Artifacts for Demystifying Device-specific Compatibility Issues in Android Apps

Step 1: Repository collection

We used the iterative algorithm to crawl repositories from GitHub, whose implementation is the script `github_crawler.py`.

The result is presented in `dataset/crawled-data-1681274668.json`.

Step 2: Identifying device-specific compatibility issues

We cloned repositories listed in the `crawled-data-*.json`

The `StageI_repos.txt` contains the repositories we used in the study, and the `StageI_repo_commithash.txt` contains the commit hash of each repository we used.

> If you encounter any difficulties accessing these repositories, please look for the copies we have provided in the repo-archive.
>
> We offer the copies that were used during our research, but we do not guarantee that they are the most recent versions.
>
> We have packaged 333 repositories, sorted by the first letter, into the corresponding zip file.

We extracted lines of code containing accesses to the class `android.os.Build`.

We used the following two commands.

Their results are stored in `rawdata_java.txt` and `rawdata_kotlin.txt`.

```
# for java
grep -irnE --include "*.java" "Build\.
(BOARD|BRAND|DEVICE|MANUFACTURER|MODEL|PRODUCT)"
# for kotlin
grep -irnE --include "*.kt" "Build\.
(BOARD|BRAND|DEVICE|MANUFACTURER|MODEL|PRODUCT)"
```

Then we used the following command to filter out irrelevant lines from the Java set. The `-v` flag is used to invert the selection. The result is stored in `StageI_filtered_java.txt`

```
cat rawdata_java.txt | grep -vE "^.*\.(java|kt).*(append|(String\.format)|
(Log\.(v|d|i|w|e|f))|(\"\ *\+\ *Build\..*\+\ *\".*\")|(//.*Build.*)|
(/\s*\*.*Build)|(\s*\".*\"\s*\+\s*(android\.os\.)?Build))"
```

The `StageI_filtered_java.txt` and `rawdata_kotlin.txt` are composed of our sampling sources.

Step 3: Manual inspection

Our inspection of the dataset is shown in the `Device-specific FIC Issues.xlsx` file.

The `Raw Java` and `Raw Kotlin` sheets contain the raw data after Stage I.

The `Final` sheet is used to count data eventually used for RQ 1,2,3.
The J (Type) column stans for our classification result of RQ.1.
The R (Android component) and S (Component 2) columns stand for our classification of RQ.2.
The K (Pattern) column stands for our classification of RQ.3.

RQ.3 Examples

We show some representative examples in the RQ.3-examples folder.

Pattern a:

1. CameraController2.java, Line 589, collected from https://github.com/MobileRoboticsSkoltech/OpenCamera-Sensors

```
this.is_samsung_s7 = Build.MODEL.toLowerCase(Locale.US).contains("sm-g93");
// ...
if( is_samsung_s7 ) {
    if( MyDebug.LOG )
        Log.d(TAG, "set EDGE_MODE_OFF");
    // see
https://sourceforge.net/p/opencamera/discussion/general/thread/48bd836b/ ,
    // https://stackoverflow.com/questions/36028273/android-camera-api-glossy-
effect-on-galaxy-s7
    // need EDGE_MODE_OFF to avoid a "glow" effect
    builder.set(CaptureRequest.EDGE_MODE, CaptureRequest.EDGE_MODE_OFF);
}
```

2. OpenVPNService.java, Line 868, collected from https://github.com/KaustubhPatange/Moviesy

```
if ("samsung".equals(Build.BRAND) && Build.VERSION.SDK_INT >=
Build.VERSION_CODES.LOLLIPOP && mDnslist.size() >= 1) {
    // Check if the first DNS Server is in the VPN range
    try {
        IpAddress dnsServer = new IpAddress(new CIDRIP(mDnslist.get(0), 32),
true);
        boolean dnsIncluded = false;
        for (IpAddress net : positiveIPv4Routes) {
            if (net.containsNet(dnsServer)) {
                dnsIncluded = true;
            }
        }
        if (!dnsIncluded) {
            String samsungwarning = String.format("Warning Samsung Android 5.0+
devices ignore DNS servers outside the VPN range. To enable DNS resolution a
route to your DNS Server (%s) has been added.", mDnslist.get(0));
            VpnStatus.logWarning(samsungwarning);
            positiveIPv4Routes.add(dnsServer);
        }
    } catch (Exception e) {
        // If it looks like IPv6 ignore error
        if (!mDnslist.get(0).contains(":"))
            VpnStatus.logError("Error parsing DNS Server IP: " +
mDnslist.get(0));
    }
}
```

Pattern b:

1. BaseActivity.java, Line 18, collected from https://github.com/deltachat/deltachat-android

```
@Override
public boolean onKeyUp(int keyCode, @NonNull KeyEvent event) {
    if (keyCode == KeyEvent.KEYCODE_MENU && isMenuWorkaroundRequired()) {
        openOptionsMenu();
        return true;
    }
    return super.onKeyUp(keyCode, event);
}
```

2. ParcelFileDescriptorBitmapDecoder.java, Line 36, collected from https://github.com/bumptech/glide

```
private boolean isSafeToTryDecoding(@NonNull ParcelFileDescriptor source) {
    if ("HUAWEI".equalsIgnoreCase(Build.MANUFACTURER)
          || "HONOR".equalsIgnoreCase(Build.MANUFACTURER)) {
        return source.getStatSize() <=
MAXIMUM_FILE_BYTE_SIZE_FOR_FILE_DESCRIPTOR_DECODER;
    }
    return true;
}
```

Pattern c:

1. CameraXActivity.java, Line 555, collected from https://github.com/androidx/androidx

```
createDefaultVideoFolderIfNotExist();
final PendingRecording pendingRecording;
if (DeviceQuirks.get(MediaStoreVideoCannotWrite.class) != null) {
    // Use FileOutputOption for devices in MediaStoreVideoCannotWrite Quirk.
    pendingRecording = getVideoCapture().getOutput().prepareRecording(
            this, getNewVideoFileOutputOptions());
} else {
    // Use MediaStoreOutputOptions for public share media storage.
    pendingRecording = getVideoCapture().getOutput().prepareRecording(
            this, getNewVideoOutputMediaStoreOptions());
}

resetVideoSavedIdlingResource();

mActiveRecording = pendingRecording
        .withAudioEnabled()
        .start(ContextCompat.getMainExecutor(CameraXActivity.this),
                mVideoRecordEventListener);
mRecordUi.setState(RecordUi.State.RECORDING);
```

2. CameraApiLegacy.java, Line 734, collected from https://github.com/TGX-Android/Telegram-X

```
private File configureRecorder (MediaRecorder recorder) {
    int orientation = mForcedOutputOrientation;
    if (cameraInfo.facing == Camera.CameraInfo.CAMERA_FACING_FRONT) {
      orientation = (cameraInfo.orientation + orientation) % 360;
      orientation = (360 - orientation) % 360;

      if (orientation == 90) {
        orientation = 270;
      }
      if ("Huawei".equals(Build.MANUFACTURER) && "angler".equals(Build.PRODUCT)
&& orientation == 270) {
        orientation = 90;
      }
    } else {
      orientation = (cameraInfo.orientation - orientation + 360) % 360;
    }
    recorder.setOrientationHint(orientation);
    // ...
}
```

Pattern d:

1. UseTorchAsFlash: collected from https://github.com/androidx/androidx

Camera2CapturePipeline.java, Line 191: function `isTorchAsFlash`

```
@ExecutedBy("mExecutor")
@NonNull
public ListenableFuture<List<Void>> submitStillCaptures(
        @NonNull List<CaptureConfig> captureConfigs, @CaptureMode int
captureMode,
        @FlashMode int flashMode, @FlashType int flashType) {

    OverrideAeModeForStillCapture aeQuirk = new
OverrideAeModeForStillCapture(mCameraQuirk);
    Pipeline pipeline = new Pipeline(mTemplate, mExecutor, mCameraControl,
mIsLegacyDevice,
            aeQuirk);

    if (captureMode == CAPTURE_MODE_MAXIMIZE_QUALITY) {
        pipeline.addTask(new AfTask(mCameraControl));
    }

    if (mHasFlashUnit) {
        if (isTorchAsFlash(flashType)) {
            pipeline.addTask(new TorchTask(mCameraControl, flashMode,
mExecutor));
        } else {
            pipeline.addTask(new AePreCaptureTask(mCameraControl, flashMode,
aeQuirk));
        }
    } // If there is no flash unit, skip the flash related task instead of
failing the pipeline.

    return Futures.nonCancellationPropagating(
            pipeline.executeCapture(captureConfigs, flashMode));
}
```

UseTorchAsFlash.java, `quirks.contains(UseTorchAsFlashQuirk.class);`

`UseTouchAsFlashQuirk` is a pure interface.

```
public class UseTorchAsFlash {

    private final boolean mHasUseTorchAsFlashQuirk;

    public UseTorchAsFlash(@NonNull Quirks quirks) {
        mHasUseTorchAsFlashQuirk = quirks.contains(UseTorchAsFlashQuirk.class);
    }

    /** Returns if torch should be used as flash. */
    public boolean shouldUseTorchAsFlash() {
        return mHasUseTorchAsFlashQuirk;
    }
}
```

`CameraNoResponseWhenEnablingFlashQuirk`, `FlashTooSlowQuirk`, `ImageCaptureFlashNotFireQuirk`, `ImageCaptureWashedOutImageQuirk`, `ImageCaptureWithFlashUnderexposureQuirk` implements the interface `UseTorchAsFlashQuirk`.

2. MediaCodecUtil.java, Line 453, function `isCodecUsableDecoder`, collected from https://github.com/google/ExoPlayer

```

  /**
   * Returns whether the specified codec is usable for decoding on the current
device.
   *
   * @param info The codec information.
   * @param name The name of the codec
   * @param secureDecodersExplicit Whether secure decoders were explicitly
listed, if present.
   * @param mimeType The MIME type.
   * @return Whether the specified codec is usable for decoding on the current
device.
   */
  private static boolean isCodecUsableDecoder(
      android.media.MediaCodecInfo info,
      String name,
      boolean secureDecodersExplicit,
      String mimeType) {
```

```
    if (info.isEncoder() || (!secureDecodersExplicit &&
name.endsWith(".secure"))) {
      return false;
    }

    // Work around broken audio decoders.
    if (Util.SDK_INT < 21
        && ("CIPAACDecoder".equals(name)
            || "CIPMP3Decoder".equals(name)
            || "CIPVorbisDecoder".equals(name)
            || "CIPAMRNBDecoder".equals(name)
            || "AACDecoder".equals(name)
            || "MP3Decoder".equals(name))) {
      return false;
    }

    // Work around https://github.com/google/ExoPlayer/issues/1528 and
    // https://github.com/google/ExoPlayer/issues/3171.
    if (Util.SDK_INT < 18
        && "OMX.MTK.AUDIO.DECODER.AAC".equals(name)
        && ("a70".equals(Util.DEVICE)
            || ("Xiaomi".equals(Util.MANUFACTURER) &&
Util.DEVICE.startsWith("HM")))) {
      return false;
    }

    // ...

    return true;
  }
```

Pattern e

1. FlashAvailabilityChecker.java, Line 65, function `isFlashAvailable`, collected from https://github.com/androidx/androidx

```
public static boolean isFlashAvailable(boolean allowRethrowOnError,
        @NonNull CameraCharacteristicsProvider provider) {
    Boolean flashAvailable;
    try {
        flashAvailable =
provider.get(CameraCharacteristics.FLASH_INFO_AVAILABLE);
    } catch (BufferUnderflowException e) {
        if (DeviceQuirks.get(FlashAvailabilityBufferUnderflowQuirk.class) !=
null) {
            Logger.d(TAG, String.format("Device is known to throw an exception
while "
                    + "checking flash availability. Flash is not available. "
                    + "[Manufacturer: %s, Model: %s, API Level: %d].",
                    Build.MANUFACTURER, Build.MODEL, Build.VERSION.SDK_INT));
        } else {
            Logger.e(TAG, String.format("Exception thrown while checking for
flash "
                            + "availability on device not known to throw
exceptions during "
```

```
                                + "this check. Please file an issue at "
                                + "https://issuetracker.google.com/issues/new?
component=618491"
                                + "&template=1257717 with this error message "
                                + "[Manufacturer: %s, Model: %s, API Level: %d].\n"
                                + "Flash is not available.",
                        Build.MANUFACTURER, Build.MODEL, Build.VERSION.SDK_INT),
e);
        }

        if (allowRethrowOnError) {
            throw e;
        } else {
            flashAvailable = false;
        }
    }
    if (flashAvailable == null) {
        Logger.w(TAG, "Characteristics did not contain key
FLASH_INFO_AVAILABLE. Flash is not"
                + " available.");
    }
    return flashAvailable != null ? flashAvailable : false;
}
```

2. RQTunnelingVpnService.java, Line 147, function `connect`, collected from https://github.com/krlvm/PowerTunnel-Android

```
try {
    vpn = getBuilder().establish();
} catch (Throwable t) {
    Log.e(LOG_TAG, "Failed to establish VPN Service: " + t.getMessage(), t);
    error = t.getMessage();
    if(t instanceof SecurityException &&
            Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q &&
            (Build.MODEL != null && Build.MODEL.toLowerCase().startsWith("sm-
"))
    ) {
        Log.e(LOG_TAG, "Most likely VPN Service establishing failure was caused
by firmware bug");
        isFirmwareBug = true;
    }
}
```

See the issues above for details. Multiple VPN software suffer from this bug.

https://github.com/krlvm/PowerTunnel-Android/issues/29

https://github.com/tailscale/tailscale/issues/2180

https://github.com/schwabe/ics-openvpn/issues/555

https://github.com/AdguardTeam/AdguardForAndroid/issues/3299