Aktywacja aplikacji IKO – Kampania złośliwego oprogramowania

Archived: 2026-04-05 14:09:00 UTC

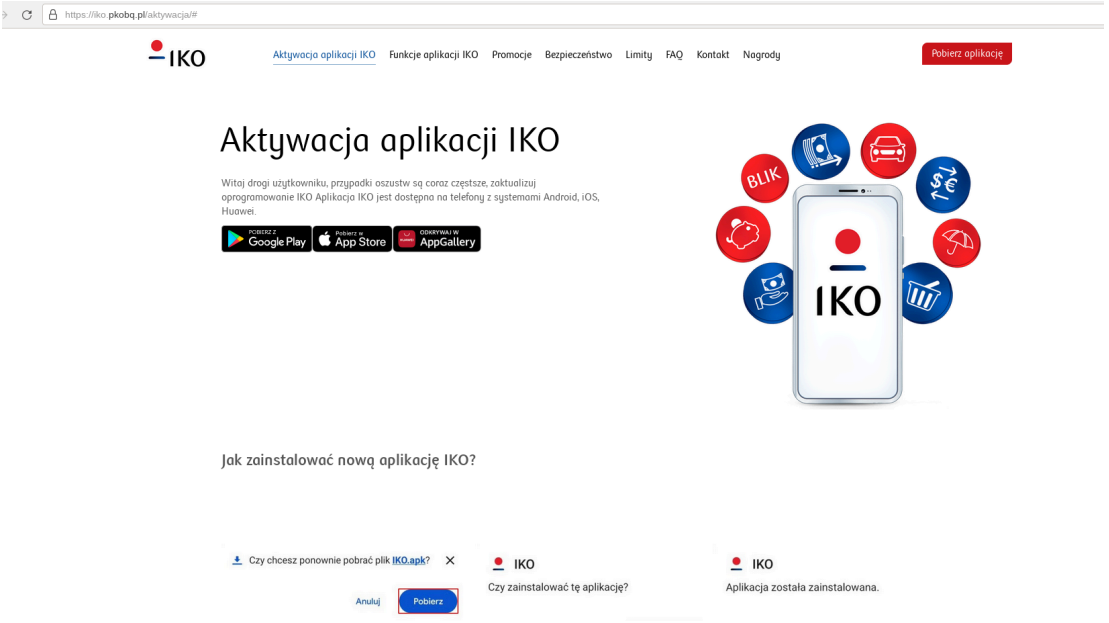
W ostatnim czasie obserwowaliśmy kolejną kampanię złośliwego oprogramowania wymierzoną w użytkowników urządzeń mobilnych z systemem Android. Szkodliwa aplikacja, podszywa się pod aplikację banku PKO BP – IKO. Po analizie okazało się, że w atakach wykorzystywany jest obserwowany po raz pierwszy w Polsce trojan bankowy **Coper**.

W dniu 28 grudnia 2021 dostaliśmy pierwsze zgłoszenie o podejrzanym SMSie w którym nadawca wiadomości prosi o odnowienie systemu poprzez "aktywację" aplikacji:

```
 Szanowny kliencie.W ostatnim czasie odnotowaliśmy aktywność ataków chakerskich.
Prosimy odnowić system https://iko.pkobq.pl/aktywacja/
```

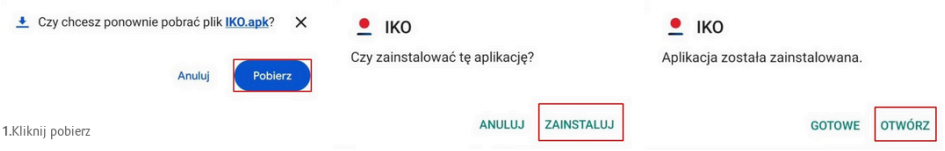
Na uwagę zasługuje słowo "chakerskich" pisane przez ch i brzmiące dziwnie po polsku zdanie "Prosimy odnowić system". Oczywiście złośliwa domena od razu znalazła się na naszej liście ostrzeżeń, a dla samej domeny `pkobq.pl` został wystawiony wniosek o usunięcie z rejestru.

Po kliknięciu w link zawarty w wiadomości SMS, użytkownik zostaje przeniesiony na fałszywą stronę banku PKO BP. Jest ona przygotowana profesjonalnie i przypomina prawdziwą stronę aplikacji IKO.



Atakujący przygotowali instrukcję dla użytkownika krok po kroku w jaki sposób zainstalować aplikację i nadać jej dodatkowe uprawnienia. Po pierwszym uruchomieniu pojawia się okno, które w natarczywy sposób domaga się wyrażenia zgody na korzystanie z usług ułatwień dostępu (ang. accessibility services).

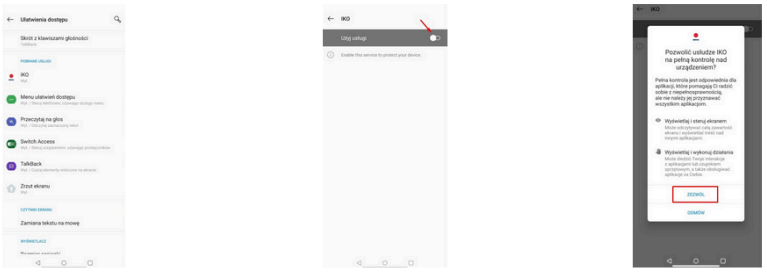
Ułatwienia dostępu, mają z założenia wspomagać obsługę systemu osobom niepełnosprawnym, ale są one często wykorzystywane przez złośliwe oprogramowanie do przejęcia kontroli nad urządzeniem. Jeżeli użytkownik zgodzi się, aby złośliwe oprogramowanie korzystało z tej funkcjonalności, może ono samodzielnie imitować działania użytkownika jak klikanie w przyciski, czy zamykanie okien. W efekcie przejmuje ono pełną kontrolę nad urządzeniem.



1.Kliknij pobierz

2. Kliknij Zainstaluj.

3. Otwórz nową aplikację IKO.

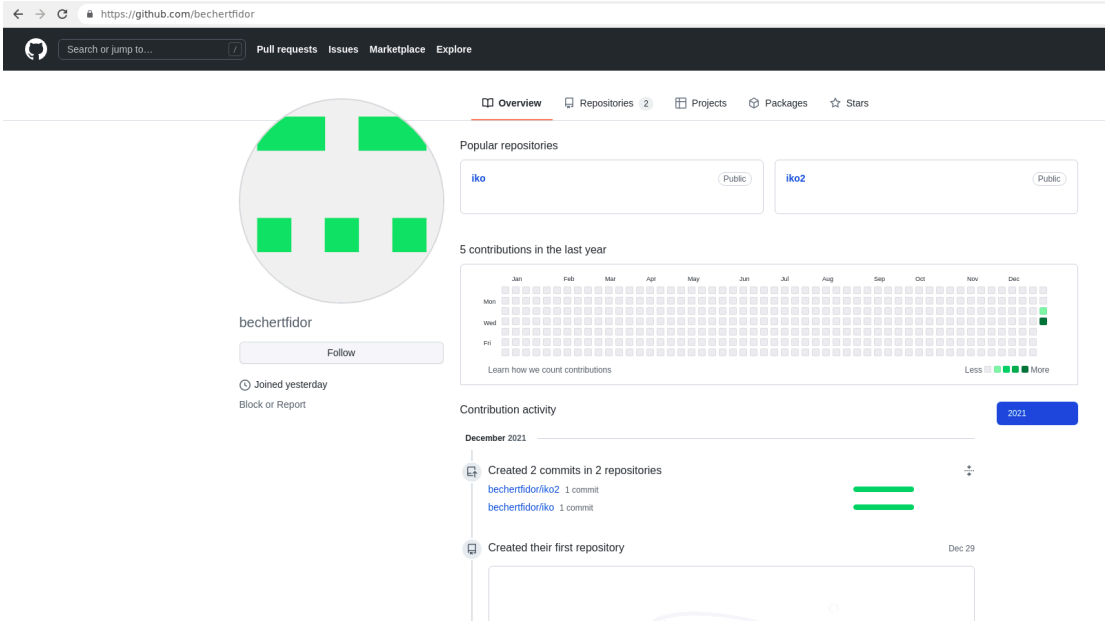


4.

5. Aktywuj tę funkcję

6. kliknij zezwól, aby kontynuować z nowym programem

Ciekawą obserwacją jest to, że przez pewien czas, aktor kampanii do hostowania złośliwego pliku APK używał platformy GitHub. Jednak po usunięciu kilku kolejnych kont zdecydował się na kierowanie użytkowników bezpośrednio na aplikację znajdującą się na jego stronie.



Pobierany plik oczywiście nie jest prawdziwą aplikacją banku, lecz próbką mało znanego, złośliwego oprogramowania z rodziny **Coper**. Jedyny na te chwile dostępny materiał na jego temat pochodzi z analizy dokonanej przez firmę Dr.WEB, gdy była ona używana do infekowania użytkowników z Kolumbii: https://news.drweb.com/show/?i=14259

Analiza techniczna

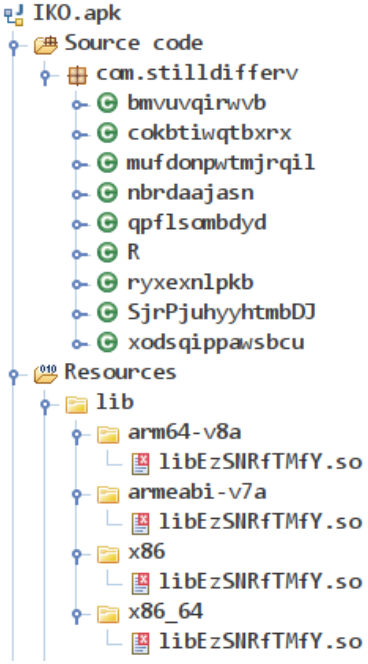
Po zdekompilowaniu pliku APK, np. narzędziem JADX, na pierwszy rzut oka nie wydaje się on szczególnie ciekawy – większość klas jest pusta i nie widać w nich żadnych ciekawych funkcjonalności. Możemy zacząć podejrzewać, że aplikacja została w jakiś sposób spakowana.

Naszą uwagę może przykuć klasa która odwołuje się do załączonej biblioteki natywnej `EzSNRfTMfY`.

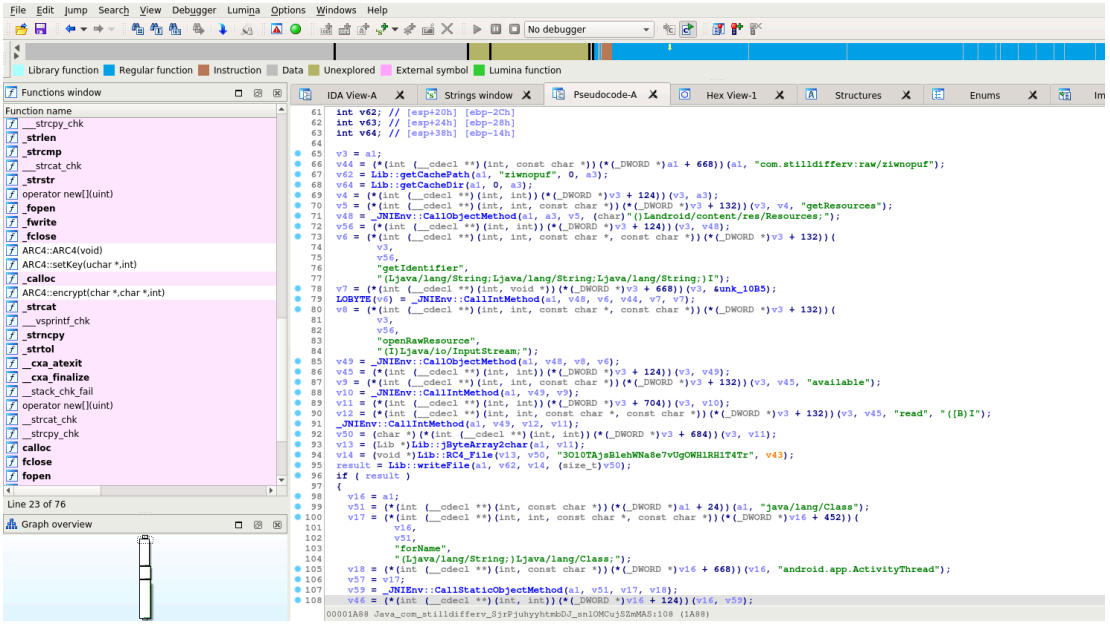
```
public class SjrPjuhyyhtmbDJ extends Application {
   static {
```

```
        System.loadLibrary("EzSNRfTMfY");
    }

    /* access modifiers changed from: protected */
    public void attachBaseContext(Context context) {
        super.attachBaseContext(context);
        snlOMCujSZmMAS(context);
    }

    public native void snlOMCujSZmMAS(Object obj);
}
```

Możemy również stwierdzić jej obecność korzystając z przeglądarki plików projektu. Na poniższym screenie jest widoczna jako `libEzSNRfTMfY.so`.



Kolejnym krokiem będzie więc analiza tej biblioteki, a w szczególności wywoływanej z niej funkcji `snlOMCujSZmMAS`. W tym celu możemy skorzystać np. z dekompilatora dostępnego w IDA Pro.

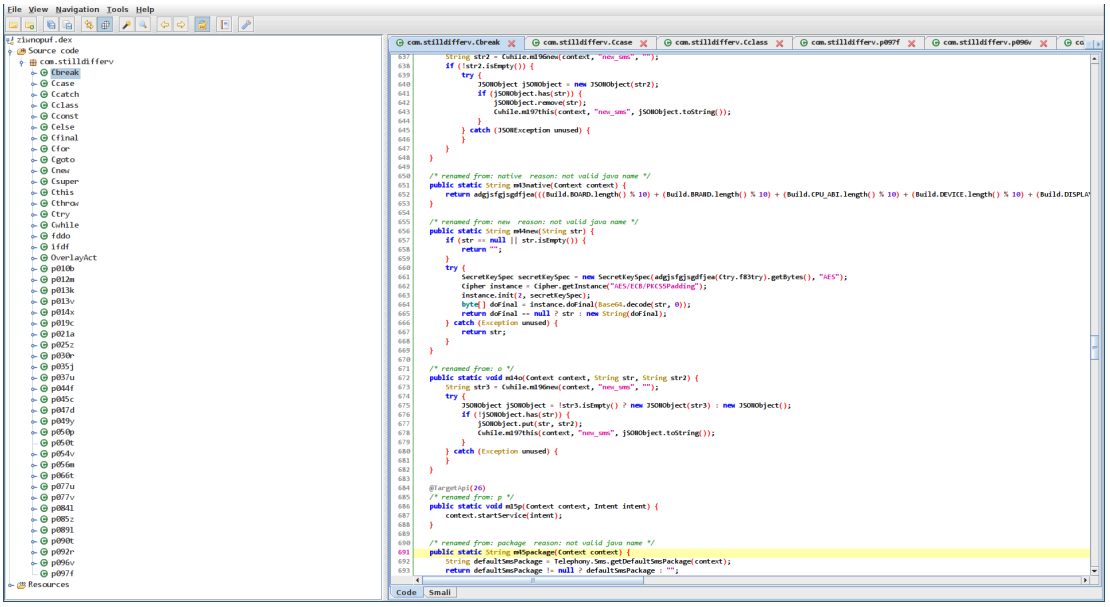


Dostęp do zdekompilowanego kodu bardzo pomaga w ustaleniu działania biblioteki. Szybko dochodzimy do wniosku, że deszyfruje ona załączony plik `ziwnopuf`, korzystając z szyfru RC4 i stałego klucza – w tym przypadku `3O10TAjsBlehWNa8e7vUgOWHlRH1T4Tr`.

Działanie tego kodu możemy bardzo prosto zasymulować, korzystając ze stworzonej przez nas biblioteki do pracy nad złośliwym oprogramowaniem – Malduck. Kod z jej wykorzystaniem pozwalający odszyfrować plik `ziwnopuf` wygląda następująco:

```
 from malduck import rc4
with open("res/raw/ziwnopuf", "rb") as f:
    data = f.read()

decrypted = rc4(b"3O10TAjsBlehWNa8e7vUgOWHlRH1T4Tr", data)

with open("decrypted.dex", "wb") as f:
    f.write(decrypted)
```

Wynikowy plik dex jest już o wiele ciekawszy.



Niektóre łańcuchy znaków są szyfrowane ponownie za pomocą RC4 i klucza `7BLiz2PK6wbk1mhp`, ale również tym razem nie sprawia to zbyt wiele problemu i analogicznie możemy wykorzystać Malduck.

Wynikowo dostaniemy odszyfrowane stringi. Jednym z nich są wykorzystywane serwery C&C:
`'https://s22231232fdnsjds.top/PArhFzp5sG2sN/|https://s32231232fdnsjds.top/PArhFzp5sG2sN/|https://s42231232fdnsjds.top/PArhFz`

Spędzając trochę czasu na dalszej analizie możemy ustalić, że zapytania i odpowiedzi są szyfrowane za pomocą szyfru AES w trybie blokowym ECB, a następnie kodowane z wykorzystaniem Base64:

```
public static String m59try(String str) {
   if (str == null || str.isEmpty()) {
       return "";
   }
   try {
       SecretKeySpec secretKeySpec = new SecretKeySpec(adgjsfgjsgdfjea(Ctry.f83try).getBytes(), "AES");
       Cipher instance = Cipher.getInstance("AES/ECB/PKCS5Padding");
       instance.init(1, secretKeySpec);
       byte[] doFinal = instance.doFinal(str.getBytes());
       return doFinal == null ? str : Base64.encodeToString(doFinal, 0);
   } catch (Exception unused) {
       return str;
```

```
    }
}
```

Natomiast klucz dla tego szyfrowania jest generowany korzystając z funkcji skrótu MD5 – `54569d2aaae7176335a67bf72e86736f`:

```
public static String adgjsfgjsgdfjea(String str) {
   try {
       MessageDigest instance = MessageDigest.getInstance("MD5");
       instance.update(str.getBytes());
       byte[] digest = instance.digest();
       StringBuilder sb = new StringBuilder();
       for (byte b : digest) {
           String hexString = Integer.toHexString(b & 255);
           while (hexString.length() < 2) {
               hexString = "0" + hexString;
           }
           sb.append(hexString);
       }
       return sb.toString();
   } catch (NoSuchAlgorithmException e) {
       e.printStackTrace();
       return "";
   }
}
```

Potrafiąc odszyfrować komunikację możemy przeanalizować jak ona wygląda, a następnie spreparować odpowiedni pakiet podszywający się za zainfekowane urządzenie. Więcej informacji o protokole komunikacji można znaleźć w raporcie Dr.WEB. Dzięki nawiązaniu takiej komunikacji, możemy uzyskać odpowiedź od serwera z aktualnymi danymi konfiguracyjnymi, m.in listą aplikacji, dla których autor przygotował webinjecty, czy dodatkowe domeny C&C.

```
{
   "response": "er1",
   "tasks": [],
   "injects_list": "au.com.auswidebank.auswidebank|au.com.bankwest.mobile|au.com.ingdirect.android|au.com.nab.mobile|au.c
   "extra_domains": "https://s122231232fdnsjds.top/|https://s222231232fdnsjds.top/|https://s322231232fdnsjds.top/|",
   "keylogger_enabled": null,
   "net_delay": "20"
}
```

W przypadku nieprawidłowego zapytania serwer udaje zwykły serwer HTTP, jednak można dostrzec zakomentowaną informację o błędzie w dekodowaniu zapytania – `<!-- aes_err: -->`:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL /PArhFzp5sG2sN/ was not found on this server.</p>
</body></html>
<!-- aes_err: -->
```

Na uwagę zasługuje lista webinjectów, czyli aplikacji dla których złośliwe oprogramowanie będzie próbowało wykraść dane. W większości przypadków mają one dopisek `.test` który powoduje, że nie zadziałają one i mogą sugerować wczesną fazę rozwoju oprogramowania. Gdy je usuniemy dostajemy następującą listę targetowanych aplikacji:

```
 au.com.auswidebank.auswidebank
au.com.bankwest.mobile
au.com.ingdirect.android
au.com.nab.mobile
au.com.pnbank.android
au.com.suncorp.SuncorpBank
bot.accessibility.hint
com.anz.android.gomoney
com.bankofqueensland.boq
com.bbva.bbvacontigo
com.bbva.netcash
com.bendigobank.mobile
com.commbank.netbank
com.fusion.banking
com.fusion.beyondbank
enterprise.com.anz.shield
es.bancosantander.apps
org.banksa.bank
org.bom.bank
org.stgeorge.bank
org.westpac.bank
uk.co.tsb.newmobilebank
```

Ciekawe jest to, że na chwilę obecną nie znajdziemy na niej polskich aplikacji, co nie znaczy, że infekcja tym złośliwym oprogramowaniem nie jest groźna. Poza mechanizmem webinjectów Coper pozwala m.in na przechwytywanie i wysyłanie SMSów, czy uruchomienie keyloggera.

IOC

Pliki APK:

Nazwa pliku	**Użytkownik Githuba**	**MD5**
IKO.apk	@steve229898	368f4b4d74a749ff55d76e929b52fedd
2.iko_com.mindsoonvfk_IKO.apk	@bechertfidor	aebe71b857e868b1af752f90255aaab5
2.iko_com.stoptravelg_IKO.apk	@bechertfidor	d2d8027baebf285703dc753219574d3e
3.iko_com.growmainwkmm_IKO.apk	@fidorde	d6f521d9e83160ee06d71dc217c56693
IKO.apk		d6f521d9e83160ee06d71dc217c56693

Serwery C&C zaszyte na stałe w próbkach:

- `https://s22231232fdnsjds[.]top/PArhFzp5sG2sN/`
- `https://s32231232fdnsjds[.]top/PArhFzp5sG2sN/`
- `https://s42231232fdnsjds[.]top/PArhFzp5sG2sN/`

Dynamiczne serwery C&C:

- `https://s122231232fdnsjds[.]top/`
- `https://s222231232fdnsjds[.]top/`
- `https://s322231232fdnsjds[.]top/`

Source: https://cert.pl/posts/2021/12/aktywacja-aplikacji-iko/