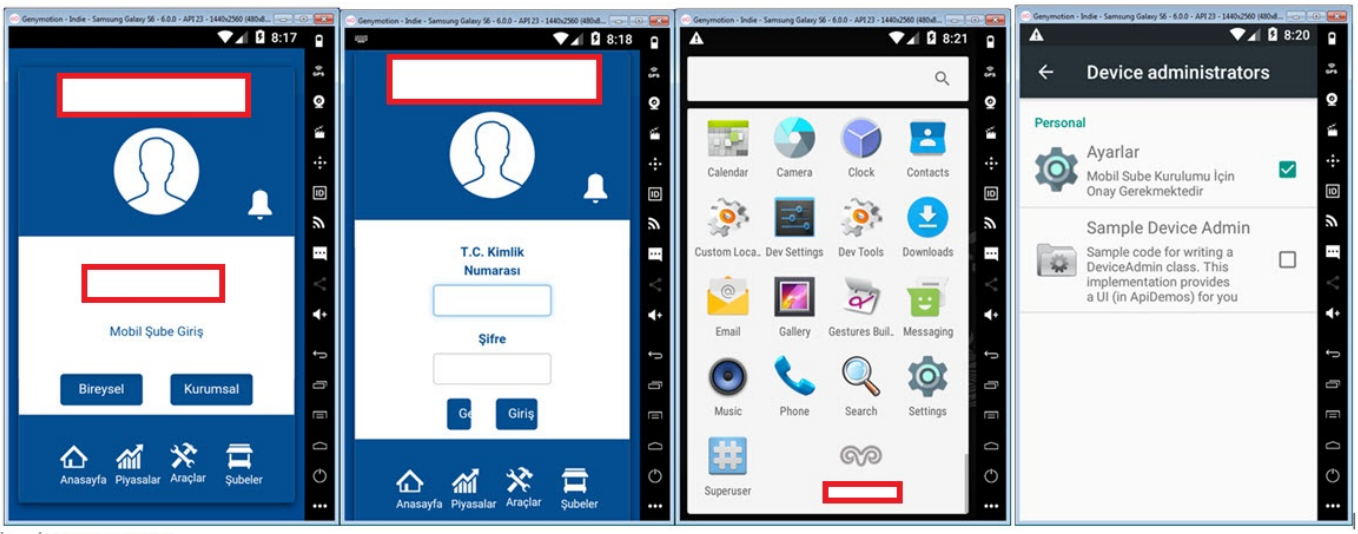
Android Hata Ayıklaması

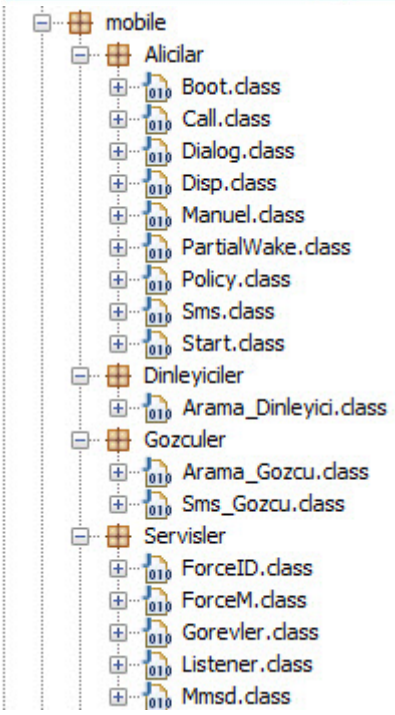
written by Mert SARICA | 1 April 2018

Yıllar içinde Android işletim sistemi yüklü mobil cihazların pazar payının bir hayli yükselmesi, teknoloji meraklıları kadar zararlı yazılım geliştiricilerinin de dikkatini çekmeyi başardı. Öyle ki günümüzde Windows işletim sistemi kullanıcıları nasıl zararlı yazılımlara karşı ilave güvenlik yazılımları kullanma zorunluluğu hissedip, günlük işleri arasında aldıkları sıradan bir e-postada yer alan bir bağlantı adresine (link) tıklamadan önce 40 kere düşünmek durumunda kalıyorlarsa, Android işletim sistemi kullanıcıları da benzer bir paranoya içinde günlerini geçirmeye başladılar. Nitekim haksız olduklarını söylemem oldukça güç özellikle bankacılık zararlı yazılımlarının gelişimini hem işi hem de özel ilgisi nedeniyle yakından takip eden bir güvenlik araştırmacısı olarak son aylarda karşılaştığım zararlı yazılımlar beni bir hayli şaşırtıyor.

Şöyle dönüp 2012 yılında yayımlamış olduğum Android Zararlı Yazılım Analizi başlıklı blog yazıma baktığımda, o zamanlar Android işletim sistemi için geliştirilmiş olan zararlı yazılımları analiz etmenin çok da zor olmadığını, günümüze kıyasla statik analizin zararlı yazılımları analiz etmede tek başına yeterli olabildiğini görüyorum. Günümüzde ise işin rengi git gide değişip işler, zararlı yazılım analistleri için her geçen gün daha da karmaşık bir hal alıyor.

İlk olarak ~2 yıl önce görülen ve hem Bakır EMRE'nin yazısına hem de benim Android Anti Anti-Emulator başlıklı blog yazıma konu olmuş olan, çağrı dinleme ve SMS mesajlarını okuma özelliklerine de sahip zararlı yazılımın güncel sürümü ile geçtiğimiz aylarda karşılaştığımda, kullanıcı arabiriminin (UI) yıllar içinde geldiği nokta beni fazlasıyla şaşırttı.

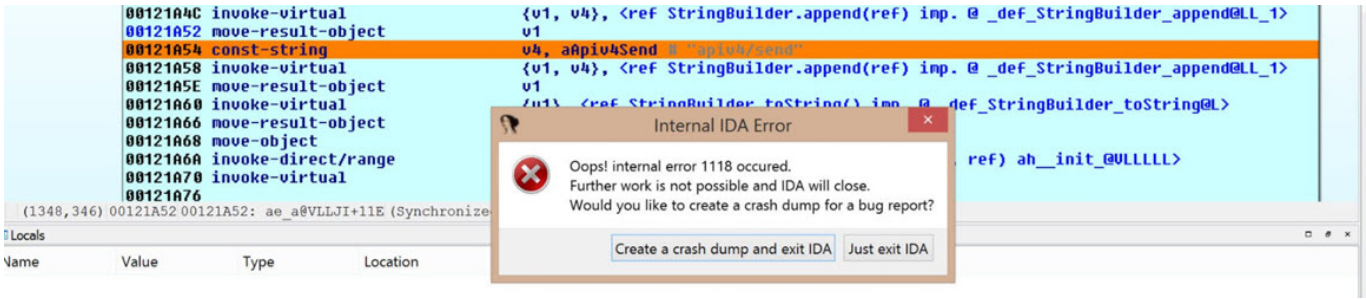




Kullanıcı arabirimi bir kenara, uygulamanın kaynak kodunda okunaklı olmayan karakter dizilerinin (strings) çalışma esnasında özel bir fonksiyon ile XOR işleminden geçirilerek çözülüyor olması ve ayrıca bazı önemli değerlerin (ip adresi, tor adresi vs.) AES şifreleme algoritması ile şifreli bir şekilde saklanıp yine benzer yöntemde farklı bir fonksiyon ile çalışma esnasında (run-time) çözülüyor olması, fonksiyon isimlerinin IıIIIııII gibi isimlerle adlandırılmış olması, statik analize karşı zararlı yazılım geliştiricisinin azmini ortaya koyuyordu. “Azmi görüyorum ve arttırıyorum!” dedikten sonra her ne kadar karakter dizilerini ve şifrelenmiş değerleri statik analiz ile çözmek, zor da olsa pratikte mümkün olsa da çok daha zor durumlarla karşı karşıya kalındığında hangi yöntemlerden, araçlardan faydalanılabileceğine bu yazıda yer vermek istedim.

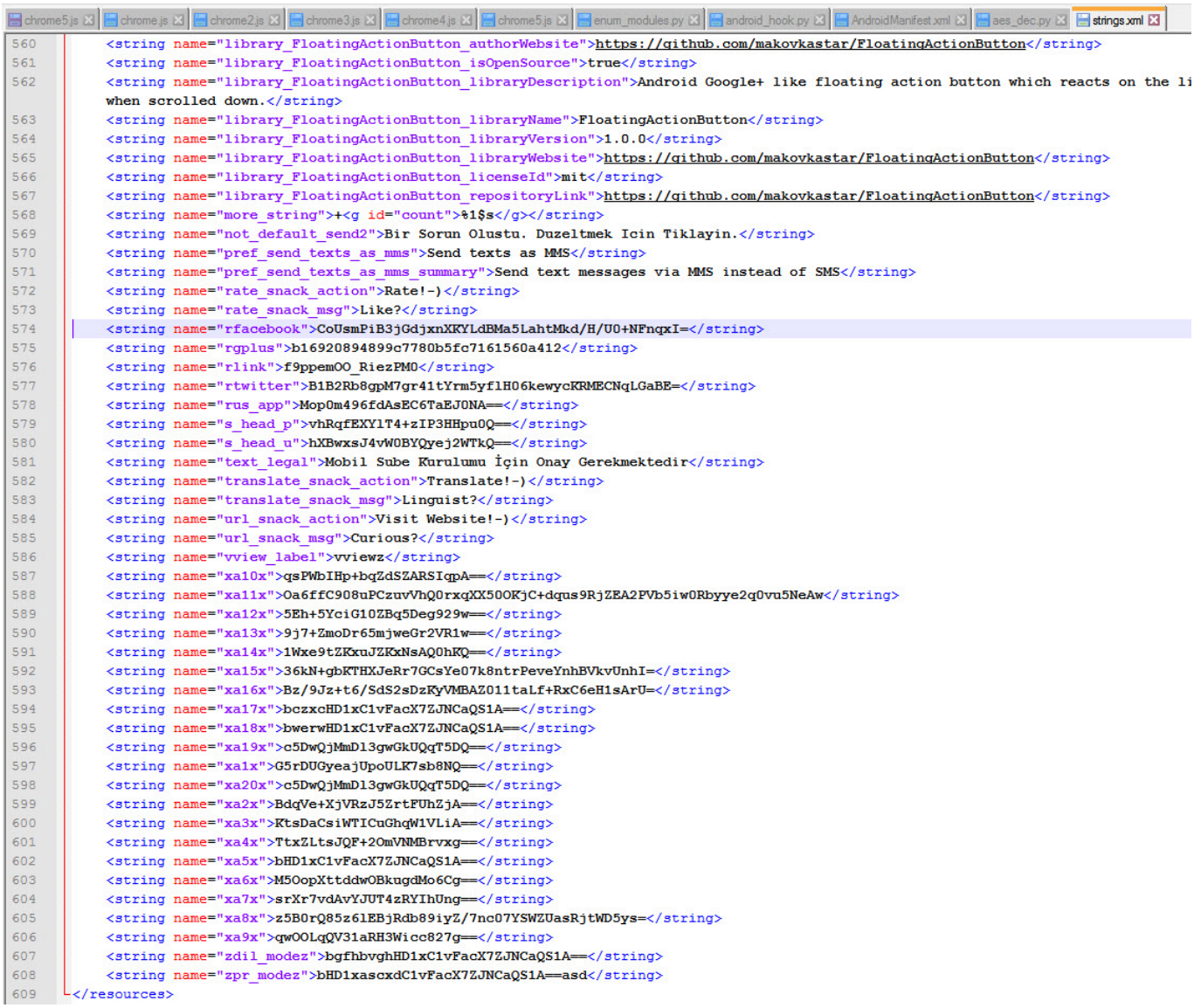
Windows zararlı yazılımlarını analiz ederken statik analizin yetersiz olduğu

noktalarda dinamik kod analizi ve x64dbg, IDA Pro gibi araçlar her daim kurtarcımız olsa da mevzu bahis Android işletim sistemi olduğunda araçlar ve yetenekleri sınırlı olabiliyor. Her ne kadar Casus Telefon başlıklı blog yazımda IDA Pro ile dinamik kod analizi yaptığıma yer vermiş olsam da perde arkasında IDA Pro'nun Android hata ayıklaması (debugging) konusunda stabil olmaması, yer yer göçmesi beni analiz esnasında oldukça zorlamıştı.

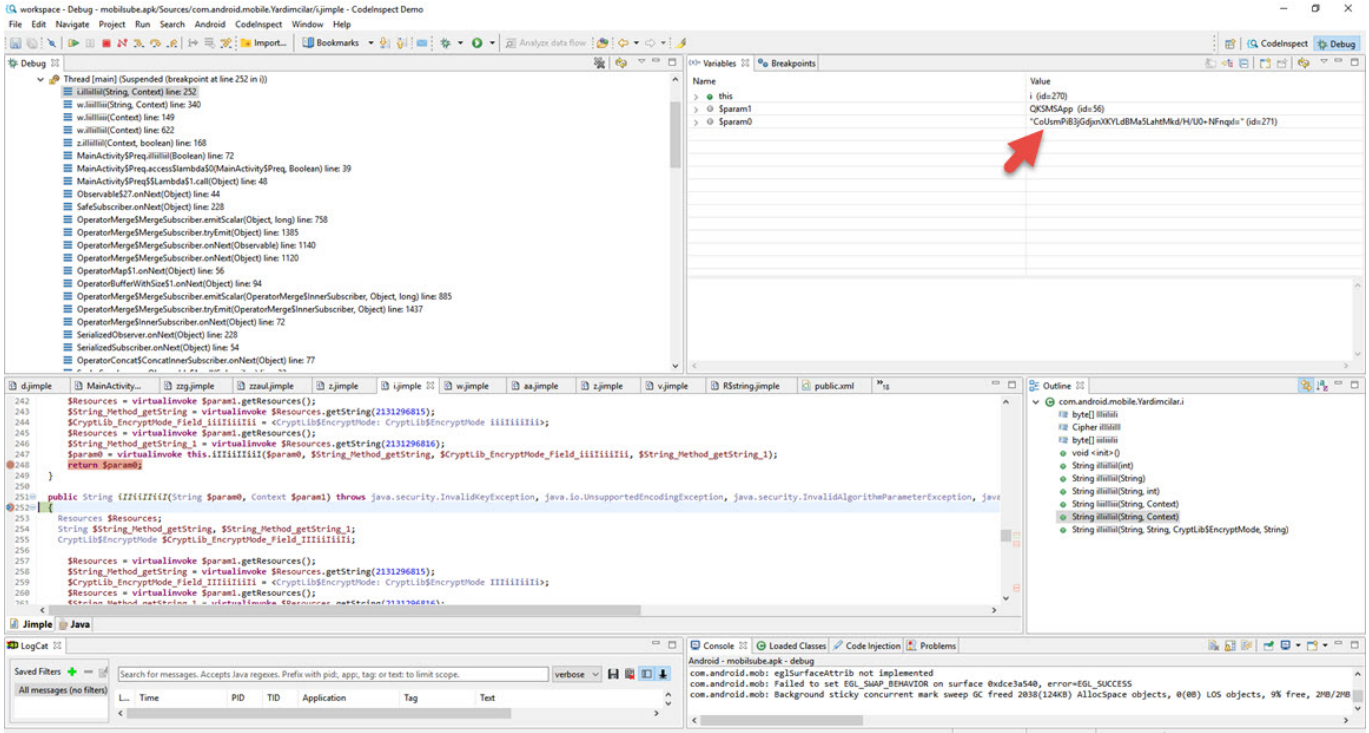


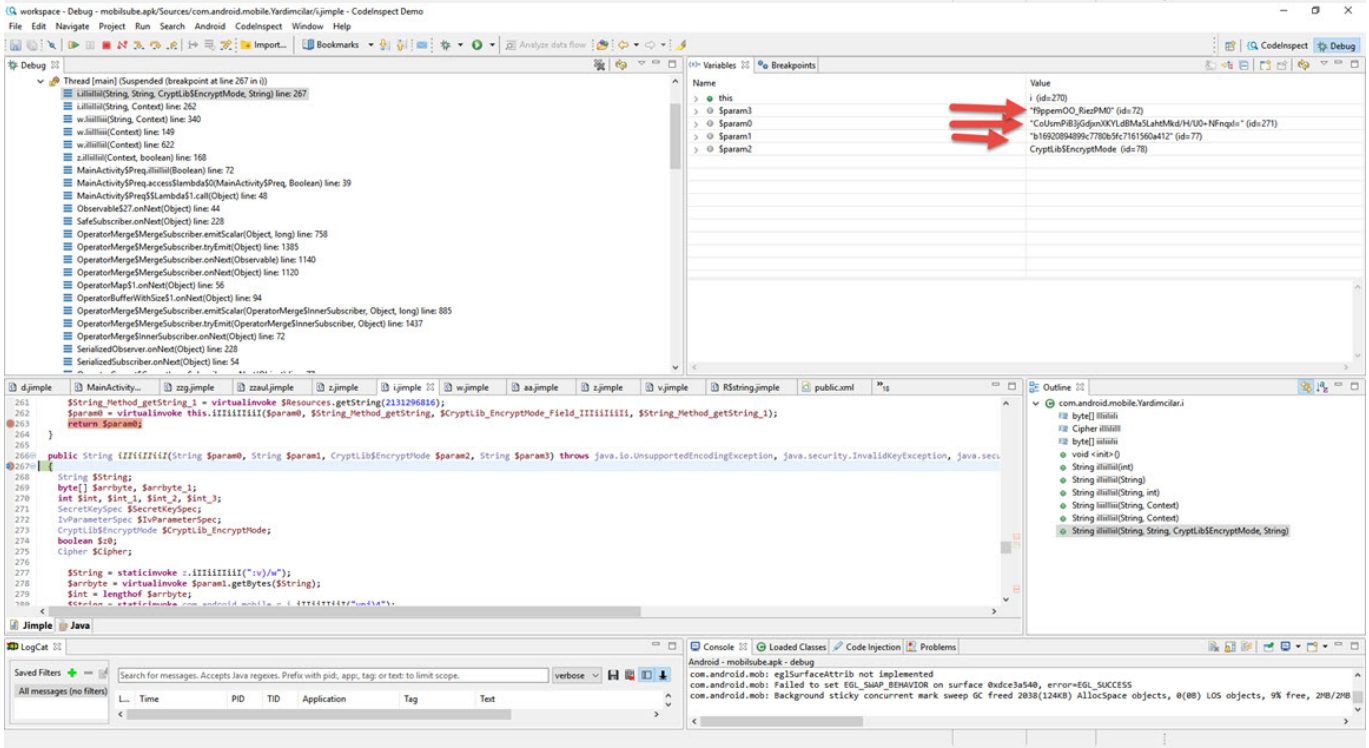
Geçtiğimiz aylarda Fortinet'in güvenlik araştırmacılarından biri olan Axelle APVRILLE'nin Android zararlı yazılım analizini konu alan bir sunumuna bakarken Code Inspect adındaki bir araç dikkatimi çekti. Daha önce böyle bir ticari aracın varlığından haberdar olmamış biri olarak elimin altındaki bu zararlı yazılım üzerinde denemeye karar verdim. Jadx aracı ile gerçekleştirdiğim statik analizde, şifreli değerlerin Resources dosyasında yer aldığını ve şifre çözme işleminin ise com.android.mobile.Yardimcilar.i sınıfında yer alan iIIiiIIiiI fonksiyonunda gerçekleştiğini tespit ettim.

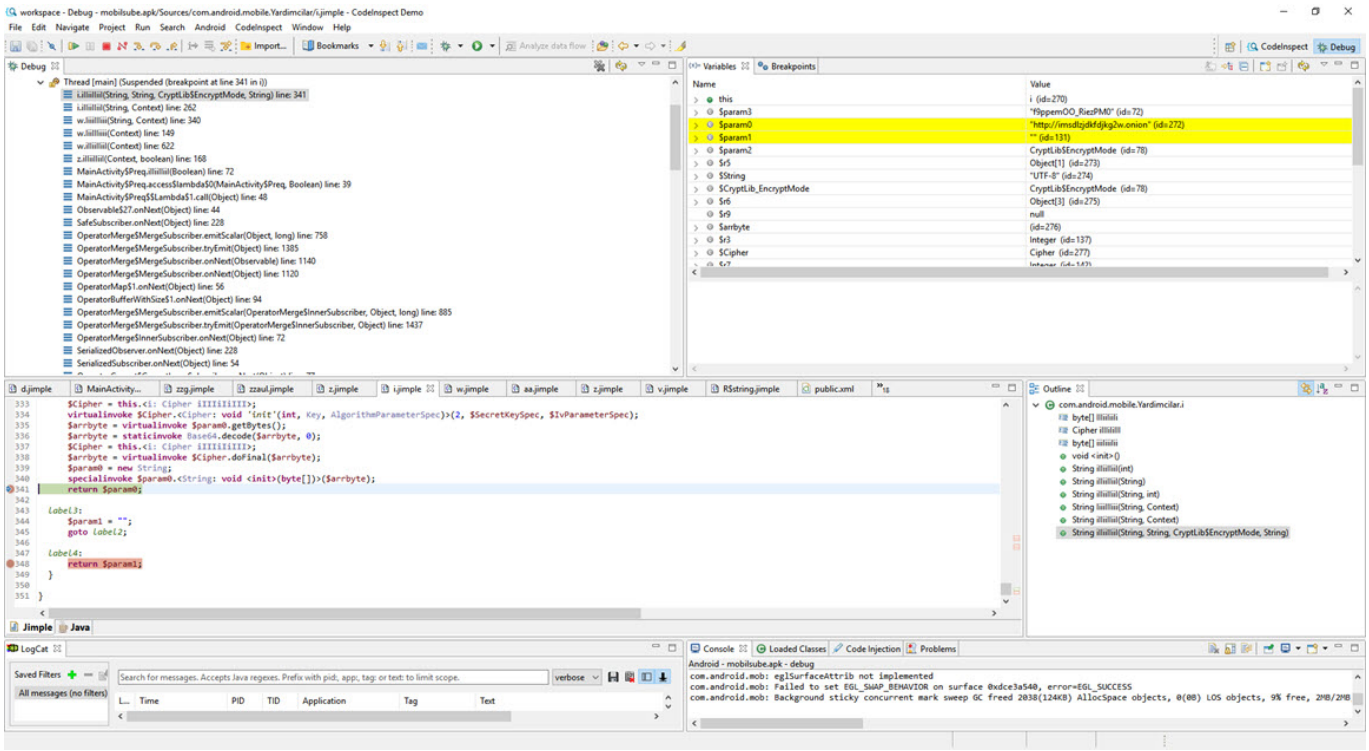

```
C:\Users\Mert\Desktop\YeniYazi\Frida\Gradle\src>grep -ri "rgplus" *
main/java/com/android/mobile/Yardimcilar/i.java:        return iIIiiIIiiI(arg0, arg1.getResources().getString(R.string.r
gplus), CryptLib$EncryptMode.iiiIiiiIii, arg1.getResources().getString(R.string.rlink));
main/java/com/android/mobile/Yardimcilar/i.java:        return iIIiiIIiiI(arg0, arg1.getResources().getString(R.string.r
gplus), CryptLib$EncryptMode.IIIiiIiiIi, arg1.getResources().getString(R.string.rlink));
main/res/values/strings.xml:    <string name="rgplus">b16920894899c7780b5fc7161560a412</string>

C:\Users\Mert\Desktop\YeniYazi\Frida\Gradle\src>grep -ri "rlink" *
main/java/android/support/v4/text/util/LinkifyCompat.java:            gatherLinks(links, spannable, PatternsCompat.AUTOL
INK_WEB_URL, new String[]{"http://", "https://", "rtsp://"}, Linkify.sUrlMatchFilter, null);
main/java/android/support/v4/text/util/LinkifyCompat.java:            gatherLinks(links, text, PatternsCompat.AUTOLINK_E
MAIL_ADDRESS, new String[]{"mailto:"}, null, null);
main/java/android/support/v4/text/util/LinkifyCompat.java:    private static void gatherLinks(ArrayList<LinkSpec> links,
 Spannable s, Pattern pattern, String[] schemes, MatchFilter matchFilter, TransformFilter transformFilter) {
main/java/com/android/mobile/Yardimcilar/i.java:        return iIIiiIIiiI(arg0, arg1.getResources().getString(R.string.r
gplus), CryptLib$EncryptMode.iiiIiiiIii, arg1.getResources().getString(R.string.rlink));
main/java/com/android/mobile/Yardimcilar/i.java:        return iIIiiIIiiI(arg0, arg1.getResources().getString(R.string.r
gplus), CryptLib$EncryptMode.IIIiiIiiIi, arg1.getResources().getString(R.string.rlink));
main/res/values/strings.xml:    <string name="rlink">f9ppemOO_RiezPM0</string>
```



CodeInspect aracı ile mobilsube.apk dosyasını açıp, com.android.mobile.Yardimcilar.i sınıfında yer alan iIIiiIIiiI fonksiyonu kesme noktası koyup, GenyMotion öykünücüsü (emulator) üzerinde hata ayıklama (debugging) yapmaya başladıktan kısa bir süre sonra bu fonksiyon ile gerçekleştirilen AES şifrelemesinin anahtarını, IV (initialization vector)'yi, şifreli verileri ve çözülen verileri (tor adresi gibi) kolaylıkla elde edebildim.


workspace - Debug - mobilsube.apk/Sources/com.android.mobile.Yardimcilar/i.jimple - CodeInspect Demo
File Edit Navigate Project Run Search Android CodeInspect Window Help
Debug
Thread [main] (Suspended (breakpoint at line 252 in i))
Variables
Breakpoints
Name
Value
this
i (id=270)
$param1
QKSMSApp (id=56)
$param0
"CoUsmPiB3jGdjxnXKYLdBMa5LahtMkd/H/U0+NFnqvI=" (id=271)
Outline
com.android.mobile.Yardimcilar.i
byte[] llliiiii
Cipher iiiiiiii
byte[] iiiiiiii
void <init>()
String iliiliii(int)
String iliiliii(String)
String iliiliii(String, int)
String liiiliii(String, Context)
String iliiliii(String, Context)
String iliiliii(String, String, CryptLib$EncryptMode, String)
Jimple
Java
LogCat
Saved Filters
All messages (no filters)
Search for messages. Accepts Java regexes. Prefix with pid:, app:, tag: or text: to limit scope.
verbose
Time
PID
TID
Application
Tag
Text
Console
Loaded Classes
Code Injection
Problems
Android - mobilsube.apk - debug
com.android.mob: eglSurfaceAttrib not implemented
com.android.mob: Failed to set EGL_SWAP_BEHAVIOR on surface 0xdce3a540, error=EGL_SUCCESS
com.android.mob: Background sticky concurrent mark sweep GC freed 2038(124KB) AllocSpace objects, 0(0B) LOS objects, 9% free, 2MB/2MB


workspace - Debug - mobilsube.apk/Sources/com.android.mobile.Yardimcilar/i.jimple - CodeInspect Demo
File Edit Navigate Project Run Search Android CodeInspect Window Help
Debug
Thread [main] (Suspended (breakpoint at line 267 in i))
Variables
Breakpoints
Name
Value
this
i (id=270)
$param3
"f9ppemOO_RiezPM0" (id=72)
$param0
"CoUsmPiB3jGdjxnXKYLdBMa5LahtMkd/H/U0+NFnqvI=" (id=271)
$param1
"b16920894899c7780b5fc7161560a412" (id=77)
$param2
CryptLib$EncryptMode (id=78)
Outline
com.android.mobile.Yardimcilar.i
byte[] llliiiii
Cipher iiiiiiii
byte[] iiiiiiii
void <init>()
String iliiliii(int)
String iliiliii(String)
String iliiliii(String, int)
String liiiliii(String, Context)
String iliiliii(String, Context)
String iliiliii(String, String, CryptLib$EncryptMode, String)
Jimple
Java
LogCat
Saved Filters
All messages (no filters)
Search for messages. Accepts Java regexes. Prefix with pid:, app:, tag: or text: to limit scope.
verbose
Time
PID
TID
Application
Tag
Text
Console
Loaded Classes
Code Injection
Problems
Android - mobilsube.apk - debug
com.android.mob: eglSurfaceAttrib not implemented
com.android.mob: Failed to set EGL_SWAP_BEHAVIOR on surface 0xdce3a540, error=EGL_SUCCESS
com.android.mob: Background sticky concurrent mark sweep GC freed 2038(124KB) AllocSpace objects, 0(0B) LOS objects, 9% free, 2MB/2MB



Kıssadan hisse, günümüz Android zararlı yazılımlarını açık kaynak kodlu, ücretsiz araçlarla analiz etmek her geçen gün daha da zorlaşırken, kullanımı IDA Pro’dan çok daha kolay ve oldukça stabil çalışan CodeInspect gibi ticari araçlar sayesinde parayı veren düdüğü kolaylıkla çalabiliyor ve hayattaki, analizdeki en değerli şey olan zaman, yanınıza kar kalıyor. Ticari hata ayıklama aracı satın alma konusunda benim gibi gözünü karartmış olanlara CodeInspect’in muadili, betik desteği de olan JEB aracına da bir göz atmalarını da tavsiye ederim.

Bir sonraki yazıda görüşmek dileğiyle herkese güvenli günler dilerim.