Research-fueled Security Services

\ WHITE PAPER \

Facial Recognition Research

Gabriel Gonzalez
Director of Hardware Security

Alejo Moles
Security Consultant

February 2022

Contents

Introduction

IOActive, Inc. (IOActive) has conducted extensive research and testing of facial recognition systems on commercial mobile devices. Our testing lab includes testing setups for 2D- and 3D-based algorithms, including technologies using stereo IR cameras.

For each of the different technologies, we first try to understand the underlying algorithms and then come up with creative and innovative setups to bypass them. Once an unlock is achieved, we calculate the Spoof Acceptance Rate (SAR), as described in the Measuring Biometric Unlock Security" section of the Android Compatibility Definition Document.[1] This metric allows us to compare different solutions and evaluate the strength of each solution.

This document describes IOActive's results for commercially available mobile phones implementing face authentication mechanisms to unlock the device. All them relied on the "selfie-camera," a single lens producing 2D RGB images. IOActive used 2D and 3D masks when attempting to bypass the security features.

Our testing procedure for 2D masks included techniques such as retouching the pictures (i.e., increasing contrast and brightness to make images appear more realistic), using different types of background (e.g., white and black), and placing the 2D masks on different surfaces to evaluate the influence of the surroundings and sense of depth on each algorithms' behavior. For 3D masks, our testing setup included composite-made 3D face masks of several subjects. These were produced by generating 3D models based on 2D pictures.

IOActive successfully bypassed the facial recognition security mechanism on all of the devices we tested for at least one of the participating subjects.

In addition to these test results, IOActive selected one device and conducted further analysis of the software to get a better sense of the algorithm running behind the scenes.

Note: all the parties affected by the results of this research have been notified accordingly.

[1] https://source.android.com/security/biometric/measure

Facial Recognition Testing Methodology

Protocol Test Procedure

Protocol testing includes several procedures to evaluate the performance of the facial recognition algorithms against 2D images, either using a display or a simple 2D mask (printed on paper) created from a photo.

Test Environment

The studio setup is used to enroll the subjects and take photos. The setup consists of an off-white background, a chair to position the subject facing a window (source of natural light), a lamp to illuminate the face from the side, and a tripod to hold the mobile phone.

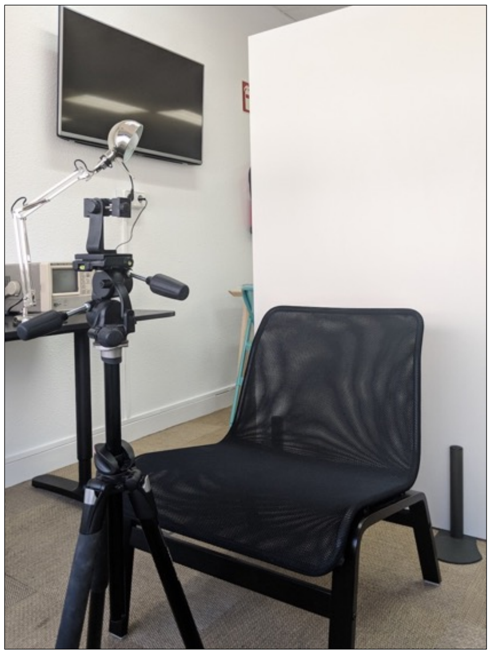


Figure 1. Studio Setup

2D Test Setup

For 2D testing, IOActive constructed two setups: one with a display and one with a 2D mask. We use the display as the primary setup and the printed mask as the secondary setup. The idea is to determine if the secondary setup yields better results in situations where the primary setup fails.

The first setup consists of a regular screen displaying the photo of the subject and a tripod to hold the mobile phone. In this setup, after a picture of the subject is taken, it is checked for sharpness and retouched as needed.

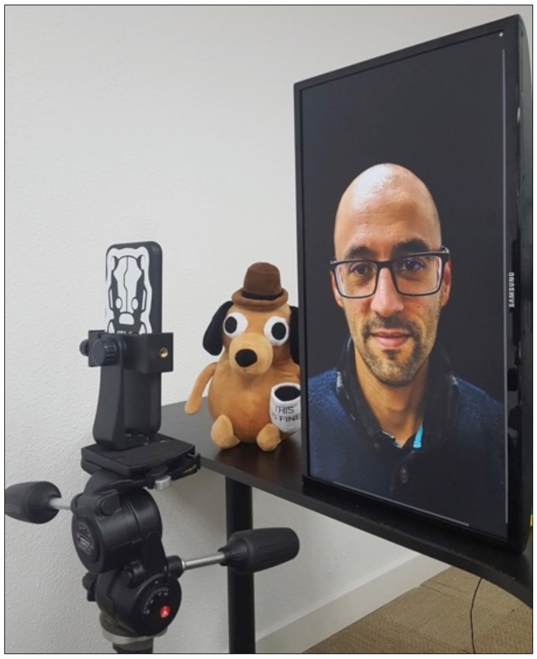


Figure 2. 2D Display Setup

The second setup consists of the same components as mentioned above, with the addition of a 2D mask and a desk lamp. The retouched photos from the first setup are printed at a local retail photography store. The mask is cut out and attached to the display, which is leveraged to modify the background on-the-fly. The desk lamp is used to simulate warm light and make the photos' colors appear more realistic.

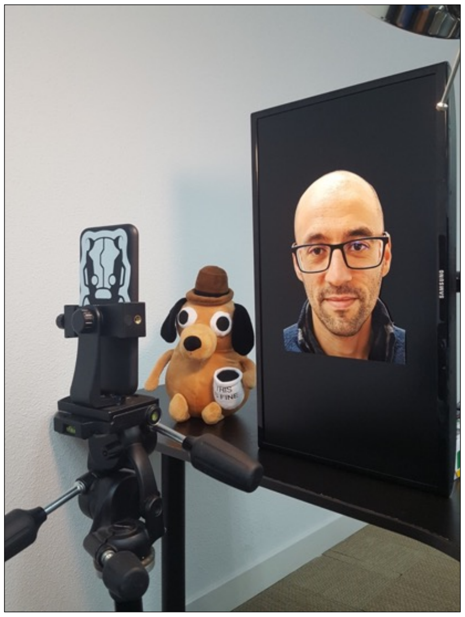


Figure 3. 2D Mask Setup

Retouching

The physical accuracy of the represented subject is vital since an inaccurate representation of facial features will likely not result in a match when compared against the enrolled reference image. For this reason, the pictures are taken with the subjects facing the camera directly at eye-level under the same light conditions and at approximately the same distance as during enrollment.

Adjusting the temperature of colors can more faithfully reproduce the skin color of the subject. Furthermore, increasing the contrast, selectively lightening or darkening parts of the face, and blurring the background or changing its color altogether can improve the perceived depth.

3D Test Setup

For the 3D tests, the setup consists of a 3D mask, a desk lamp to illuminate it, and a tripod to hold the mobile phone. The lamp is used to simulate warm light and give the appearance of a more natural skin tone; it also creates shadows.

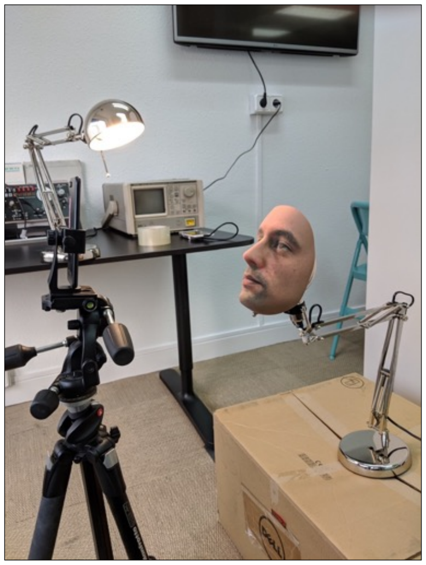


Figure 4. 3D Setup

Partially Covered Face Procedure

IOActive also tested the behavior of facial recognition systems with partially covered faces. From a black-box perspective, this test helps narrow down the number of features required to properly unlock a device.

The consultants covered both the subjects' faces as well as the 2D and 3D masks used to impersonate the subjects. The following materials were used to cover the surfaces:

- Textile
- Translucent paper
- Regular paper
- High-quality photographic paper

The surfaces were covered both in the median and transverse planes using the following patterns:

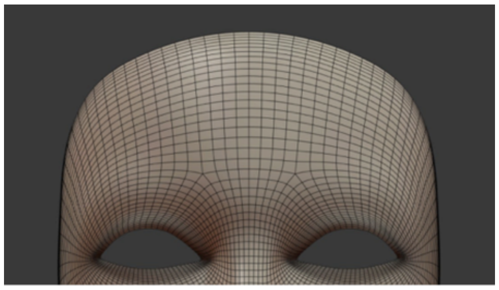


Figure 5. Transverse Cover Pattern

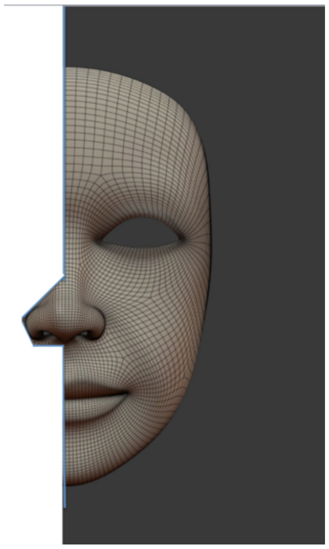


Figure 6. Median Cover Pattern

Imposter Test Procedure

IOActive also conducted imposter tests using pairs of subjects with similar physical characteristics (candidates A and B).

Tests are carried out in both directions. Candidate A is enrolled on a device and candidate B attempts to unlock it, then candidate B is enrolled and candidate A attempts to unlock the device.

IOActive looked for corner cases where the algorithm barely recognized a face. Under these circumstances, it is more likely the system could fail. Some of these corner cases were positions and angles where facial characteristics were hidden or heavily deformed by the camera lens.

Facial Recognition Research

Devices

The devices used in this research were a Samsung Galaxy S10+, OnePlus 7 Pro, VIVO V15 Pro, Xiaomi Mi 9, and Nokia 9 Pure View with the characteristics shown below.

Table 1. Overview of tested devices and technical specifications

Device Model	OS Version	Build Number	Resolution	Focal Length	Aperture
Samsung S10(+)	Android 10/ One UI 2.1	6975FXXU7CTF1	10MP	35mm	f/1.9
OnePlus 7 Pro	Android 10/ Oxygen OS	10.0.7.GM21BA	16MP	25mm	f/2.0
Nokia 9 Pure View	Android 10	00WW_5_13D_SP01	20MP	1.0µm	f/2.0
Xiaomi Mi 9	Android 10/ MIUI 11.0.6	QKQ1.190825.002	20MP	0.9µm	f/2.2
Vivo V15 Pro	Android 10/ Funtouch OS_10	PD1832F_EX_A_6.19.9	32MP	X	f/2.0

Test Parameters

Subjects

We used subjects of various ethnicities: two Asian (male and woman), two African American (male and female), and one Caucasian (male).

Note that while we have subjects of three different ethnicities, the sample size is not sufficiently large enough to conclusively identify a statistically significant causal relationship between ethnicity and success rate.

Success Rate

For each subject and each device, the SAR was calculated as follows:

$$SAR = \frac{\sum_1^E S_i}{U * E} * 100\%$$

Where S_i is a successful authentication attempt, E is the number of enrolled faces and U is the number of authentication attempts. For these tests, an U of 20 was used, unless stated otherwise.

Black-box 2D Test Results

The following table illustrates the unlock probability observed during black-box testing using the following color code:

- 🟥 The mobile phone would unlock reliably with the 2D image without triggering the maximum attempts mechanism
- 🟧 The mobile phone would unlock occasionally with the 2D image
- 🟩 The mobile phone never unlocked with the 2D image

Table 2. Black-box 2D Test results

Subject	Samsung S10(+)	OnePlus 7 Pro	Nokia 9 Pure View	Xiaomi Mi 9	Vivo V15 Pro
Subject 1 (Caucasian, Male)	🟩	🟩	🟩	🟩	🟩
Subject 2 (Asian, Female)	🟩	🟥	🟥	🟥	🟩
Subject 3 (Asian Male)	🟧	🟩	🟥	🟧	🟩
Subject 4 (African Female)	🟩	🟥	🟥	🟩	🟩
Subject 5 (African Male)	🟥	🟩	🟥	🟥	🟥

Again, while this sample size is insufficient to produce a statistically significant link between ethnicity and unlock success rate, it does indicate additional investigation is warranted.

Results Discussion

Potential physical causes for this weakly indicated causal relationship include the dynamic range of the camera and the facial feature contrast of individuals due to skin tone. The test setup attempts to control the impact of environmental and lighting conditions during the unlock attempt.

Implementation issues may include a composition bias in the corpus used to initially train the facial recognition algorithms used in these devices.

Should a statistically significant link between ethnicity and unlock success rate be confirmed and the primary causative factor for unintended unlock success rate be tied to the contrast of facial features, then alternate solutions for biometric security fingerprint detection should be preferred over facial recognition. Many earlier generations of phones and tablets supported these methods in addition to or in lieu of facial recognition.

Case Study: OnePlus 7 Pro

Facial Recognition System Analysis

In this section we describe the modules that compose the facial recognition environment in Android devices. All vendors follow the guidelines describe in the Face Authentication HIDL documentation.[2] We are using the OnePlus 7 Pro device's implementation as an example as we found it quite descriptive.

All vendors use the basic architecture shown in Figure 7.

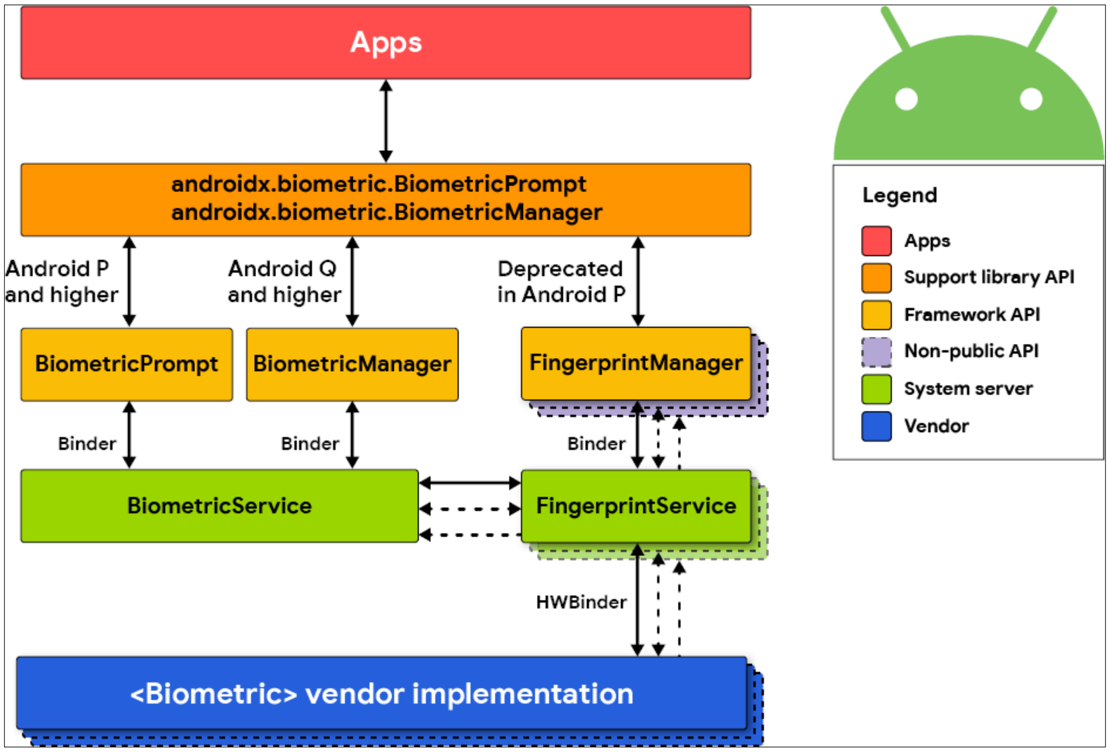


Figure 7. Basic Architecture

There are three interesting components that are useful for our goals:

- **App:** Each vendor has its own Android application whose main function is to collect images, extract the facial features they consider interesting, and manage all the lock/unlock logic based on an IA model (this model as we will see has to be in a secure environment by definition).
- **BiometricManager**: This is a private interface that maintains a connection with BiometricService.

[2] https://source.android.com/security/biometric/face-authentication

- **Biometric vendor implementation**: This must use secure hardware to ensure the integrity of the stored face data and the authentication comparison.

Figure 8 shows the flow the solution should follow.

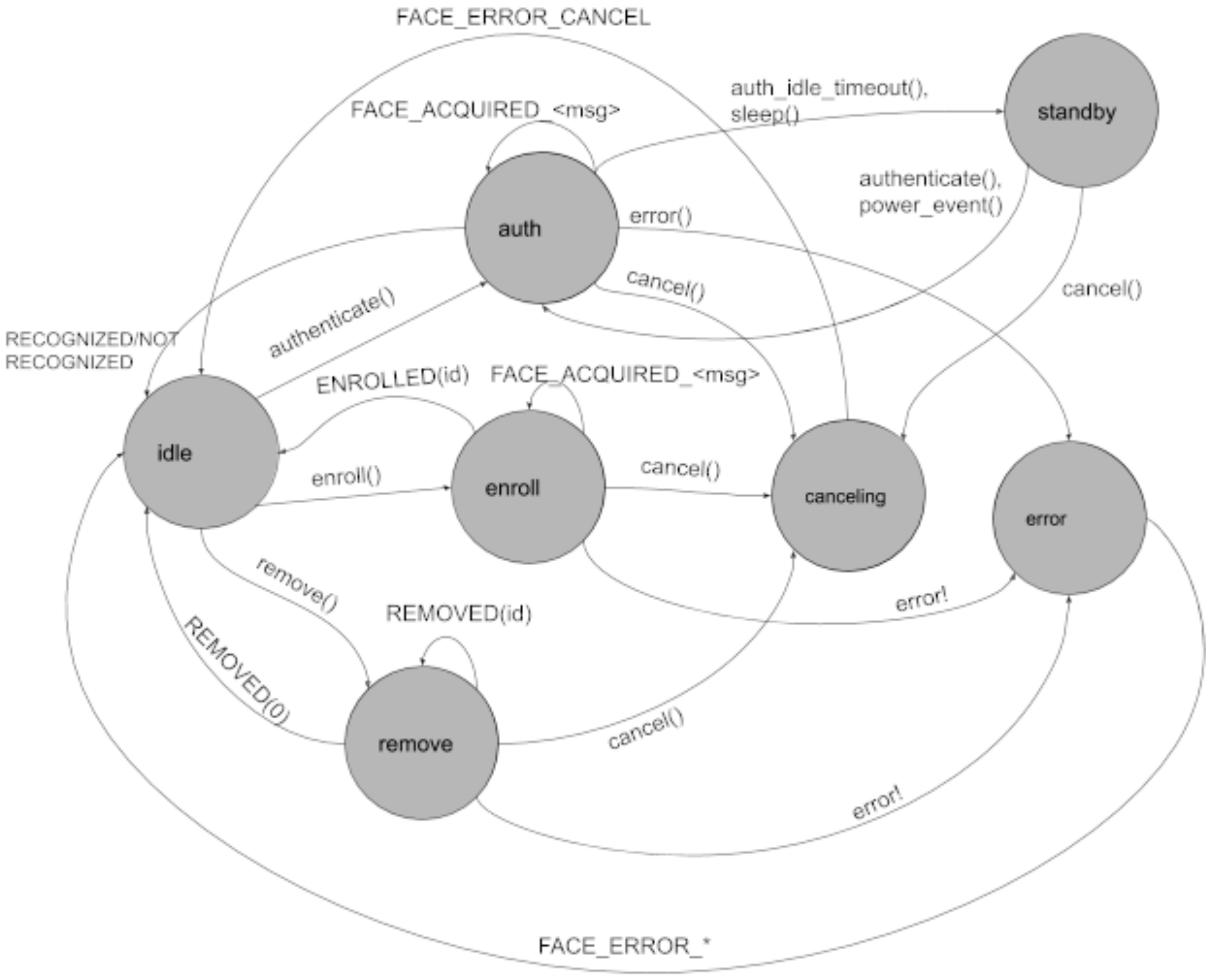
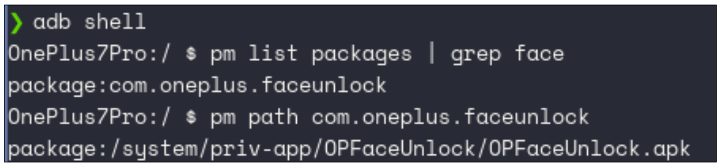


Figure 8. Flow Diagram

To identify the components, we took a close look the vendor's application named OPFaceUnlock (**App**).



```
❯ adb shell
OnePlus7Pro:/ $ pm list packages | grep face
package:com.oneplus.faceunlock
OnePlus7Pro:/ $ pm path com.oneplus.faceunlock
package:/system/priv-app/OPFaceUnlock/OPFaceUnlock.apk
```

The following flow graph provides a quick overview of the part of the app we are interested in analyzing.

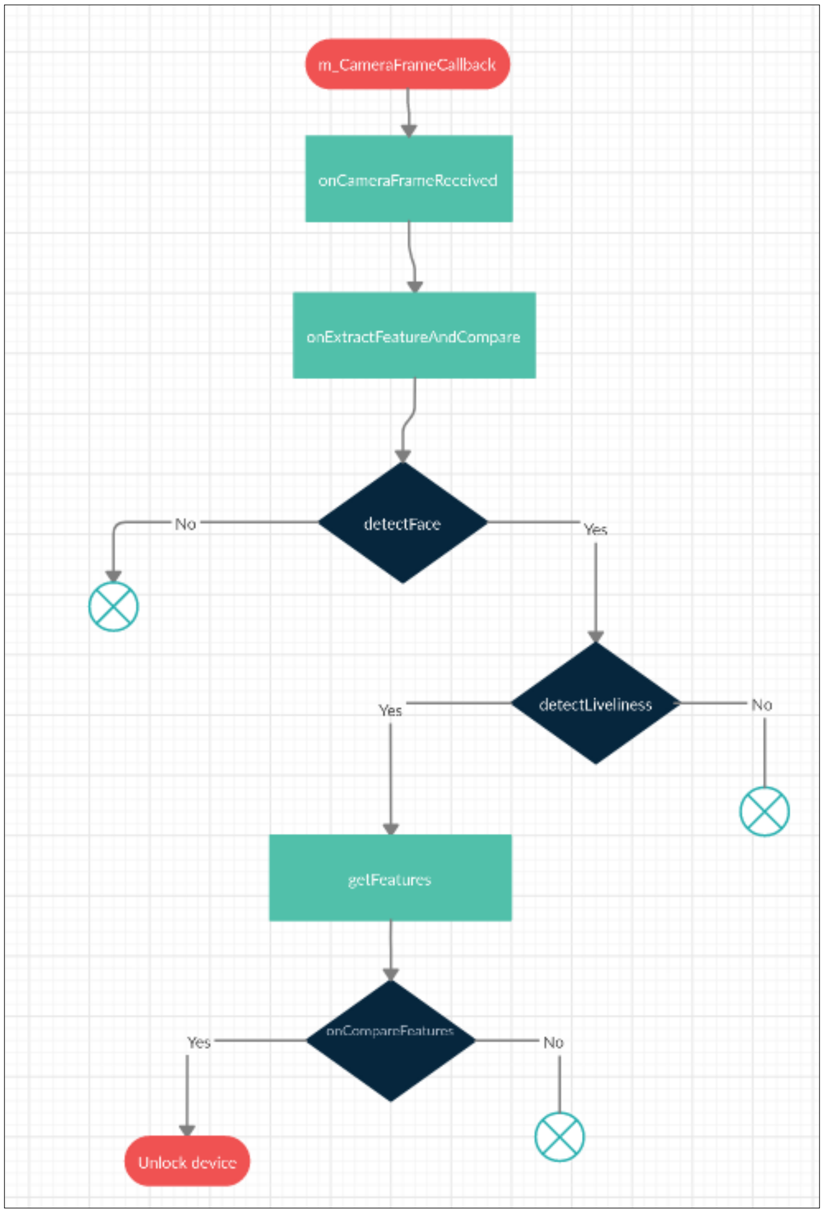
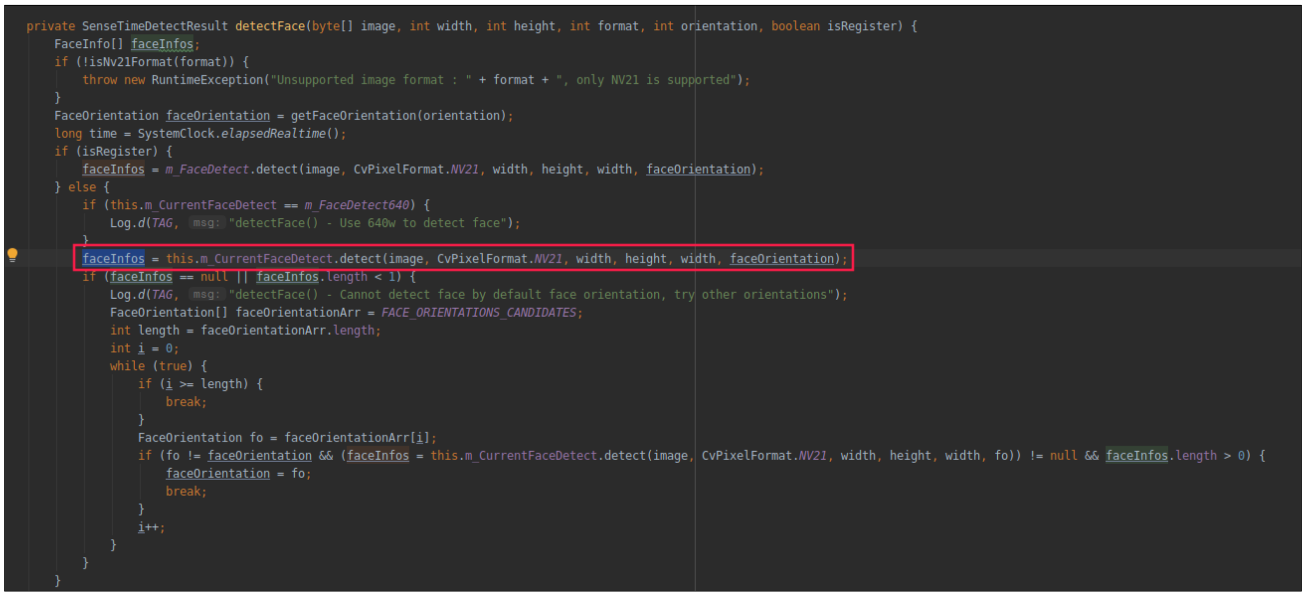


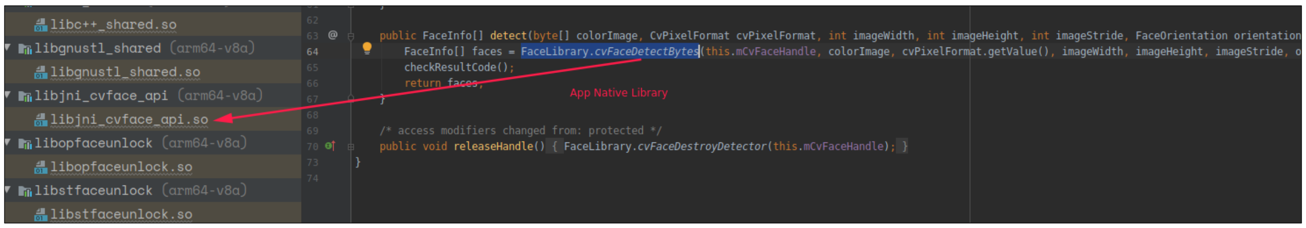
Figure 9. OPFaceUnlock Flow

We used the `onExtractFeatureAndCompare` function as a starting point**.**

First it checks if the image captured contains a valid face. Below we have the function responsible for detecting the face. We have something interesting here, all the analysis is done using an external native library.



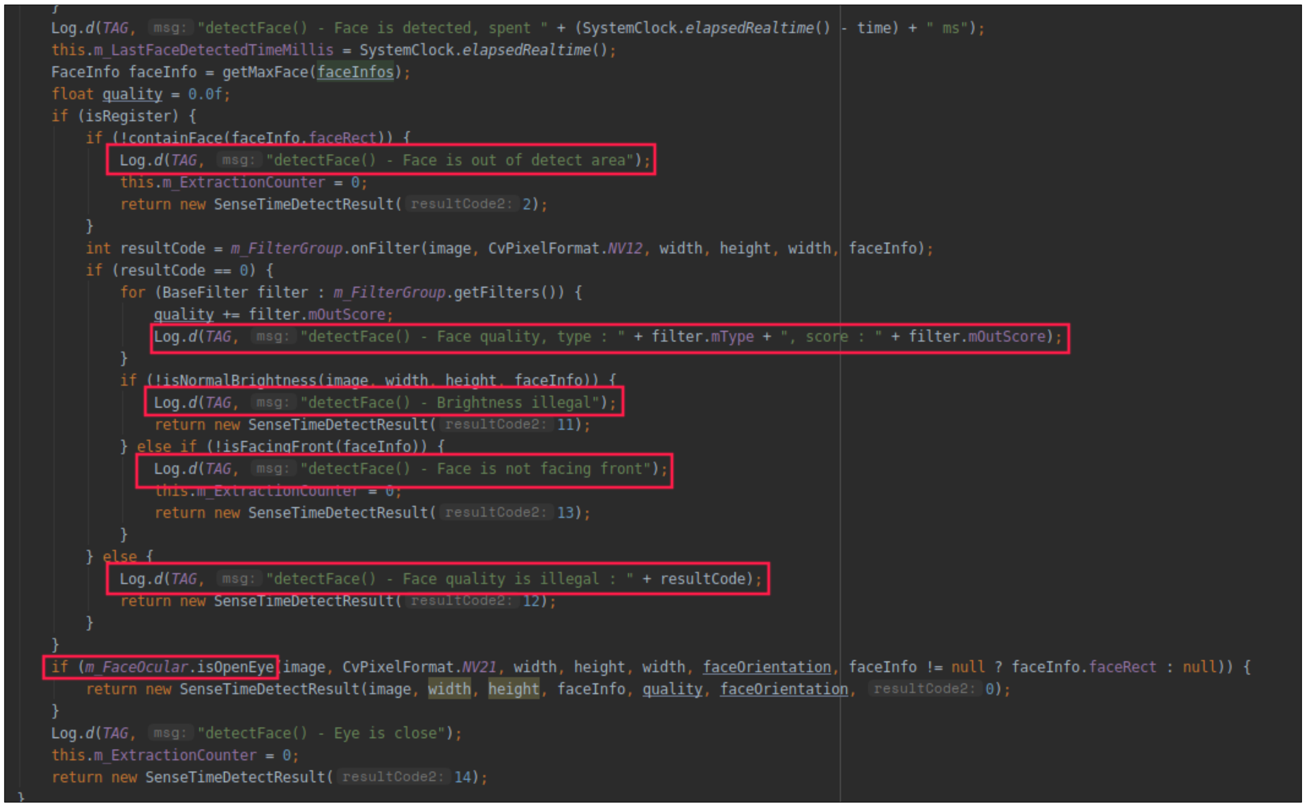
```
private SenseTimeDetectResult detectFace(byte[] image, int width, int height, int format, int orientation, boolean isRegister) {
    FaceInfo[] faceInfos;
    if (!isNv21Format(format)) {
        throw new RuntimeException("Unsupported image format : " + format + ", only NV21 is supported");
    }
    FaceOrientation faceOrientation = getFaceOrientation(orientation);
    long time = SystemClock.elapsedRealtime();
    if (isRegister) {
        faceInfos = m_FaceDetect.detect(image, CvPixelFormat.NV21, width, height, width, faceOrientation);
    } else {
        if (this.m_CurrentFaceDetect == m_FaceDetect640) {
            Log.d(TAG,  msg: "detectFace() - Use 640w to detect face");
        }
        faceInfos = this.m_CurrentFaceDetect.detect(image, CvPixelFormat.NV21, width, height, width, faceOrientation);
        if (faceInfos == null || faceInfos.length < 1) {
            Log.d(TAG,  msg: "detectFace() - Cannot detect face by default face orientation, try other orientations");
            FaceOrientation[] faceOrientationArr = FACE_ORIENTATIONS_CANDIDATES;
            int length = faceOrientationArr.length;
            int i = 0;
            while (true) {
                if (i >= length) {
                    break;
                }
                FaceOrientation fo = faceOrientationArr[i];
                if (fo != faceOrientation && (faceInfos = this.m_CurrentFaceDetect.detect(image, CvPixelFormat.NV21, width, height, width, fo)) != null && faceInfos.length > 0) {
                    faceOrientation = fo;
                    break;
                }
                i++;
            }
        }
    }
```



```
libc++_shared.so
▼ libgnustl_shared (arm64-v8a)
    libgnustl_shared.so
▼ libjni_cvface_api (arm64-v8a)
    libjni_cvface_api.so
▼ libopfaceunlock (arm64-v8a)
    libopfaceunlock.so
▼ libstfaceunlock (arm64-v8a)
    libstfaceunlock.so
```

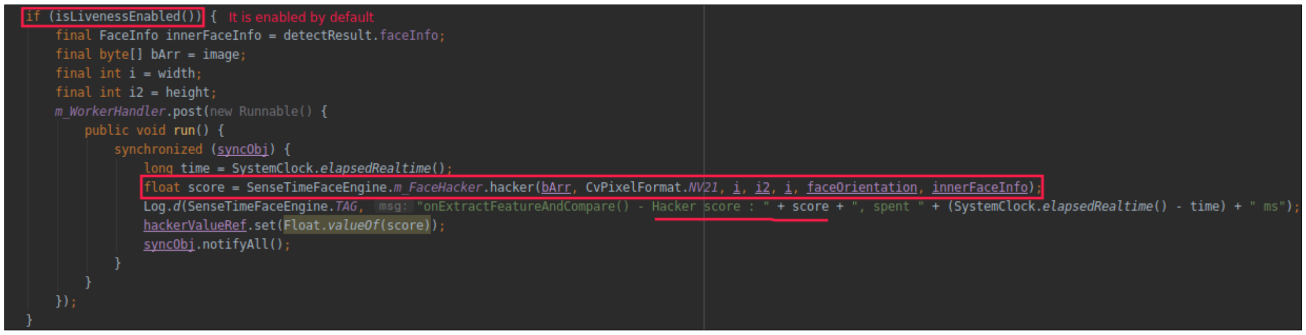
```
    public FaceInfo[] detect(byte[] colorImage, CvPixelFormat cvPixelFormat, int imageWidth, int imageHeight, int imageStride, FaceOrientation orientation
        FaceInfo[] faces = FaceLibrary.cvFaceDetectBytes(this.mCvFaceHandle, colorImage, cvPixelFormat.getValue(), imageWidth, imageHeight, imageStride, o
        checkResultCode();
        return faces;
    }

    /* access modifiers changed from: protected */
    public void releaseHandle() { FaceLibrary.cvFaceDestroyDetector(this.mCvFaceHandle); }
}
```

Continuing to look at the function, we can spot the basis for face detection (full face in the image, quality, brightness, front facing, and opened eyes):



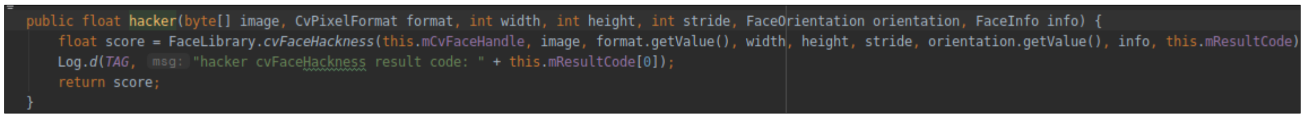
```
    Log.d(TAG,  msg: "detectFace() - Face is detected, spent " + (SystemClock.elapsedRealtime() - time) + " ms");
    this.m_LastFaceDetectedTimeMillis = SystemClock.elapsedRealtime();
    FaceInfo faceInfo = getMaxFace(faceInfos);
    float quality = 0.0f;
    if (isRegister) {
        if (!containFace(faceInfo.faceRect)) {
            Log.d(TAG,  msg: "detectFace() - Face is out of detect area");
            this.m_ExtractionCounter = 0;
            return new SenseTimeDetectResult( resultCode2: 2);
        }
        int resultCode = m_FilterGroup.onFilter(image, CvPixelFormat.NV12, width, height, width, faceInfo);
        if (resultCode == 0) {
            for (BaseFilter filter : m_FilterGroup.getFilters()) {
                quality += filter.mOutScore;
                Log.d(TAG,  msg: "detectFace() - Face quality, type : " + filter.mType + ", score : " + filter.mOutScore);
            }
            if (!isNormalBrightness(image, width, height, faceInfo)) {
                Log.d(TAG,  msg: "detectFace() - Brightness illegal");
                return new SenseTimeDetectResult( resultCode2: 11);
            } else if (!isFacingFront(faceInfo)) {
                Log.d(TAG,  msg: "detectFace() - Face is not facing front");
                this.m_ExtractionCounter = 0;
                return new SenseTimeDetectResult( resultCode2: 13);
            }
        } else {
            Log.d(TAG,  msg: "detectFace() - Face quality is illegal : " + resultCode);
            return new SenseTimeDetectResult( resultCode2: 12);
        }
    }
    if (m_FaceOcular.isOpenEye(image, CvPixelFormat.NV21, width, height, width, faceOrientation, faceInfo != null ? faceInfo.faceRect : null)) {
        return new SenseTimeDetectResult(image, width, height, faceInfo, quality, faceOrientation,  resultCode2: 0);
    }
    Log.d(TAG,  msg: "detectFace() - Eye is close");
    this.m_ExtractionCounter = 0;
    return new SenseTimeDetectResult( resultCode2: 14);
}
```

Once it verifies a valid face is in the image, the next step is to validate that the detected face is a real face and not a printed image, as we used in our testing. We can see that liveliness is measured with a score value. Taking a closer look:



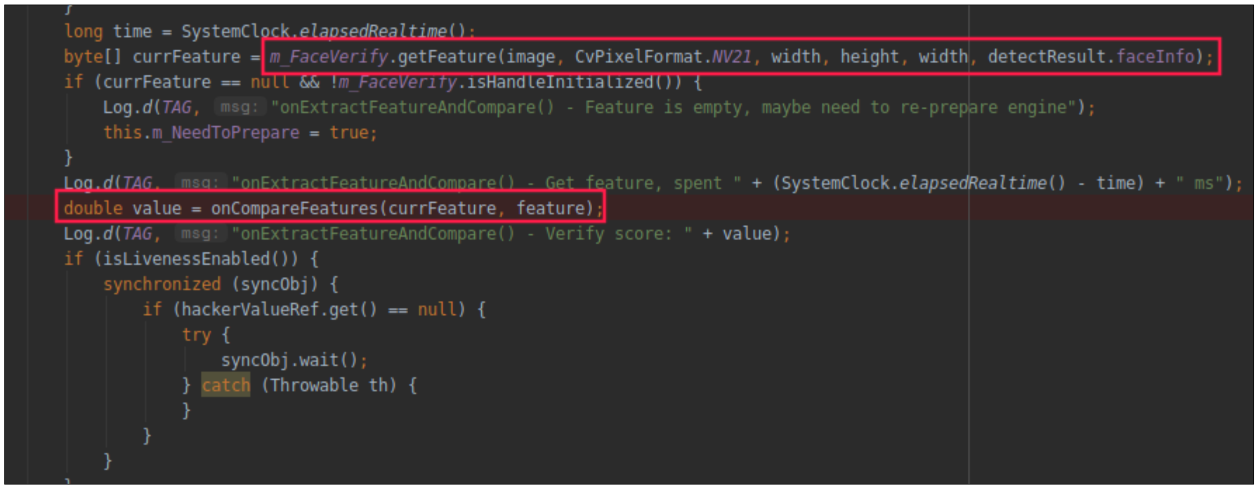
```
if (isLivenessEnabled()) {  It is enabled by default
    final FaceInfo innerFaceInfo = detectResult.faceInfo;
    final byte[] bArr = image;
    final int i = width;
    final int i2 = height;
    m_WorkerHandler.post(new Runnable() {
        public void run() {
            synchronized (syncObj) {
                long time = SystemClock.elapsedRealtime();
                float score = SenseTimeFaceEngine.m_FaceHacker.hacker(bArr, CvPixelFormat.NV21, i, i2, i, faceOrientation, innerFaceInfo);
                Log.d(SenseTimeFaceEngine.TAG,  msg: "onExtractFeatureAndCompare() - Hacker score : " + score + ", spent " + (SystemClock.elapsedRealtime() - time) + " ms");
                hackerValueRef.set(Float.valueOf(score));
                syncObj.notifyAll();
            }
        }
    });
}
```

Again, the native library is used to determinate this value.



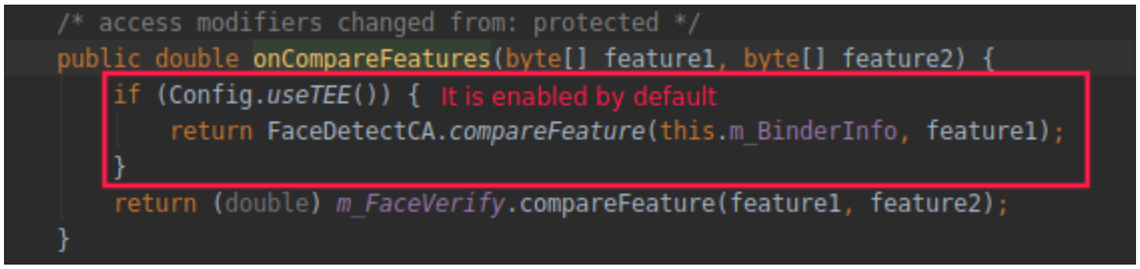
```
public float hacker(byte[] image, CvPixelFormat format, int width, int height, int stride, FaceOrientation orientation, FaceInfo info) {
    float score = FaceLibrary.cvFaceHackness(this.mCvFaceHandle, image, format.getValue(), width, height, stride, orientation.getValue(), info, this.mResultCode)
    Log.d(TAG,  msg: "hacker cvFaceHackness result code: " + this.mResultCode[0]);
    return score;
}
```

Then the function proceeds to the most important part, extracting image features and comparing them to enrolled subject's facial data.



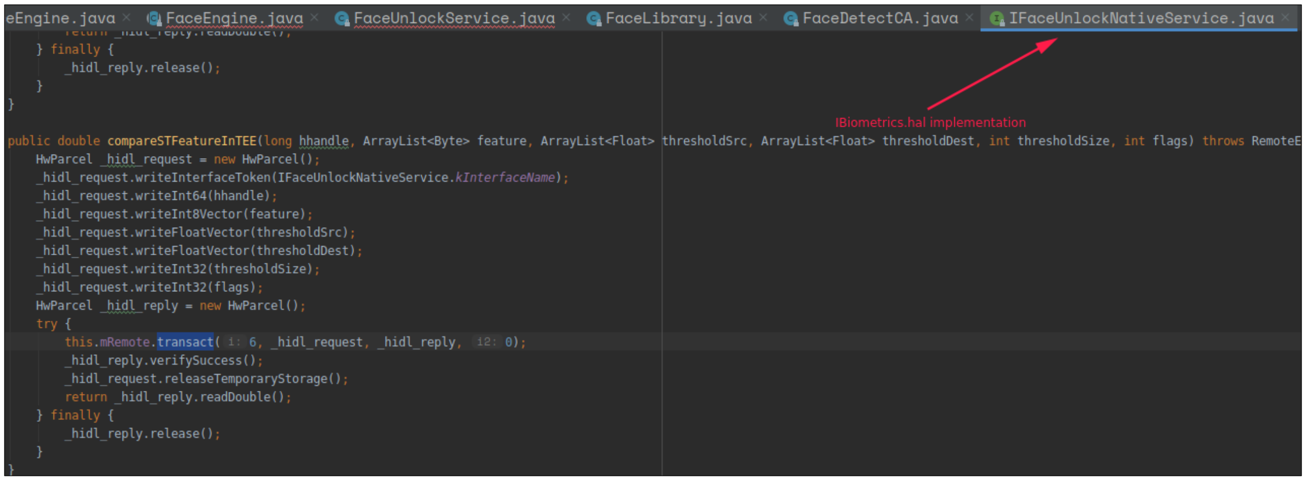
```
    long time = SystemClock.elapsedRealtime();
    byte[] currFeature = m_FaceVerify.getFeature(image, CvPixelFormat.NV21, width, height, width, detectResult.faceInfo);
    if (currFeature == null && !m_FaceVerify.isHandleInitialized()) {
        Log.d(TAG, msg: "onExtractFeatureAndCompare() - Feature is empty, maybe need to re-prepare engine");
        this.m_NeedToPrepare = true;
    }
    Log.d(TAG, msg: "onExtractFeatureAndCompare() - Get feature, spent " + (SystemClock.elapsedRealtime() - time) + " ms");
    double value = onCompareFeatures(currFeature, feature);
    Log.d(TAG, msg: "onExtractFeatureAndCompare() - Verify score: " + value);
    if (isLivenessEnabled()) {
        synchronized (syncObj) {
            if (hackerValueRef.get() == null) {
                try {
                    syncObj.wait();
                } catch (Throwable th) {
                }
            }
        }
```

To extract image features, the function also relies on the native library; however, when comparing features, it calls the TEE for the first time to request the match score.



```
/* access modifiers changed from: protected */
public double onCompareFeatures(byte[] feature1, byte[] feature2) {
    if (Config.useTEE()) {  It is enabled by default
        return FaceDetectCA.compareFeature(this.m_BinderInfo, feature1);
    }
    return (double) m_FaceVerify.compareFeature(feature1, feature2);
}
```

Continuing our analysis, we find `IFaceUnlockNativeService` **(BiometricManager)** which is the interface that will talk to the TEE hardware environment.



```
    } finally {
        _hidl_reply.release();
    }
}

public double compareSTFeatureInTEE(long hhandle, ArrayList<Byte> feature, ArrayList<Float> thresholdSrc, ArrayList<Float> thresholdDest, int thresholdSize, int flags) throws RemoteE
    HwParcel _hidl_request = new HwParcel();
    _hidl_request.writeInterfaceToken(IFaceUnlockNativeService.kInterfaceName);
    _hidl_request.writeInt64(hhandle);
    _hidl_request.writeInt8Vector(feature);
    _hidl_request.writeFloatVector(thresholdSrc);
    _hidl_request.writeFloatVector(thresholdDest);
    _hidl_request.writeInt32(thresholdSize);
    _hidl_request.writeInt32(flags);
    HwParcel _hidl_reply = new HwParcel();
    try {
        this.mRemote.transact( i: 6, _hidl_request, _hidl_reply,  i2: 0);
        _hidl_reply.verifySuccess();
        _hidl_request.releaseTemporaryStorage();
        return _hidl_reply.readDouble();
    } finally {
        _hidl_reply.release();
    }
}
```

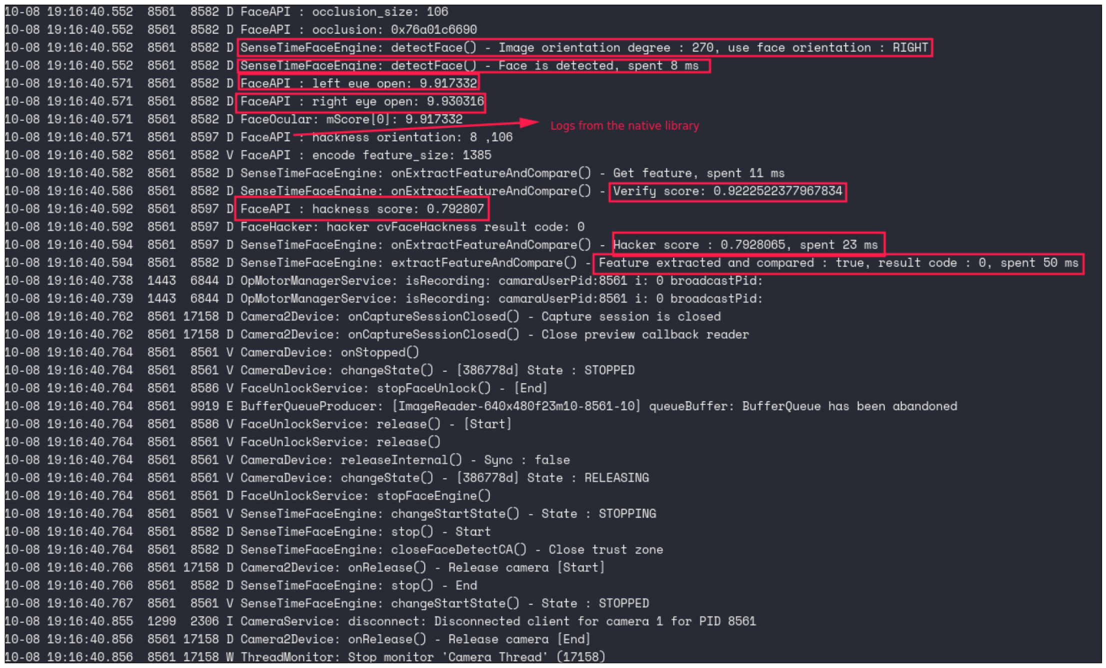
Once the match score is received, if it is a positive match and `hackerValue` is less than 0.95, the check process is complete and the phone will unlock.

Additional Observations

We observed that the code contains numerous log calls. This makes our task easier by disclosing useful information in real time while the phone is evaluating a face.

```
adb logcat | grep "$(adb shell ps | grep com.oneplus.faceunlock |
awk '{print $2}')"
```

Output:



```
10-08 19:16:40.552  8561  8582 D FaceAPI : occlusion_size: 106
10-08 19:16:40.552  8561  8582 D FaceAPI : occlusion: 0x76a01c6690
10-08 19:16:40.552  8561  8582 D SenseTimeFaceEngine: detectFace() - Image orientation degree : 270, use face orientation : RIGHT
10-08 19:16:40.552  8561  8582 D SenseTimeFaceEngine: detectFace() - Face is detected, spent 8 ms
10-08 19:16:40.571  8561  8582 D FaceAPI : left eye open: 9.917332
10-08 19:16:40.571  8561  8582 D FaceAPI : right eye open: 9.930316
10-08 19:16:40.571  8561  8582 D FaceOcular: mScore[0]: 9.917332
10-08 19:16:40.571  8561  8597 D FaceAPI : hackness orientation: 8 ,106
10-08 19:16:40.582  8561  8582 V FaceAPI : encode feature_size: 1385
10-08 19:16:40.582  8561  8582 D SenseTimeFaceEngine: onExtractFeatureAndCompare() - Get feature, spent 11 ms
10-08 19:16:40.586  8561  8582 D SenseTimeFaceEngine: onExtractFeatureAndCompare() - Verify score: 0.9222522377967834
10-08 19:16:40.592  8561  8597 D FaceAPI : hackness score: 0.792807
10-08 19:16:40.592  8561  8597 D FaceHacker: hacker cvFaceHackness result code: 0
10-08 19:16:40.594  8561  8597 D SenseTimeFaceEngine: onExtractFeatureAndCompare() - Hacker score : 0.7928065, spent 23 ms
10-08 19:16:40.594  8561  8582 D SenseTimeFaceEngine: extractFeatureAndCompare() - Feature extracted and compared : true, result code : 0, spent 50 ms
10-08 19:16:40.738  1443  6844 D OpMotorManagerService: isRecording: camaraUserPid:8561 i: 0 broadcastPid:
10-08 19:16:40.739  1443  6844 D OpMotorManagerService: isRecording: camaraUserPid:8561 i: 0 broadcastPid:
10-08 19:16:40.762  8561 17158 D Camera2Device: onCaptureSessionClosed() - Capture session is closed
10-08 19:16:40.762  8561 17158 D Camera2Device: onCaptureSessionClosed() - Close preview callback reader
10-08 19:16:40.764  8561  8561 V CameraDevice: onStopped()
10-08 19:16:40.764  8561  8561 V CameraDevice: changeState() - [386778d] State : STOPPED
10-08 19:16:40.764  8561  8586 V FaceUnlockService: stopFaceUnlock() - [End]
10-08 19:16:40.764  8561  9919 E BufferQueueProducer: [ImageReader-640x480f23m10-8561-10] queueBuffer: BufferQueue has been abandoned
10-08 19:16:40.764  8561  8586 V FaceUnlockService: release() - [Start]
10-08 19:16:40.764  8561  8561 V FaceUnlockService: release()
10-08 19:16:40.764  8561  8561 V CameraDevice: releaseInternal() - Sync : false
10-08 19:16:40.764  8561  8561 V CameraDevice: changeState() - [386778d] State : RELEASING
10-08 19:16:40.764  8561  8561 D FaceUnlockService: stopFaceEngine()
10-08 19:16:40.764  8561  8561 V SenseTimeFaceEngine: changeStartState() - State : STOPPING
10-08 19:16:40.764  8561  8582 D SenseTimeFaceEngine: stop() - Start
10-08 19:16:40.764  8561  8582 D SenseTimeFaceEngine: closeFaceDetectCA() - Close trust zone
10-08 19:16:40.766  8561 17158 D Camera2Device: onRelease() - Release camera [Start]
10-08 19:16:40.766  8561  8582 D SenseTimeFaceEngine: stop() - End
10-08 19:16:40.767  8561  8561 V SenseTimeFaceEngine: changeStartState() - State : STOPPED
10-08 19:16:40.855  1299  2306 I CameraService: disconnect: Disconnected client for camera 1 for PID 8561
10-08 19:16:40.856  8561 17158 D Camera2Device: onRelease() - Release camera [End]
10-08 19:16:40.856  8561 17158 W ThreadMonitor: Stop monitor 'Camera Thread' (17158)
```

Our next step will be to use same approach as during black-box testing, but to leverage the reference information in the logs provided by the **App** to further improve the probability of unlocking devices.

Conclusions

The use of facial recognition systems has become pervasive on mobile phones and is making inroads in other sectors as way to authenticate the end user of a system. These technologies rely on models created from an image or facial scan, selecting specific features that will be checked in a live environment against the actual user or an attacker. The algorithms need be accurate enough to detect a spoof attempt but flexible enough to make the technology useful under different lighting conditions and given normal physical changes in the legitimate users.

As has been shown in this paper, the technologies behind facial recognition have room for improvement. A number of techniques have been used to successfully bypass the algorithms and there is plenty of room for additional creative attack methods.

As a general recommendation, no authentication technology should be released to production without a thorough security review to help tune the system and understand the potential risk.

About Gabriel Gonzalez
As the Director of Hardware Security for IOActive and a vulnerability researcher, Gabriel has completed hundreds of projects in the areas of reverse engineering, code review, and integrated hardware and software penetration testing. His primary focus is hardware and embedded system technologies,with specialized expertise in low-level attack vectors, and fault-injection and side-channel analysis. Gabriel has engaged his research efforts in the fields of automotive, avionics, smart grid/utilities, SATCOM, banking devices, IOT, and many others. He has also been key in creating the Biometric Testing Lab and has heavily contributed in developing IOActive's Biometric Methodology.

Gabriel is actively engaged with the security research community and has presented numerous original cybersecurity research projects at major conferences such as Black Hat Europe and RootedCON.

About Alejo Moles
As a Security Consultant for IOActive and vulnerability researcher, Alejo has expertise in reverse engineering, penetration testing, and software development. Having engaged in numerous projects, Alejo is knowledgeable in multiple technologies including mobile OS, infrastructure, and embedded source code reviews. With a master's degree in cybersecurity, Alejo's passion for research, together with analytical and organization skills, have enabled him to expand his knowledge base while serving a growing client base.

Keyword
Face Authentication, Face Recognition, Face Unlock, Biometric Authentication