Logi's Circle of Security

A Penetration Test Report of Logitech Circle

Amar Lakshya, Calvin Menezes, Pier Giuseppe Rotondo, Primanio Sutiharjoyo

School of Computer Science, University of Birmingham

1 Introduction

The Logitech Circle device was chosen to conduct a thorough penetration test. The test involved an initial information gathering phase, several active reconnaissance measures to extract the software architecture used and resulted in establishing that the device in question appears to have been built with good security measures in place and can be regarded as secure against many attacks simply by using techniques like secure defaults and defence-in-depth.

2 Scope

There were three major hurdles that defined the scope of the penetration test:

- Restriction on physical access to the insides of the device.
- Lack of any Bluetooth sniffing hardware to monitor Bluetooth communication between the device and the Mobile Application.
- Restriction on getting an Oauth2.0 security token from Logitech due the coronavirus pandemic.

Therefore, the test focused on the following parts:

- Device capabilities
- Device architecture
- Communication between the device and the application
- Communication between the device and the server
- Communication between the application and the server

3 Investigation

3.1 Initial information gathering

A lot of information about the device capabilities was extracted through tracking the FCC ID `VR0005`. It was further discovered that the device could function in the following wireless communication frequency bands:

Bluetooth Bands

- 2.402 - 2.48 GHz

Wi-Fi Bands

- 2.4 GHz (2.412 - 2.462 GHz)
- 5 GHz (5.18 - 5.24 GHz)
- 5 GHz Super High Frequency (5.18 - 5.24 GHz)

A few other observations were also made:

- The device's MAC Address is `44:73:d6:01:90:b9.`
- The official developer API for the device was found at https://developers.logitech.com/circle
- A python wrapper for the API was also found at: https://github.com/evanjd/python-logi-circle

3.2 Device architecture analysis

The device setup contains the following parts:

- The Logitech Circle Camera
- A Mobile Application (Android)
- A Logitech Account (Server)

These parts interact with each other in two different stages to create an architecture that emphasizes default, secure communication only between the required parts through a layered defence-in-depth approach.

In order of operation, these two stages are:

- Connection Stage
- Streaming/Recording Stage

3.2.1 Connection stage

The connection stage as seen below focuses on:

1. Establishing a bluetooth connection between the device and the mobile application

2. Sending the wireless network configuration details from the application to the device.

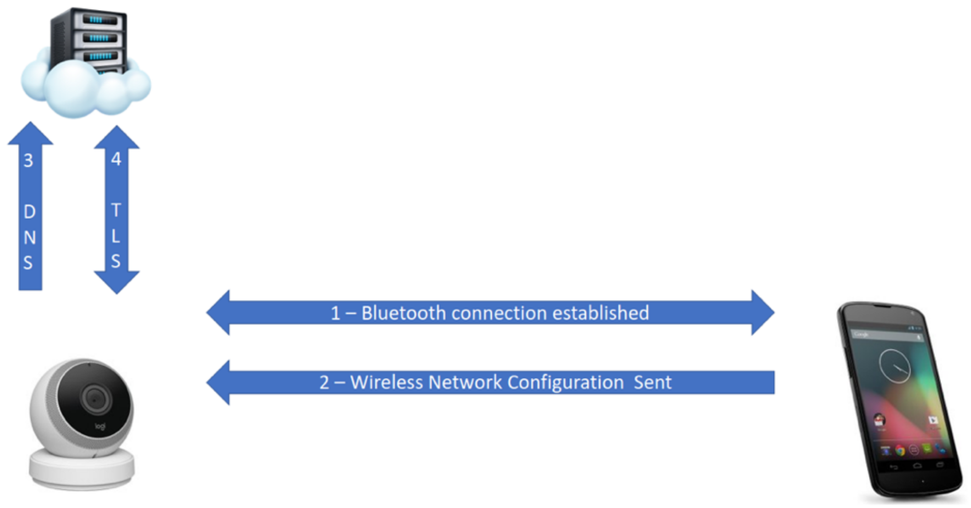
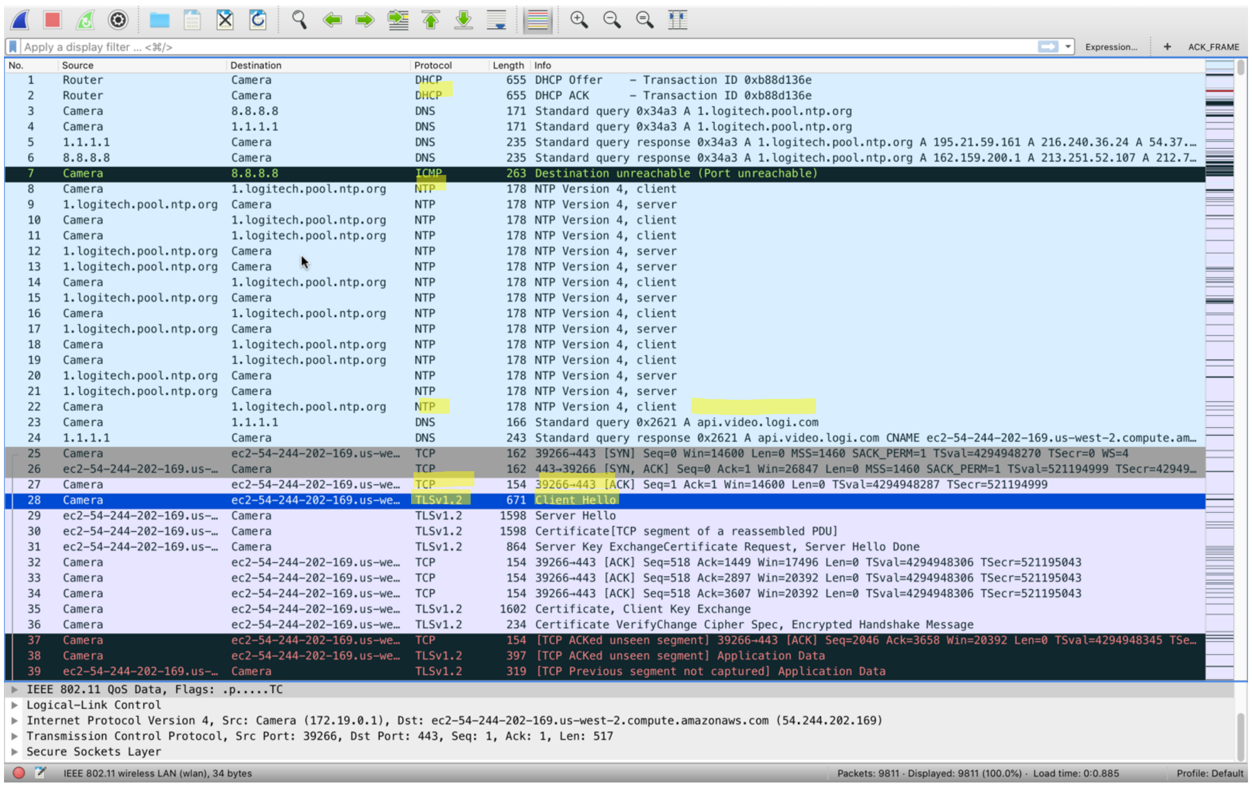

3 DNS
4 TLS
1 – Bluetooth connection established
2 – Wireless Network Configuration Sent

Figure 1 Connection stage of the device

3.2.2 Streaming/recording stage

When the camera starts up for the first time, it is configured using the Logi Circle mobile app (over Bluetooth) to connect to a particular Wi-Fi network and link all the footage captured to the provided Logi Circle account in the cloud. From this point onward, the camera stays connected to the intended wireless network and automatically streams to a cloud server (which is hosted on Amazon Web Services) as long as it is powered on with its in-built battery or plugged in with the AC adapter. This configuration is saved by the camera so even if it is restarted, it connects to the known network and resumes streaming video to the cloud server right away. The screenshot below shows a Wireshark trace of the camera on boot up. There is no transfer of data directly from the phone to camera and vice versa through the local network.



```
Apply a display filter ... <⌘/>                                                     Expression...  +  ACK_FRAME
No. Source                  Destination             Protocol  Length Info
 1  Router                  Camera                  DHCP       655 DHCP Offer    - Transaction ID 0xb88d136e
 2  Router                  Camera                  DHCP       655 DHCP ACK      - Transaction ID 0xb88d136e
 3  Camera                  8.8.8.8                 DNS        171 Standard query 0x34a3 A 1.logitech.pool.ntp.org
 4  Camera                  1.1.1.1                 DNS        171 Standard query 0x34a3 A 1.logitech.pool.ntp.org
 5  1.1.1.1                 Camera                  DNS        235 Standard query response 0x34a3 A 1.logitech.pool.ntp.org A 195.21.59.161 A 216.240.36.24 A 54.37.…
 6  8.8.8.8                 Camera                  DNS        235 Standard query response 0x34a3 A 1.logitech.pool.ntp.org A 162.159.200.1 A 213.251.52.107 A 212.7…
 7  Camera                  8.8.8.8                 ICMP       263 Destination unreachable (Port unreachable)
 8  Camera                  1.logitech.pool.ntp.org NTP        178 NTP Version 4, client
 9  1.logitech.pool.ntp.org Camera                  NTP        178 NTP Version 4, server
10  Camera                  1.logitech.pool.ntp.org NTP        178 NTP Version 4, client
11  Camera                  1.logitech.pool.ntp.org NTP        178 NTP Version 4, client
12  1.logitech.pool.ntp.org Camera                  NTP        178 NTP Version 4, server
13  1.logitech.pool.ntp.org Camera                  NTP        178 NTP Version 4, server
14  Camera                  1.logitech.pool.ntp.org NTP        178 NTP Version 4, client
15  1.logitech.pool.ntp.org Camera                  NTP        178 NTP Version 4, server
16  Camera                  1.logitech.pool.ntp.org NTP        178 NTP Version 4, client
17  1.logitech.pool.ntp.org Camera                  NTP        178 NTP Version 4, server
18  Camera                  1.logitech.pool.ntp.org NTP        178 NTP Version 4, client
19  Camera                  1.logitech.pool.ntp.org NTP        178 NTP Version 4, client
20  1.logitech.pool.ntp.org Camera                  NTP        178 NTP Version 4, server
21  1.logitech.pool.ntp.org Camera                  NTP        178 NTP Version 4, server
22  Camera                  1.logitech.pool.ntp.org NTP        178 NTP Version 4, client
23  Camera                  1.1.1.1                 DNS        166 Standard query 0x2621 A api.video.logi.com
24  1.1.1.1                 Camera                  DNS        243 Standard query response 0x2621 A api.video.logi.com CNAME ec2-54-244-202-169.us-west-2.compute.am…
25  Camera                  ec2-54-244-202-169.us-we… TCP      162 39266→443 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=4294948270 TSecr=0 WS=4
26  ec2-54-244-202-169.us-… Camera                  TCP        162 443→39266 [SYN, ACK] Seq=0 Ack=1 Win=26847 Len=0 MSS=1460 SACK_PERM=1 TSval=521194999 TSecr=42949…
27  Camera                  ec2-54-244-202-169.us-we… TCP      154 39266→443 [ACK] Seq=1 Ack=1 Win=14600 Len=0 TSval=4294948287 TSecr=521194999
28  Camera                  ec2-54-244-202-169.us-we… TLSv1.2  671 Client Hello
29  ec2-54-244-202-169.us-… Camera                  TLSv1.2   1598 Server Hello
30  ec2-54-244-202-169.us-… Camera                  TLSv1.2   1598 Certificate[TCP segment of a reassembled PDU]
31  ec2-54-244-202-169.us-… Camera                  TLSv1.2    864 Server Key ExchangeCertificate Request, Server Hello Done
32  Camera                  ec2-54-244-202-169.us-we… TCP      154 39266→443 [ACK] Seq=518 Ack=1449 Win=17496 Len=0 TSval=4294948306 TSecr=521195043
33  Camera                  ec2-54-244-202-169.us-we… TCP      154 39266→443 [ACK] Seq=518 Ack=2897 Win=20392 Len=0 TSval=4294948306 TSecr=521195043
34  Camera                  ec2-54-244-202-169.us-we… TCP      154 39266→443 [ACK] Seq=518 Ack=3607 Win=20392 Len=0 TSval=4294948306 TSecr=521195043
35  Camera                  ec2-54-244-202-169.us-we… TLSv1.2 1602 Certificate, Client Key Exchange
36  Camera                  ec2-54-244-202-169.us-we… TLSv1.2  234 Certificate VerifyChange Cipher Spec, Encrypted Handshake Message
37  Camera                  ec2-54-244-202-169.us-we… TCP      154 [TCP ACKed unseen segment] 39266→443 [ACK] Seq=2046 Ack=3658 Win=20392 Len=0 TSval=4294948345 TSe…
38  Camera                  ec2-54-244-202-169.us-we… TLSv1.2  397 [TCP ACKed unseen segment] Application Data
39  ec2-54-244-202-169.us-… Camera                  TLSv1.2    319 [TCP Previous segment not captured] Application Data
▶ IEEE 802.11 QoS Data, Flags: .p.....TC
▶ Logical-Link Control
▶ Internet Protocol Version 4, Src: Camera (172.19.0.1), Dst: ec2-54-244-202-169.us-west-2.compute.amazonaws.com (54.244.202.169)
▶ Transmission Control Protocol, Src Port: 39266, Dst Port: 443, Seq: 1, Ack: 1, Len: 517
▶ Secure Sockets Layer
IEEE 802.11 wireless LAN (wlan), 34 bytes            Packets: 9811 · Displayed: 9811 (100.0%) · Load time: 0:0.885    Profile: Default
```

Figure 2 Wireshark capture shows data flow between camera and the internet

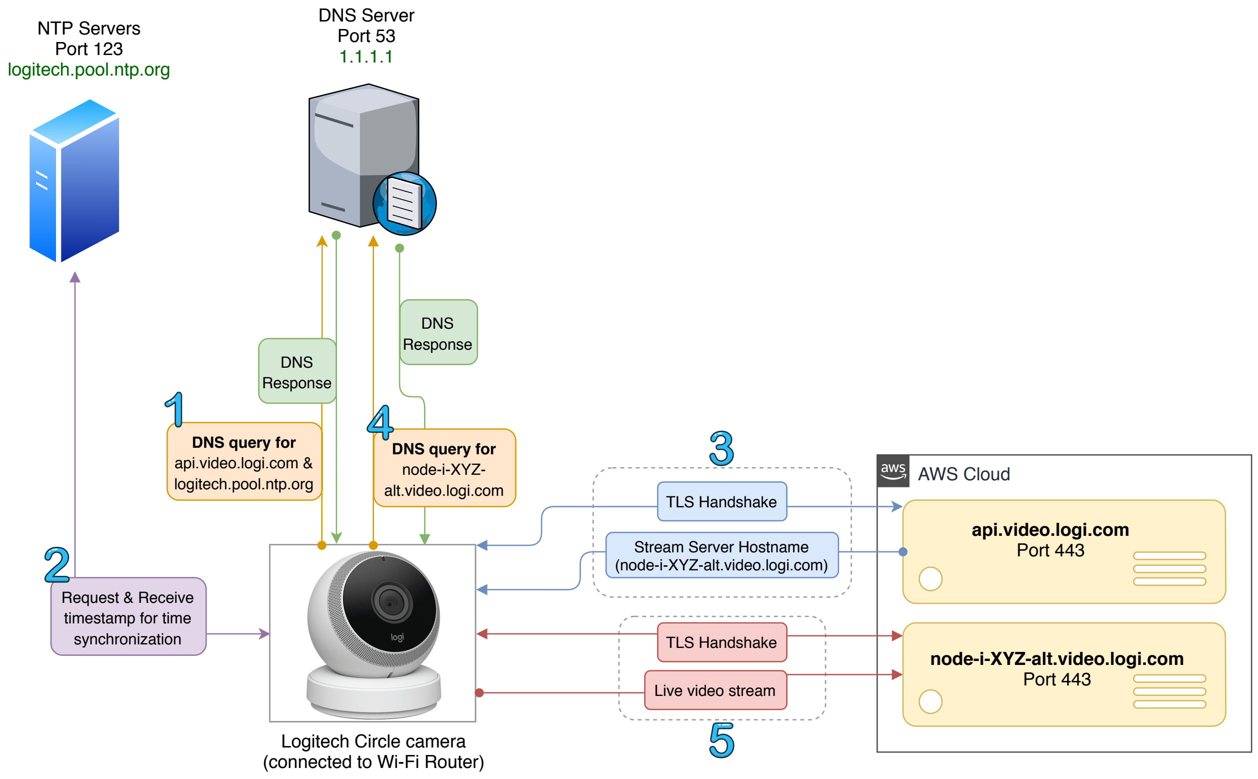
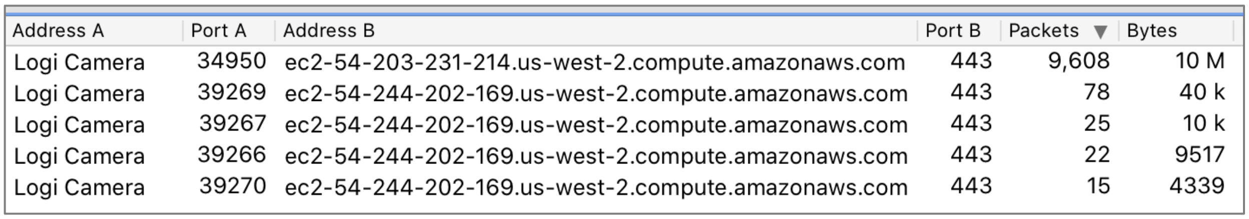


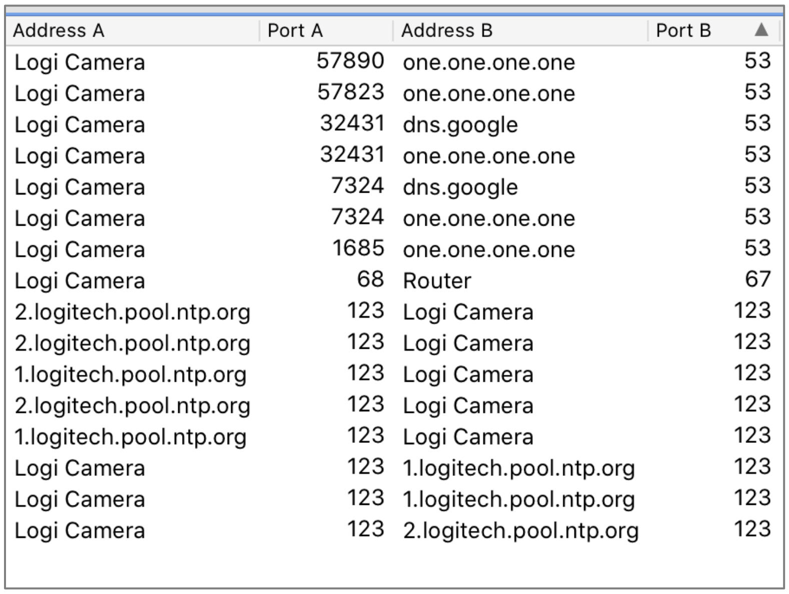
Figure 3 Sequence of steps once a connection to a wireless network is established

1. The camera connects to the Wi-Fi network and is assigned an IP by the router. Once it gains internet access, it performs two DNS queries; one for `api.video.logi.com` and another for `logitech.pool.ntp.org`. As the hostnames suggest, the camera tries to connect to the Logitech cloud server API and also sync its time with a pool of NTP servers. It receives a response from the DNS server (in our case, this was `1.1.1.1` or `8.8.8.8`), fetches their IPs and attempts to connect to them.
2. The camera sends NTP request packets on port 123 to one or many of the NTP server IPs it received from logitech.pool.ntp.org. It receives a response from these servers and maintains time synchronization.
3. The camera then starts an encrypted connection from a random TCP port by initiating a TLS handshake to `api.video.logi.com` on port 443 of the server. It sends a list of cipher suites it supports which is very similar to what the mobile app uses to connect to the cloud as well. All network traffic from this point onward is encrypted. We are certain that `api.video.logi.com` is only an authentication or configuration server which links the device and its camera footage to the Logitech account. Video is not streamed to this server, as we found out from our network captures that it connects to another server shortly after only for this purpose which leads us to the next step.
4. `api.video.logi.com` sends the camera the hostname of another server which it can stream to. The camera makes a DNS request on port 53 for that hostname and receives its corresponding IP address. We also noticed the hostname of this streaming server always followed the pattern "`node-i-X-alt.video.logi.com`", the X being a random string of characters that it receives from the `api.video.logi.com`.
5. The camera starts another encrypted connection on TCP port 443 of the streaming server, similar to step 3 with the authentication server, by doing a TLS handshake. We now see bigger flow of encrypted packets between the camera and this server. The camera remains connected to both servers throughout the streaming period.



Address A	Port A	Address B	Port B	Packets ▼	Bytes
Logi Camera	34950	ec2-54-203-231-214.us-west-2.compute.amazonaws.com	443	9,608	10 M
Logi Camera	39269	ec2-54-244-202-169.us-west-2.compute.amazonaws.com	443	78	40 k
Logi Camera	39267	ec2-54-244-202-169.us-west-2.compute.amazonaws.com	443	25	10 k
Logi Camera	39266	ec2-54-244-202-169.us-west-2.compute.amazonaws.com	443	22	9517
Logi Camera	39270	ec2-54-244-202-169.us-west-2.compute.amazonaws.com	443	15	4339

Figure 4 All TCP conversations between the camera and cloud



Address A	Port A	Address B	Port B
Logi Camera	57890	one.one.one.one	53
Logi Camera	57823	one.one.one.one	53
Logi Camera	32431	dns.google	53
Logi Camera	32431	one.one.one.one	53
Logi Camera	7324	dns.google	53
Logi Camera	7324	one.one.one.one	53
Logi Camera	1685	one.one.one.one	53
Logi Camera	68	Router	67
2.logitech.pool.ntp.org	123	Logi Camera	123
2.logitech.pool.ntp.org	123	Logi Camera	123
1.logitech.pool.ntp.org	123	Logi Camera	123
2.logitech.pool.ntp.org	123	Logi Camera	123
1.logitech.pool.ntp.org	123	Logi Camera	123
Logi Camera	123	1.logitech.pool.ntp.org	123
Logi Camera	123	1.logitech.pool.ntp.org	123
Logi Camera	123	2.logitech.pool.ntp.org	123

Figure 5 All UDP conversations between the camera and internet

3.3 Android application analysis

The camera has a companion mobile app called Logi Circle which is available for Android and iOS devices.

For this instance, we analysed the Android version of the app. The official latest version of the app is available on the Google Play Store. Older versions of the app are also available on a third-party app source such as apkmirror (https://www.apkmirror.com) and apkpure (https://apkpure.com).

There are two modes the application functions in. It is related to how the device works as explained above. The two modes are:

Setup Mode

Used in initial camera setup. If the camera has not been set up yet, or no Logitech account yet, the user must enter this mode to configure the camera for the first time. The screenshots below show how the camera is set up in a sequence.

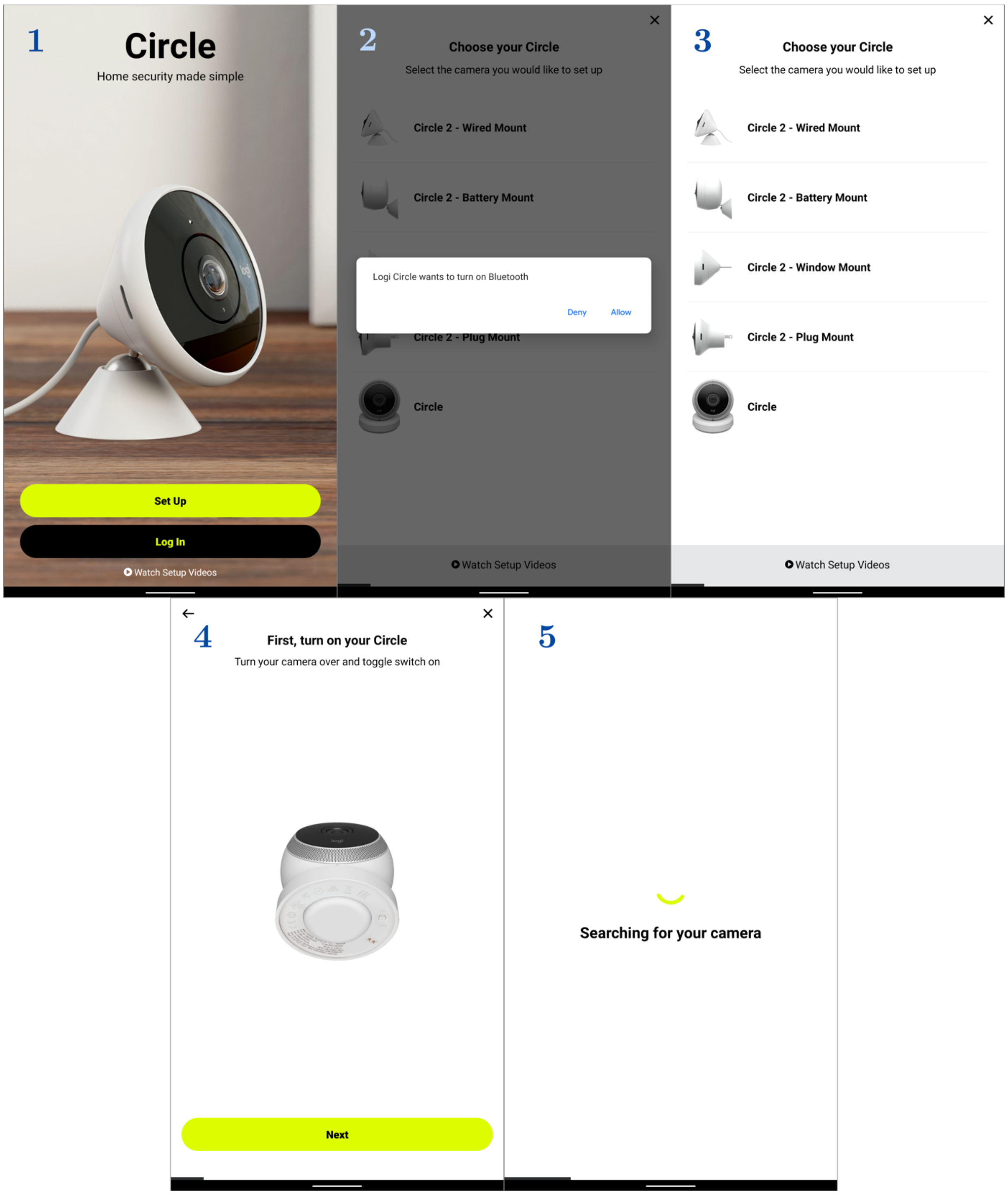
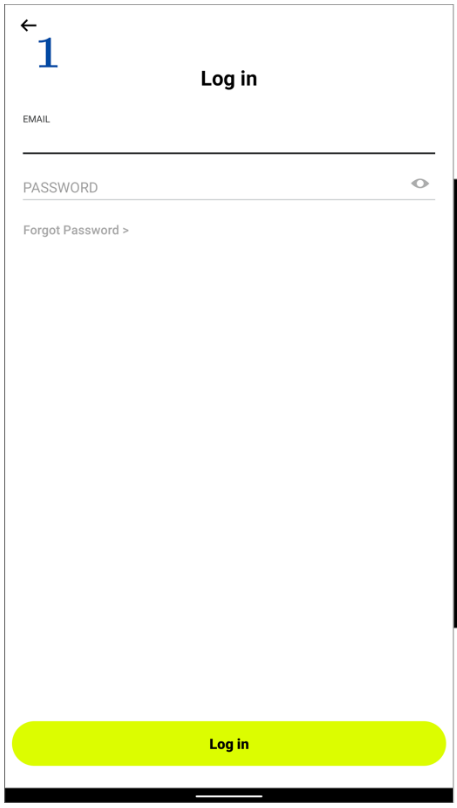
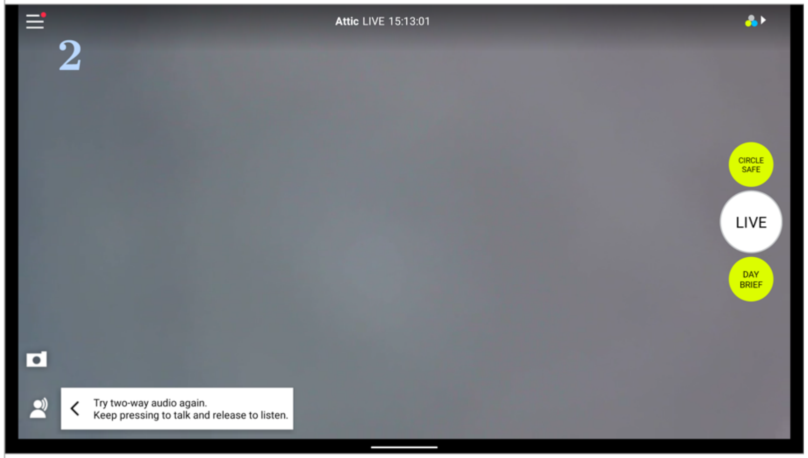


Figure 6 Series of in-app screenshots depicting setup with camera

Streaming Mode

Used if the camera has already been set up, or user is trying to access the camera remotely using the app.





Try two-way audio again.
Keep pressing to talk and release to listen.

Figure 7 Logitech account log-in followed by streaming from the cloud

After decompiling the APK file using JADX, the detailed flow of the application can be analysed. Although many of the original class, method, and variable names was not retrieved correctly, the general application workflow can be depicted as follows

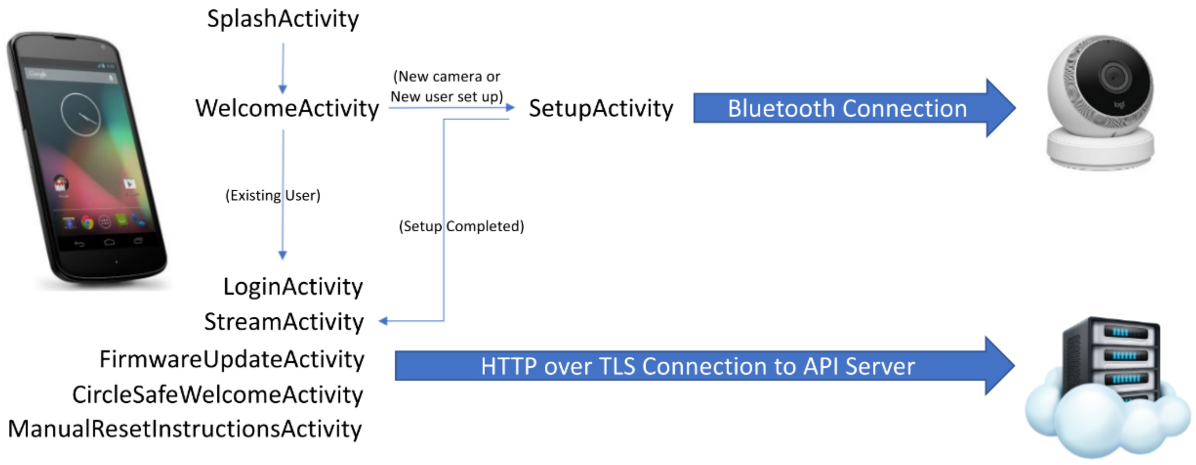


Figure 8 Android application workflow

Focusing on security aspect of the application, some specific keywords were used to look for interesting part of the application. Some of the keywords we used often are: "`UID`", "`user`", "`username`", "`password`", "`crypto/crypt`", "`encrypt`", "`decrypt`", "`cipher`", "`certificate`", "`RSA`", "`AES`", "`SHA`", and "`MD5`".

Relevant code snippets were also found in the application:

1. Password is not logged by the application

   - In the `AccountServiceLog` class, the class constructor method is overridden to exclude any password related string

```
public class AccountServiceLog extends TokenServiceLog {
    private static String replaceOldPasswordTo = "\"oldPassword\":";
    private static String replacePasswordTo = "\"password\":";
    private Pattern excludeOldPassword = Pattern.compile("\"oldPassword\"[ :\"]{1,4}(.*?)(\\\\\\\\\"|[^\\\\]\")");
    private Pattern excludePassword = Pattern.compile("\"password\"[ :\"]{1,4}(.*?)(\\\\\\\\\"|[^\\\\]\")");

    public AccountServiceLog(String str) {
        super(str);
        addExcludingRule(new TokenServiceLog.ExcludingRule(this.excludePassword, replacePasswordTo));
        addExcludingRule(new TokenServiceLog.ExcludingRule(this.excludeOldPassword, replaceOldPasswordTo));
    }
}
```

2. Root Certificate stored in `res/raw/rootcert.pem`
3. Communication of account data from and to the camera are encrypted using RSA/ECB/PKCS1Padding. The encryption utilities class (C7244z) is used in several other activities throughout the app.

Key-pair generator

```
public void m26914b() {
        KeyStore keyStore;
        try {
            keyStore = KeyStore.getInstance("AndroidKeyStore");
        } catch (KeyStoreException e) {
            try {
                keyStore = KeyStore.getInstance(KeyStore.getDefaultType());
            } catch (KeyStoreException e2) {
                C7262a.m26918a(getClass().getSimpleName()).mo20477c(e2);
                keyStore = null;
            }
        }
        if (keyStore != null) {
            try {
                keyStore.load((KeyStore.LoadStoreParameter) null);
                if (!keyStore.containsAlias("pwdEncryptionKey")) {
                    Calendar instance = Calendar.getInstance();
                    Date time = instance.getTime();
                    instance.add(1, 1);
                    Date time2 = instance.getTime();
                    KeyPairGenerator instance2 = KeyPairGenerator.getInstance("RSA", "AndroidKeyStore");
                    instance2.initialize(new
KeyPairGeneratorSpec.Builder(this.f19437d).setAlias("pwdEncryptionKey").setStartDate(time).setEndDate(time2).setSe
rialNumber(BigInteger.valueOf(1)).setSubject(new X500Principal("CN=Circle")).build());
                    instance2.generateKeyPair();
                }
            } catch (Exception e3) {
                C7262a.m26918a(getClass().getSimpleName()).mo20477c(e3);
```

Output Stream Encryption

```
public String mo20392a(String str) {
        if (!mo20393a()) {
            return null;
        }
        try {
            Cipher instance = Cipher.getInstance("RSA/ECB/PKCS1Padding");
            instance.init(1, (RSAPublicKey) ((KeyStore.PrivateKeyEntry) this.f19436c.getEntry("pwdEncryptionKey",
(KeyStore.ProtectionParameter) null)).getCertificate().getPublicKey());
            ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
            CipherOutputStream cipherOutputStream = new CipherOutputStream(byteArrayOutputStream, instance);
            cipherOutputStream.write(str.getBytes(Utf8Charset.NAME));
            cipherOutputStream.close();
            return Base64.encodeToString(byteArrayOutputStream.toByteArray(), 0);
        } catch (Exception e) {
            C7262a.C7264a a = C7262a.m26918a(getClass().getSimpleName());
            a.mo20479e("Encryption Error: " + e.getMessage(), new Object[0]);
            return null;
        }
    }
```

Input Stream Decryption

```
public String mo20394b(String str) {
        if (!mo20393a()) {
            return null;
        }
        try {
            Cipher instance = Cipher.getInstance("RSA/ECB/PKCS1Padding");
            instance.init(2, ((KeyStore.PrivateKeyEntry) this.f19436c.getEntry("pwdEncryptionKey",
(KeyStore.ProtectionParameter) null)).getPrivateKey());
            CipherInputStream cipherInputStream = new CipherInputStream(new
ByteArrayInputStream(Base64.decode(str, 0)), instance);
            ArrayList arrayList = new ArrayList();
            while (true) {
                int read = cipherInputStream.read();
                if (read == -1) {
                    break;
                }
                arrayList.add(Byte.valueOf((byte) read));
            }
            byte[] bArr = new byte[arrayList.size()];
            for (int i = 0; i < bArr.length; i++) {
                bArr[i] = ((Byte) arrayList.get(i)).byteValue();
            }
            return new String(bArr, 0, bArr.length, Utf8Charset.NAME);
        } catch (Exception e) {
            C7262a.m26918a(getClass().getSimpleName()).mo20479e("Decryption Error: " + e.getMessage(), new
Object[0]);
            return null;
        }
    }
```

4. Accepted cipher suites:

```
private static final C2215h[] f6977d = {
C2215h.TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256, C2215h.TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
C2215h.TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,
C2215h.TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,
C2215h.TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,
C2215h.TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,
C2215h.TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,
C2215h.TLS_DHE_RSA_WITH_AES_128_CBC_SHA,
C2215h.TLS_DHE_RSA_WITH_AES_256_CBC_SHA,
C2215h.TLS_RSA_WITH_AES_128_GCM_SHA256,
C2215h.TLS_RSA_WITH_AES_128_CBC_SHA,
C2215h.TLS_RSA_WITH_AES_256_CBC_SHA,
C2215h.TLS_RSA_WITH_3DES_EDE_CBC_SHA};
```

5. Firmware is not downloaded through the mobile app. As we see in `UpdateFimwareService` class, the app only commanded the camera to download and apply the firmware through the API.

```
public void update(String str, String str2, String str3) {
            this.mAccessoryID = str;
            this.mFirmwareVersion = str3;
            this.mFirmwareId = str2;
            C7262a.m26918a(getClass().getSimpleName()).mo20465c("update accId %s fwV %s fwId %s",
this.mAccessoryID, this.mFirmwareVersion, this.mFirmwareId);
            startUpdating(new StartUpdateCallback(str2) {
                        /* access modifiers changed from: package-private */
                        public void onStarted(String str) {
                                    String unused = UpdateFirmwareService.this.mUpdateFirmwareCmdId = str;
                        }
            });
}

private void startUpdating(StartUpdateCallback startUpdateCallback) {
            C7262a.m26918a(getClass().getSimpleName()).mo20465c("startUpdating ", new Object[0]);
            this.mConnectionError = LogiError.None;
            handleStateChange(UpdateState.INITIAL);
            this.mRebootState = CameraRebootState.UNDEFINED;
            this.mCheckCameraRequest = this.mAccessoryManager.getAccessoryById(this.mAccessoryID,
startUpdateCallback);
}
```

6. Authentication certificate for Wi-Fi connection in `BleGetCertificate` class.

```
public void onSuccess(String str) {
            try {
                        X509Certificate i = this.camera.mo14655i();
                        i.checkValidity();
                        X509Certificate x509Certificate = (X509Certificate)
CertificateFactory.getInstance("X.509").generateCertificate(new ByteArrayInputStream("-----BEGIN CERTIFICATE-----
\nMIIFHjCCBAagAwIBAgIDEAAHMA0GCSqGSIb3DQEBCwUAMIGBMREwDwYDVQQKEwhM\nb2dpdGVjaDENMAsGA1UECxMEcm9vdDEPMA0GA1UEBxMGTm
V3YXJrMRMwEQYDVQQI\nEwpDYWxpZm9ybmlhMQswCQYDVQQGEwJVUzEqMCgGA1UEAxMhTG9naSBDaXJjbGUg\nQ2VydGlmaWNhdGUgQXV0aG9yaXR5
MB4XDTcwMDEwMTAwMDAwMVoXDTM2MDEwMTAw\nMDAwMFowajELMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExETAPBgNV\nBAoTCExvZ2
l0ZWNoMQswCQYDVQQLEwJjYTEmMCQGA1UEAxMdQ2lyY2xlIERldmlj\nZSBJbnRlcm1lZGlhdGUgQ0EwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwgg
EKAoIB\nAQDSkuj8jigD3Nj9ehrJuF59VYVbn1IG3X79VWmzHT/HVLByXsbqJppHY94rEZm0\nG2eB/3Ekwj40KvCFE1N7zzwUWK05lv07IkIcrMl4
BxjJlcHusLHfXPEiu/Mk+MVh\n5SDl9BcKf0DS2KZZzhYYYi7818EgJieFwu5n0apeNTLxNF/9KmGTEeq60/BPtEIF\nqpwcUhB7R5QMqi1oajWZkd
cZcI+d0P+SNMH+PHTQ17iX/MS2JTmsljvfrV+GdAtA\n8i/y2Wrlkw3ia7mw4ffenmU3cRayn7ouddSXh2lvJgH7Hqagt4m/CxqqFIxjc00/\nyuce
fDamzg+t6x/cYKrWci5bAgMBAAGjggGzMIIBrzASBgNVHRMBAf8ECDAGAQH/\nAgEAMB0GA1UdDgQWBBTVnE2IG+UEZAZw5BGscvo0BI8nzzCBrgYD
VR0jBIGmMIGj\ngBRW1DK0btcAzvxdWUqlS+lrbYn0/6GBh6SBhDCBgTERMA8GA1UEChMITG9naXRl\nY2gxDTALBgNVBAsTBHJvb3QxDzANBgNVBA
cTBk5ld2FyazETMBEGA1UECBMKQ2Fs\naWZvcm5pYTELMAkGA1UEBhMCVVMxKjAoBgNVBAMTIUxvZ2kgQ2lyY2xlIENlcnRp\nZmljYXRlIEF1dGhv
cml0eYIBATAOBgNVHQ8BAf8EBAMCAYYwOwYDVR0fBDQwMjAw\noC6gLIYqaHR0cDovL2NlcnQudmlkZW8ubG9naS5jb20vY2lyY2xlX3Jvb3QuY3Js
\nMEYGCCsGAQUFBwEBBDowODA2BggrBgEFBQcwAoYqaHR0cDovL2NlcnQudmlkZW8u\nbG9naS5jb20vY2lyY2xlX3Jvb3QucGVtMDQGA1UdJQEB/w
QqMCgGCCsGAQUFBwMJ\nBggrBgEFBQcDCAYIKwYBBQUHAwIGCCsGAQUFBwMBMA0GCSqGSIb3DQEBCwUAA4IB\nAQCpYYBqI4+ynblkcde9lDfxvHdQ
CnKnGNXavAgV3FidcL3ZFdWUmaY83+uBmQWF\nKdrzdJ15/7wQLrkPwn0APnFeZZL0zoU94Pl2tzSW4Csh48ToqGJEKAMH5WZ9hBew\nR0np0wGeZm
MfLcS+RH+Z0nxxyA121VYoaxXohN30BAiU2tRqUFC5yngsUsXpIWrX\nrBTxA1DYtckLeh4VeGmc/4GZAfBq8kqnJYwTrG1c81fAz1kAa+ypf21A+4
MmomHL\ne4D0m/8srqTkW+UXSsqVYJ2scU0otAgJe1lk20z/fbWt2i+LaqdUZzbcLbmVtsWK\nyLUN6P5vGyU8UD0l0oIPdSdg\n-----END
CERTIFICATE-----".getBytes(StandardCharsets.UTF_8)));
                        x509Certificate.checkValidity();
                        i.verify(x509Certificate.getPublicKey());
                        statusChanged(new
SetupServiceBaseState.StateStatusWithPayload(BleGetCertificateStatus.SUCCESS));
            } catch (Exception e) {
                        e.printStackTrace();
                        statusChanged(new
SetupServiceBaseState.StateStatusWithPayload(BleGetCertificateStatus.INVALID_CERTIFICATE).setPayload((Object)
null));
            }
}
```

3.4 API analysis

Logitech provides an API for the Circle camera at the following link:

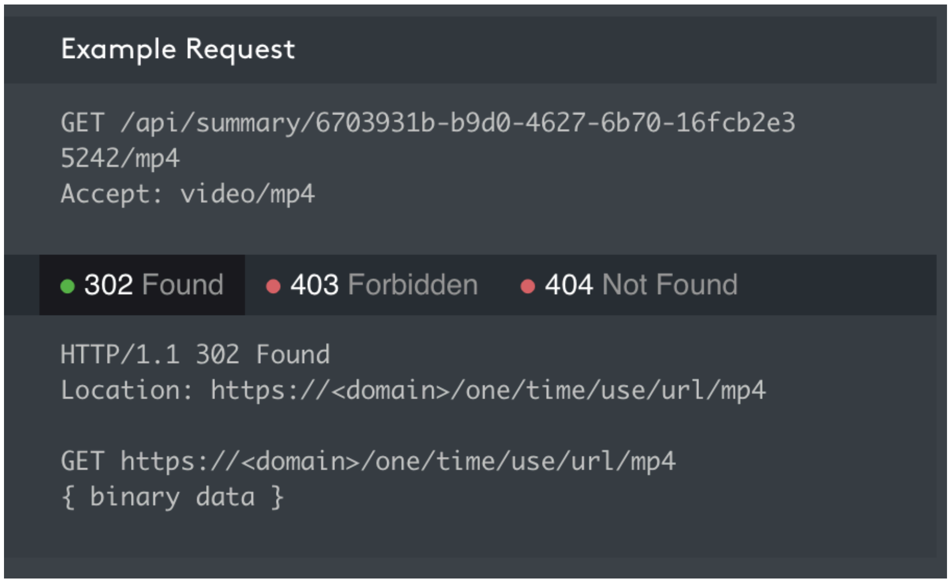
`https://developers.logitech.com/circle`

It provides control on: Authentication, Account, Notification, Accessories, Live View, Activities, Summaries and App Integration. Any camera can be wrapped as a Python object by using this API: https://github.com/evanjd/python-logi-circle

Now we are going to describe the API categories which the API provides control on.

- Authentication: it is done through `RFC6749` "The OAuth 2.0 Authorization Framework", which enables a third-party application to obtain limited access to an HTTP service. `OAuth2` refresh tokens are good for 60 days. It allows among other things to create a Logitech account.
- Account: A Logitech account allows to create a new Circle account and get information about it.
- Notification: After having registered for client notifications, in the case of an event, it allows to notify the client. Examples of an event is `MOTION_STARTED`, which indicates that the camera has detected a motion.
- Live View: It allows, specifying the ID of an accessory, to get a media stream or a snapshot. Some of the available formats are JPG for the snapshot and WebRTC for live video.
- Activities: It permits to access and delete the recordings.
- Summaries: They are a form of a short time-lapse video, of a chosen length and between a chosen date interval. It allows to access or delete summaries.

The Logitech Developer Hub provides example calls for the APIs. The following is a snippet of code for a request to get a summary file in the MP4 format. The example parameter "`6703931b-b9d0-4627-6b70-16fcb2e35242"` is the `summary_id`, that is the ID of the summary requested.


Example Request

```
GET /api/summary/6703931b-b9d0-4627-6b70-16fcb2e3
5242/mp4
Accept: video/mp4
```

```
HTTP/1.1 302 Found
Location: https://<domain>/one/time/use/url/mp4

GET https://<domain>/one/time/use/url/mp4
{ binary data }
```

Figure 6 Example of an MP4 summary request

4 Security analysis

4.1 Device security

Our Wireshark captures reveal that the camera maintains encrypted connections on port 443 of the cloud server. We did not find any unencrypted video or configuration data being sent or received apart from NTP and DNS requests. Attacks like DNS spoofing would fail as the client (the camera) attempts to check the cloud server's TLS certificate and successfully connects only if its identity has been verified. Router spoofing and network sniffing using `Bettercap` were not successful as we dealt with encrypted traffic and partially because of the bugs we encountered using the tool.

A full TCP and UDP port scan of the camera were done using `nmap`. All TCP ports appeared to be closed during our testing whereas a UDP scan revealed port 68 to be open which is not unusual because this port is used for receiving offered IP address from the DHCP server. We found another open UDP port 48457 but later found out that it uses those ports for DNS requests and that port numbers opened for use are completely randomized.

```
# Nmap 7.60 scan initiated as: nmap -Pn -n -p- -vv -T4 -oA final_TCP_scan CAMERA_IP
Nmap scan report for CAMERA_IP
Host is up, received user-set (0.0037s latency).
All 65535 scanned ports on CAMERA_IP are closed because of 65535 conn-refused

Read data files from: /usr/bin/../share/nmap
# Nmap done -- 1 IP address (1 host up) scanned in 221.09 seconds
```

Figure 9 TCP scan result - All ports closed

```
sudo nmap -p- IP_ADDR -sU -v

Starting Nmap 7.60 ( https://nmap.org ) at 2020-03-21 00:29 GMT
Initiating ARP Ping Scan at 00:29
Scanning IP_ADDR [1 port]
Completed ARP Ping Scan at 00:29, 0.23s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 00:29
Completed Parallel DNS resolution of 1 host. at 00:29, 0.01s elapsed
Initiating UDP Scan at 00:29
Scanning IP_ADDR_ADDR [65535 ports]
.
.
Completed UDP Scan at 20:47, 73107.01s elapsed (65535 total ports)
Nmap scan report for IP_ADDR
Host is up (0.089s latency).
Not shown: 65533 closed ports
PORT      STATE         SERVICE
68/udp    open|filtered dhcpc
48457/udp open|filtered unknown
MAC Address: 44:73:D6:01:90:B9 (Logitech)

Read data files from: /usr/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 73107.38 seconds
           Raw packets sent: 91388 (2.561MB) | Rcvd: 71717 (4.018MB)
```

Figure 10 UDP scan result

The recording state and other settings of the camera are controlled using the mobile app so we set up an intercepting proxy using Burpsuite in order to see and modify requests and responses sent between the app and cloud server but the proxy logs showed that the app failed to negotiate a TLS connection to `video.logi.com` on port 443, so we can confirm that it does not accept self-signed certificates.

4.2 Application security

The android companion app doesn't seem to have any security flaws.

- Almost every communication to the camera is done through the API server using a secure HTTP over TLS connection.
- One time the app connected directly to the camera is during setup mode. Analyzing the process on the android application, user account and password handling seems secure because it was encrypted before it was sent to the camera.
- All passwords were never written into the application log.
- The firmware update process does not involve any file download by the app. The application instructed the camera through the API server to download the firmware update directly. So, attacker cannot inject a modified firmware through the app.

4.3 API Security

When assessing the security of the API, we should consider that Logitech does not currently temporarily release any new API key (because of the covid-19 pandemic), therefore we had not tested it. What we can say about it is that the RFC6749 is the industry-standard protocol for authorization and there are no known critical vulnerabilities.

5 Concerns

During the test, several minor security concerns were also discovered:

- A corrupt or invalid configuration of the TLS used in device communications can lead to vulnerabilities such as heart-bleed.
- The video stream is sent to the server before logging in to the mobile application and can present attack vectors for an evil twin attack.
- The Bluetooth connection phase can potentially be target with an Evil Twin and Man-In-The-Middle attack.

6 Conclusion

The penetration test conducted confirmed that the device appears to be securely built and the following security measures were observed.

- All communication is done through TLS 1.2.
- App does not accept self-signed certificates.
- Ports do not show through port scanners.
- Things handled by/in the server:
    - Firmware updates
    - Storage of footage
    - Motion and other intensive computations
- Almost no communication directly between phone and device.

The security of the device seems to have been designed with the following security design principles in mind:

- The attack surface at each level is minimal.
- Default security controls have been implemented at each level in the architecture that reduces the risk of errors or vulnerabilities.

Based on the scope of this penetration test, this device appears to be sufficiently secure.