Digital Whisper
Technologic Papers

iBanking מבצרת עמדות

מאת עמי קאופמן, מנתח מודיעין איומי ב-RSA

הקדמה

בעולם פשעי הסייבר צריך תמיד להזהר. הגנבים עושים מאמצים כבירים כדי לחמוק מעיניהם הבוחנות של אנליסטים ורשויות החוק, זאת כדי למנוע את סגירת המיזם שלהם או גרוע יותר - מעצר. המפתחים של הסוס הטרויאני iBanking - אפליקציה זדונית הפוגעת במכשירי אנדרואיד - אינם יוצאים מן הכלל. בדומה לתוכנות דומות הפוגעות במחשבים האישיים ומיישמות מנגנוני הגנה כגון ערפול קוד (Obfuscation) ושימוש בהצפנה חזקה, גם בגרסאות האחרונות של iBanking נלקחו צעדים להקשחת התשתית של התוכנה הזדונית והפעלתה.

למרות שקוד המקור של התוכנה הודלף לפני מספר חודשים, האפליקציה עדיין נמצאת תחת פיתוח מתמשך וכיום מציגה יכולות חדשות, כגון מעקב אחר כל האפליקציות המותקנות על המכשיר הנגוע, איסוף תמונות ומידע על מיקום גיאוגרפי מדויק. כדאי לציין גם את התמיכה הגדלה המאפשרת תקיפת יעדים נוספים: הניתוח האחרון שלנו זיהה כ-30 שבלונות "גרפיות" ל-iBanking הדומים לאפליקציות או אתרים של גופים פיננסיים שונים. אך החידוש המעניין ביותר הוא ללא ספק השימוש במנגנון הגנה העצמית כגון קידוד AES, בערפול קוד ובאנטי-SDK/VM.

מנגנוני הגנה עצמית

הניסיונות הגוברים של מפתחים להמנע מגילוי הם חלק טבעי באבולוציה של תוכנות זדוניות. ככל שמנתחי האבטחה מפתחים את שיטות הניתוח וגילוי שלהם, מפתחי התוכנות הזדוניות ממהרים להפעיל צעדי מנע להגנה על פעילותם ושמירה על חשאיותם. בדומה לקידוד FUD בסוסים הטרויאנים הפוגעים במחשבים אישיים, iBanking כיום משתמש בשיטות הצפנה חזקות יותר כדי להגן על קוד המקור שלו מניסיונות הינדוס חוזר וגילוי על ידי אנטי-וירוס. האפליקציה גם מיישמת הגנת אנטי SDK כדי להתחמק מסביבת הניתוח בארגזי חול. כמו בכל דבר הקשור למובייל, התפתחויות אלו התרחשו מהר מן הצפוי.

קידוד AES

הניסיון הראשון של iBanking בהגנה עצמית הופיע כקידוד AES. כדי להסתיר את המקורות השונים שלו, iBanking השתמשה במפתח פרטי בקידוד קשיח (בתוך האפליקציה) שקידד את התוכן של קבצי XML ושל מקורות התקשורת. קבצי XML מכילים מידע הקשור למקורות חיצוניים כגון תמונות, אך גם מידע הקשור להגדרות האפליקציה. כפי שנאמר, האפליקציה גם עשתה שימוש בהצפנה עבור מקורות התקשורת שלה, כולל כתובות URL ומספרי טלפון לשליטה.

```
<resources>
    <string-array name="adm_domens">
        <item>4338aa1dc5facbdd4ec0b8be27e4ccb4</item>
        <item>6388995b4c309de03aa68ac12c9b4ff5</item>
        <item>942a7d4f210ad5e1db0c015a4726771e</item>
        <item>11412065eb093a3ade1c4825b64e6d16</item>
        <item>22a5ed85e999ace9c23a870822fda2a3</item>
        <item>9e2d6f1bd13eec8af0c19ef85e16ae55</item>
        <item>0d4a2ebf7fd4b5659a83e951d1aea77b</item>
        <item>4af46d70be43642f4a2bb74c44913ae2</item>
        <item>30679f21a7676ec32650a119b8bb5180</item>
    </string-array>
</resources>
```

[מקורות תקשורות בקידוד AES, מקור: https://blogs.rsa.com/ibanking-mobile-bot-raising-shields/untitled-37]

ערפול קוד האפליקציה

השלב הבא בניסיונות ההתחמקות היה ערפול הקוד. לאחר שעקבנו מקרוב אחד הודעות המפתח בפורומים מחתרתיים, גילינו גרסה חדשה של iBanking שבמבט ראשון נראה כמלא שגיאות שגרמו לקריסתה. הניתוח שביצענו הראה בסיס-קוד ותשתית חדשים לגמרי.

לאחר שבוצע הינדוס-לאחור (Reverse Engineering) של האפליקציה, גילינו הפתעה לא נעימה. במאמץ לבצר עוד יותר את האפליקציה מההינדוס החוזר, המפתח השתמש בערפול קוד כדי להקשות על ניתוח הקוד. לאחר היישום, ערפול הקוד הגדיל את מספר קבצי ה-JAVA מ-23 ל-245 והקצה שמות אקראיים לקבצים החדשים. יתרה מכך, הערפול החליף את שמות המשתנים הסטטיים למחרוזות חסרי משמעות ולערכי מחרוזת מקודדים. כפי שניתן לראות בתמונה מטה, המערפל חכם מספיק דיו להמנע מהצפנה/ערפול משתני מערכת, כגון "app_name". הצפנת המחרוזות נעשתה באמצעות פונקציה בקידוד קשיח ופשוטה יחסית.

```
<resources>
    <string name="app_name">[illegible]</string>
    <string name="utUVZaTWoQ">6</string>
    <string name="dagbratrtnyuty">[illegible]</string>
    <string name="JjbfUWtbOR">1</string>
    <string name="bmmNElKMOO">FFEE5FEBC097101AD78A2381F607</string>
    <string name="MsvMEJgwRE">FE2265</string>
    <string name="update_min">1</string>
    <string name="apUAntNxwl">E0B6D6D170CFE8E96CE56160F971FAFCC6454D860C276D</string>
    <string name="vgIOotMcYz">E0B6D6D170CFE8E96CE56160F971E0FA21D03700E8559BE731</string>
    <string name="lLSZyNJtjo">E0B6D6D170CFE8E96CE56160F971E0E2923AC900E283</string>
    <string name="GCGozNIYKB">E0B6D6D170CFE8E96CE575483484266AA800009ABC</string>
    <string name="tzhrUSWTQa">E0B6D6D170CFE8E96CE5616893968F0DA4E617039911</string>
    <string name="LHlygDlcmY">E0B6D6D170CFE8E96CE56160F971E36F7F07A4EA1E7D</string>
    <string name="ZAZvjhVBrX">E0B6D6D170CFE8E96CE5719CF7DE05C0B12CA8EB8A81</string>
    <string name="qOipuunKuM">53050</string>
    <string name="VbcNcDrQBW">811C40608077EE001D36F9130FD1F1</string>
    <string name="action_settings">Settings</string>
```

[משתנים שעברו ערפול והערכים המקודדים החדשים, מקור: https://blogs.rsa.com/ibanking-mobile-bot-raising-shields/untitled-38/]

מנגנון אנטי-SDK

למרות זאת, אותו ערפול מדובר לא סיפק הסבר המניח את הדעת להימצאות השגיאות באפליקציה ולקריסה. תהליך דה-באגינג רחב של האפליקציה חשף מנגנון הגנה אנטי- SDK מחוכם, שניתן להשוות למנגנון הקיים נגד ארגזי חול בתוכנות הזדוניות הפוגעות במחשבים אישיים.

הניתוח שלנו חשף כי בשלבים המוקדמים של הפעלת האפליקציה היא מריצה פונקציה המשווה מזהים ייחודיים שנאספו מהמכשיר הנגוע עם ערכים בקידוד קשיח. התאמה עם אחד מערכים אלו "יגלה" לאפליקציה שהיא מופעלת בתוך מכשיר וירטואלי ותגרום להפסקה מיידית של פעילותה. אנו זיהינו את ארבעת הערכים הבאים:

1. האם EMEI המכשיר שווה "000000000000000"?
2. האם מספר הטלפון מתחיל ב-"1555521"?
3. האם המפעיל הוא "Android"?
4. האם המספר הסידורי של ה-SIM שווה ל-"89014103211118510720"?

אם התשובה לאחת משאלות אלה היא "כן", האפליקציה מפסיקה את פעילותה באופן מיידי ומדמה קריסה. הערכים בקידוד קשיח מתכתבים לערכי ברירת המחדל המסופקים בדרך כלל על ידי SDK של אנדרואיד ומיושמים באופן שכיח בסביבות לניתוח אפליקציות.

```
SharedPreferences var3 = PreferenceManager.getDefaultSharedPreferences(this.getApplicationContext());
String var4 = ((TelephonyManager)this.getSystemService("phone")).getDeviceId();
if(this.getResources().getString(2131034115).equals("1")) {
   label141: {
      if(!var4.equals("000000000000000")) {
         TelephonyManager var77 = (TelephonyManager)this.getSystemService("phone");
         String var78 = var77.getLine1Number();
         String var79;
         if(var78 != null && !var78.toString().trim().isEmpty()) {
            var79 = var78;
         } else {
            var79 = var77.getSubscriberId();
         }

         if(!var79.startsWith("1555521") && !this.c().equals("Android") &&
            !((TelephonyManager)this.getSystemService("phone")).getSimSerialNumber().equals("89014103211118510720")) {
            break label141;
         }
      }

      Process.killProcess(Process.myPid());
   }
}
```

[פונקציית השוואת אנטי SDK - ניסיון למנוע ניתוח דינאמי, מקור: https://blogs.rsa.com/ibanking-mobile-bot-raising-shields/untitled-39]

סיכום

מפתחי התוכנות הזדוניות למחשבים אישיים, בהיותם ערים לנוכחות גוברת של מנתחי אבטחה, מיישמים מזה זמן מנגנונים שונים נגד גילוי וניתוח התוכנות שלהם. אך שיטת פעולה זו לא הייתה ההנורמה עד היום בתוכנות זדוניות לנייד. תוכנת iBanking מראה כי מפתחי התוכנות הזדוניות נהיו ערים גם לצורך להגן על המכשירים הניידים הנגועים שלהם. ייתכן כי תופעה זו מסמנת טרנד חדש בזירה המתפתחת של תוכנות זדוניות לנייד.

הניתוח המתמשך שלנו ל-iBanking חושף תוכנה זדונית מפותחת ובוגרת הפוגעת במכשירי אנדרואיד. אנו עדים לכך כי Botnets רבים החלו להשתמש בה על מנת להשתלט על סמארטפונים, וזוהי מגמה שאנו ממשיכים לעקוב אחריה.