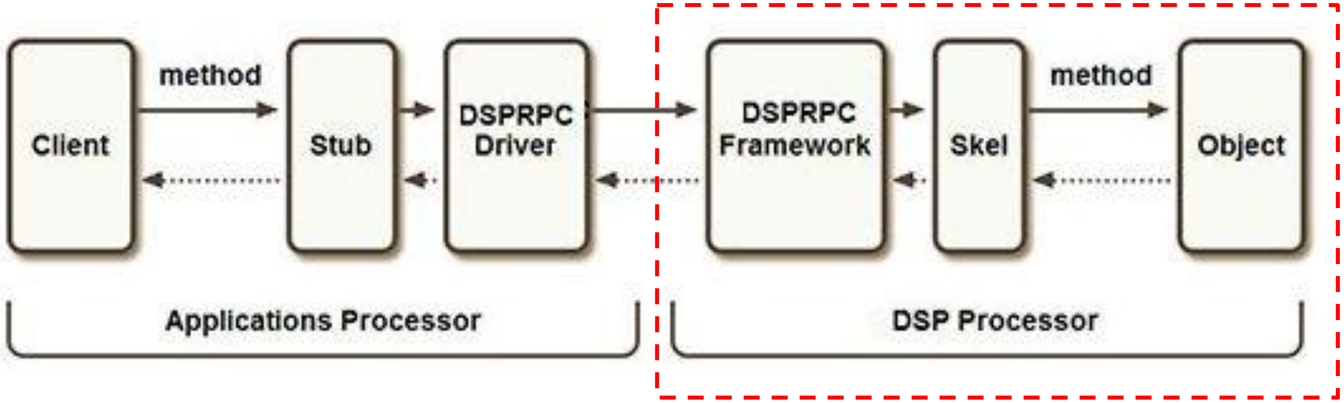
libXXX_stub.so
→ libadsprpc.so
→ libcdsprpc.so

ioctl



/dev/adsprpc-smd
/dev/cdsprpc-smd

FastRPC mechanism (DSP side)



fastrpc_shell_0
fastrpc_shell_3

libXXX_skel.so

libXXX.so

Who can run their own code on DSP?

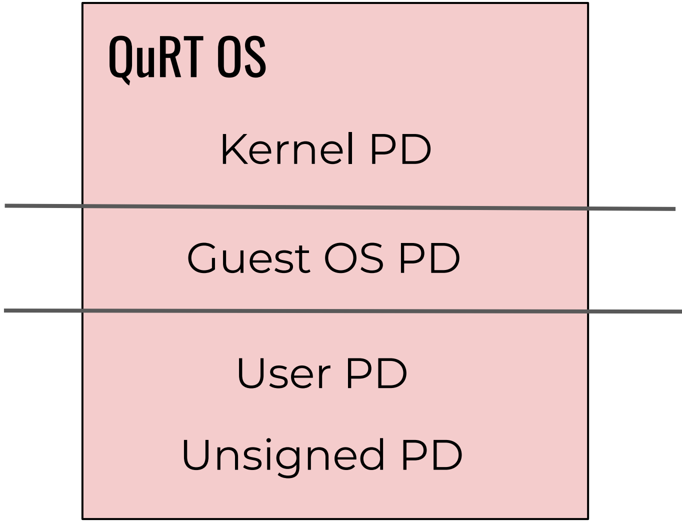
Can I compile my own DSP library? Yes

- Hexagon SDK is publically available
- *Stub* and *skel* code will be generated automatically

Can I execute this library on DSP? No

- DSP is licensed for programming by OEMs
  - The code running on the DSP is signed by Qualcomm
- Android app has no permissions to execute its own code on the DSP
  - Only prebuilt DSP libraries could be freely invoked

Who manages the DSP?



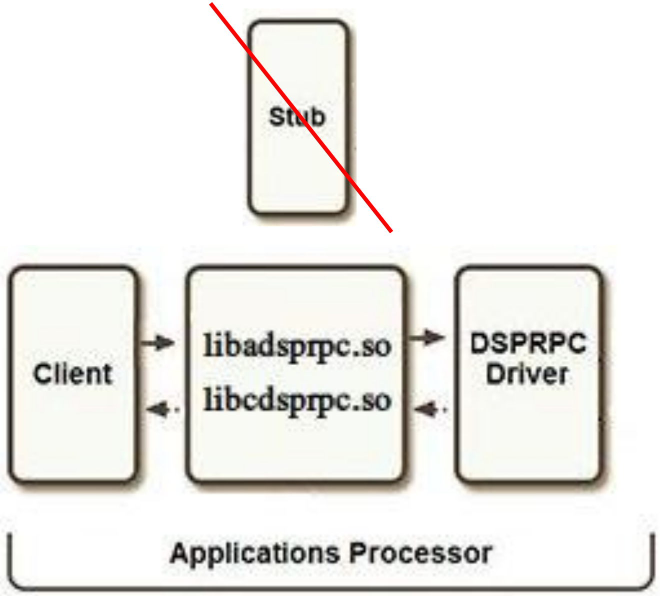
- ELF 32-bit executable, Qualcomm DSP6

```
/vendor/firmware/adsp
/vendor/firmware/cdsp
```

- Fastrpc shell ELFs
- Dozens of skeleton and object libraries

```
/dsp/*
/vendor/dsp/*
/vendor/lib/rfsa/adsp/*
```

Skipping stub code from the FastRPC flow




```
int remote_handle_open(
    const char* name,
    remote_handle *ph
)

int remote_handle_invoke(
    remote_handle h,
    uint32_t scalars,
    remote_arg *pra
)
```

Downgrade vulnerability CVE-2020-11209

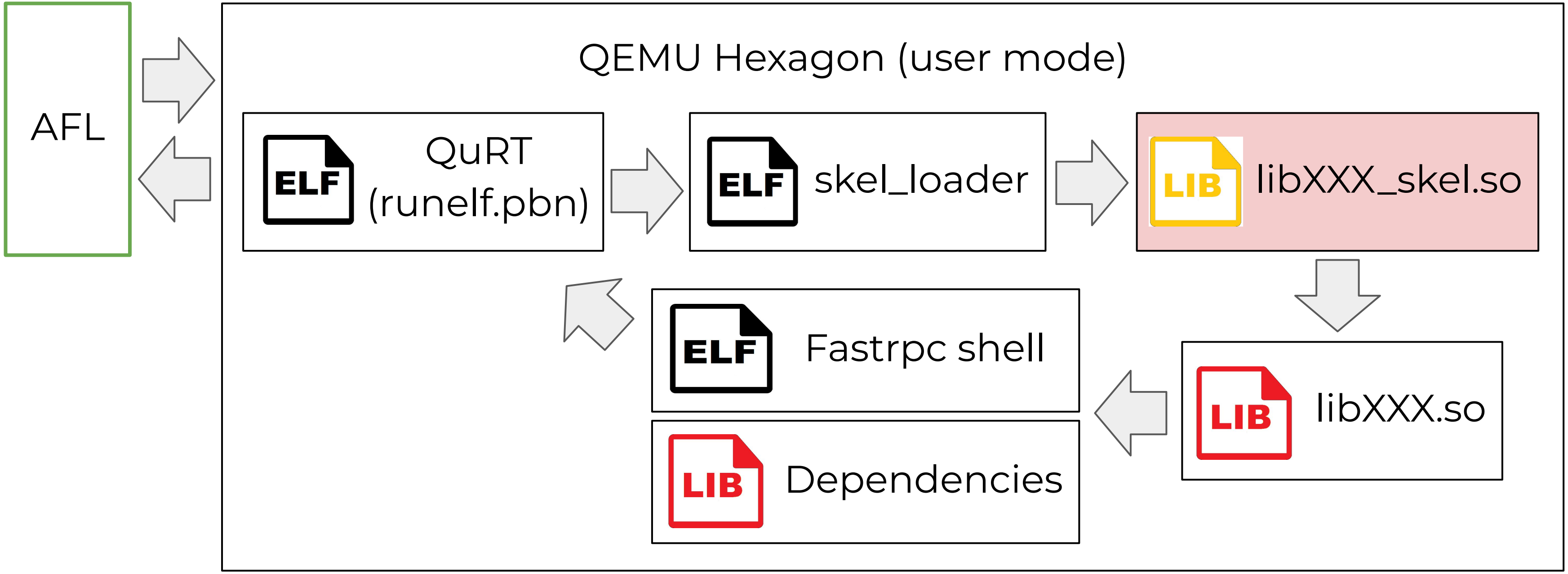
We cannot sign a skeleton library, but we can execute a signed one → Android application can bring any signed skeleton library and run it on the DSP

There is no version check of loading skeleton libraries → It is possible to run a very old skel library with a known 1-day vulnerability even if a patched library exists on the device

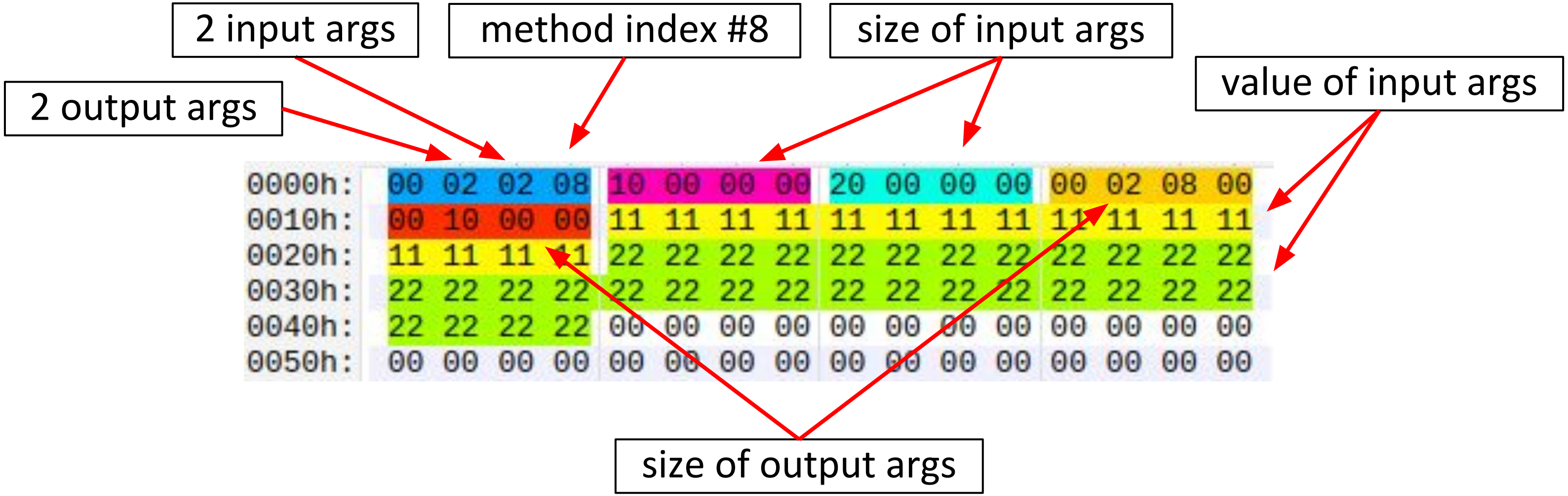
There are no lists of skeleton libraries permitted for the device → It is possible to run a library intended for one device on any other device

Feedback-based fuzzing of Hexagon libraries

Fuzzing scheme



Input file format



Fuzzing results

> 400 proven unique crashes in dozens of skeleton libraries

- `libfastcvadsp_skel.so`
- `libdepthmap_skel.so`
- `libscveT2T_skel.so`
- `libscveBlobDescriptor_skel.so`
- `libVC1DecDsp_skel.so`
- `libcamera_nn_skel.so`
- `libscveCleverCapture_skel.so`
- `libscveTextReco_skel.so`
- `libhexagon_nn_skel.so`
- `libadsp_fd_skel.so`
- `libqvr_adsp_driver_skel.so`
- `libscveFaceRecognition_skel.so`
- `libthread_blur_skel.so`
- ...

Do you remember? The skeleton code is auto generated by the Hexagon SDK. So, we are dealing with SDK issues!

Automatically Generated Code

Qualcomm Interface Definition Language (IDL)

- Define interfaces across memory protection and processor boundaries
- Exposes only what that object does, but not where it resides or the programming language in which it is implemented

`Hexagon SDK 3.5.1, hexagon_nn 2.10.1 library, hexagon_nn.idl`

```
/* Given a name, return the op ID */
long op_name_to_id(in string name, rout unsigned long node_id);

/* Pretty print the graph.  */
long snpprint(in hexagon_nn_nn_id id, inrout sequence<octet> buf);
```

Example: Marshaling an in-out buffer

hexagon_nn_stub.c

```c
static __inline int _stub_method_6(remote_handle _handle, uint32_t _mid,
                                   uint32_t _in0[1], char* _in1[1], uint32_t _in1Len[1],
                                   char* _rout1[1], uint32_t _rout1Len[1]) {
   ...
   _pra[0].buf.pv = (void*)_primIn;
   _pra[0].buf.nLen = sizeof(_primIn);

   _COPY(_primIn, 4, _in1Len, 0, 4);
   _COPY(_primIn, 8, _rout1Len, 0, 4);
   ...
}
__QAIC_STUB_EXPORT int __QAIC_STUB(hexagon_nn_snpprint)(hexagon_nn_nn_id id,
                                   unsigned char* buf, int bufLen) __QAIC_STUB_ATTRIBUTE {
   uint32_t _mid = 6;
   return _stub_method_6(_hexagon_nn_handle(), _mid, (uint32_t*)&id,
         (char**)&buf, (uint32_t*)&bufLen, (char**)&buf, (uint32_t*)&bufLen);
}
```

save buffer lengths as data

split *in-out* buffer into one *in* and one *out* buffer

Example: Unmarshaling an in-out buffer

hexagon_nn_skel.c

```
static __inline int _skel_method_25(int (*_pfn)(uint32_t, char*, uint32_t),
                                    uint32_t _sc, remote_arg* _pra) {
  ...
  _primIn = _pra[0].buf.pv;
  _COPY(_in1Len, 0, _primIn, 4, 4);
  _COPY(_rout1Len, 0, _primIn, 8, 4);
  _ASSERT(_nErr, (int)(_rout1Len[0]) >= (int)(_in1Len[0]));
  _MEMMOVEIF(_rout1[0], _in1[0], (_in1Len[0] * 1));
  ...
}
```

signed comparison of the buffer lengths

heap overflow

Hexagon SDK vulnerability CVE-2020-11208

- Hexagon SDK hiddenly injects vulnerabilities in the DSP libraries provided by Qualcomm, OEM and third-party vendors
- Dozens of DSP libraries embedded in Samsung, Pixel, LG, Xiaomi, OnePlus, HTC, Sony and other devices are vulnerable due to issues in Hexagon SDK

```
Qualcomm closed ~400 reported issues with one CVE-2020-11208 patch.
Did you use Hexagon SDK? Recompile your code!

In addition, CVE-2020-11201, CVE-2020-11202, CVE-2020-11206,
CVE-2020-11207 were assigned to issues in DSP object libraries
```

Exploiting a DSP vulnerability

Let’s execute unsigned code on DSP

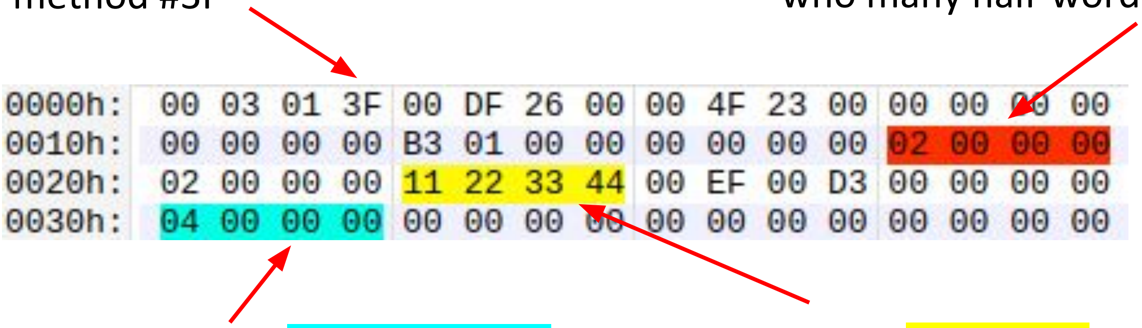
libfastcvadsp_skel.so library, version 1.7.1 from Sony Xperia XZ Premium (G8142) device

```
################################ Process on aDSP CRASHED!!!!!!! #####################################
--------------------- Crash Details are furnished below ------------------------------------
process "/frpc/f0554f20 skel_exec" crashed in thread "/frpc/f0554f20 " due to TLBMISS RW occurrence
Crashed Shared Object ./libfastcvadsp_skel.so load address : 0xEE500000
fastrpc_shell_0 load address : E9800000  and size : D6188
Fault PC   :    0xE04582BC
LR         :    0xEE54FB08
SP         :    0x3A688B88
Bad va     :    0xD1332491
FP         :    0x3A688BD8
SSR        :    0x21970870
Call trace:
[<EE54FB08>] fastcvadsp_fcvColorRGB888toYCrCbu8Q+0x808:     (./libfastcvadsp_skel.so)
[<EE569B4C>] fastcvadsp_fcvColorCbCrSwapu8Q+0x1C:     (./libfastcvadsp_skel.so)
[<EE52D408>] fastcvadsp_skel_invoke+0xE738:     (./libfastcvadsp_skel.so)
[<E9876C68>] mod_table_invoke+0x22C:     (fastrpc_shell_0)
[<E98958DC>] fastrpc_invoke_dispatch+0x15C:     (fastrpc_shell_0)
[<E98712B0>] HAP_proc_adaptive_qos+0x3BC:     (fastrpc_shell_0)
[<E9872F8C>] _pl_fastrpc_uprocess+0x794:     (fastrpc_shell_0)
----------------------------- End of Crash Report ---------------------------------------------
```

Arbitrary read-write in User PD

method #3F

who many half-words to read (the size)



where to read (the destination)

what to read (the source): the offset from the start of the first output argument in the DSP heap

Impact on device security

Android application gains DSP User PD possibilities:

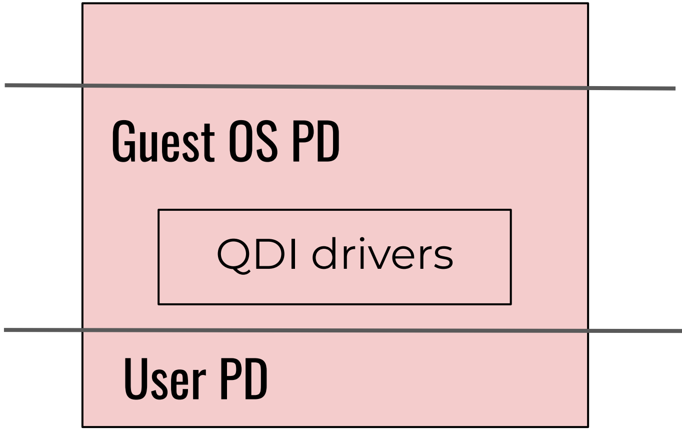
- Persistent DoS. Trigger a DSP kernel panic and reboot the mobile device
- Hide malicious code. Antiviruses do not scan the Hexagon instruction set
- The DSP is responsible for preprocessing streaming video from camera sensors. An attacker can take over this flow

  …

`The next step is to gain privileges of the Guest OS PD!`

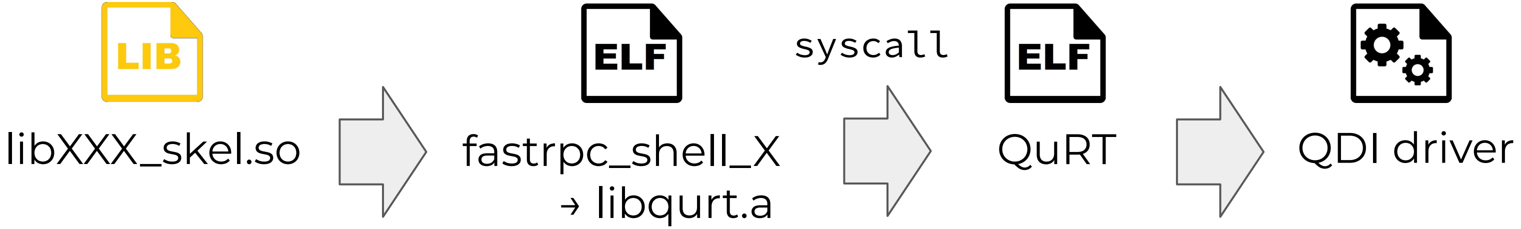
QuRT drivers

QuRT Driver Invocation (QDI) model



QuRT contains dozens of QDI drivers:

```
/dev/*      /drv/*      ...
/qdi/*      /adsp/*
/power/*    /qos/*
```



QDI API

driver name

method number

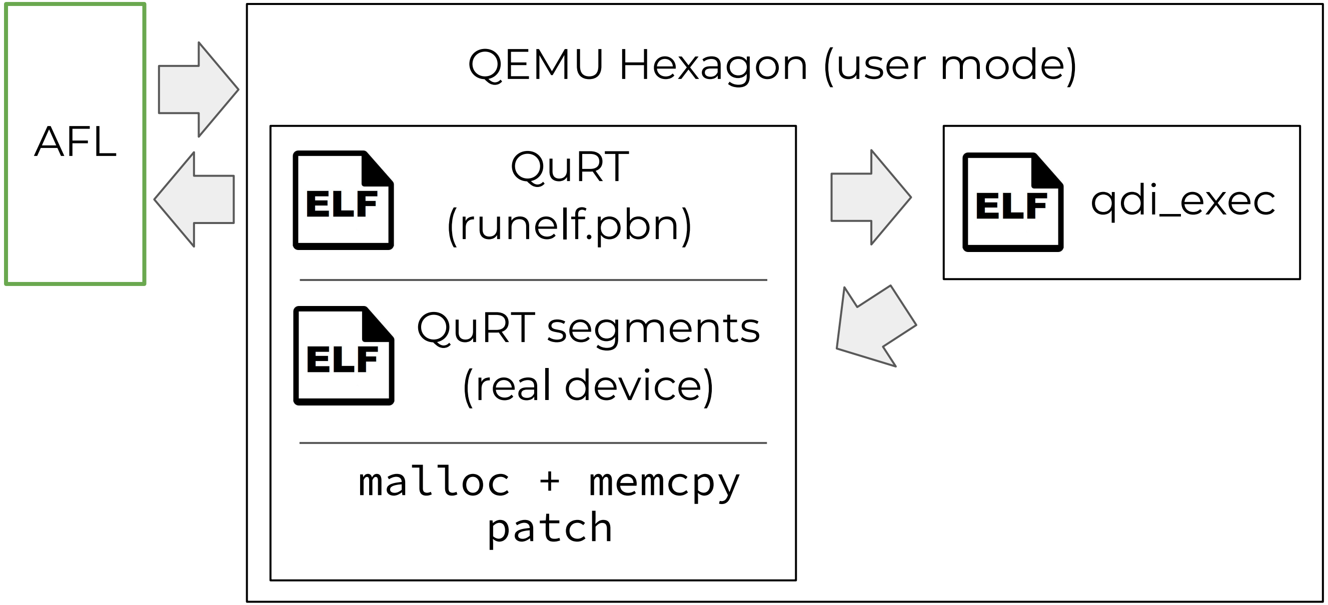
```c
int handle = qurt_qdi_open("/power/adsppm");
if (handle >= 0) {
    uint32_t clientId = 1;
    uint32_t result;
    int ret = qurt_qdi_handle_invoke(handle, 0x103, clientId, &result);
    ...
}
```

QDI handle

0 to 9 optional 32-bit arguments

```c
typedef union {
  void *ptr;
  int num;
} qurt_qdi_arg_t;
```

QDI feedback-based fuzzing



QDI vulnerabilities

A dozen Snapdragon 855 QDI drivers are vulnerable for PE and DoS attacks

Any failure in QDI drivers can be used to cause the DSP kernel panic

```
qurt_qdi_handle_invoke(qurt_qdi_open("/dev/procinfo"), 0x100, 2, 0, 0x05050505);
qurt_qdi_handle_invoke(qurt_qdi_open("/power/adsppm"), 0x101, 0, 0x05050505);
qurt_qdi_handle_invoke(qurt_qdi_open("/adsp/dcvs"), 0x102, 1, 0x05050505);
qurt_qdi_handle_invoke(qurt_qdi_open("/qos/dangergen"), 0x103, 0x05050505);
qurt_qdi_handle_invoke(qurt_qdi_open("/dev/diag"), 0x104, 0xf, 0, 0, 0x05050505, 1, 1);
qurt_qdi_handle_invoke(qurt_qdi_open("/dev/smp2p"), 0x105, 0x05050505);
```

We exploited

- several arbitrary kernel read and write vulnerabilities in *dev/i2c* QDI driver
- two code execution vulnerabilities in */dev/glink* QDI driver

Demo. Code execution in Guest OS PD

Instead of a conclusion

Qualcomm aDSP and cDSP subsystems are very promising areas for security research

- The DSP is accessible for invocations from third-party Android applications
- The DSP processes personal information such as video and voice data that passes through the device's sensors
- As we have proven, there are many security issues in the DSP components

Thank you!

slavam@checkpoint.com

@_cpresearch_
research.checkpoint.com