The Fun Part of Android

מאת ניר גלאון

הקדמה

מאמר זה יסביר בצורה עמוקה יחסית על המערכת אנדרואיד, כיצד היא עובדת, איך קבצי Apk בנויים (תמונות, מחרוזות, וחלקי smali), איך ניתן לשנות קבצי Apk, ולבסוף, איך הם נראים מנקודת המבט של המפתח.

מטרת המאמר אינה להסביר כיצד לפתוח קבצי Apk, וכדי לא להאריך את המאמר יתר על המידה, ולכן, ברשותכם, אפנה אתכם לפוסט שכתבתי המסביר בפירוט כיצד לבצע את זה, בעזרת התוכנה Apktool.
לחצו כאן לכניסה לפוסט.

בשביל שנהיה באותו קו, אציין כי החבילה (האפליקציה) שעליה אעבוד בהמשך היא של החייגן, תוכלו להוריד את שלי מכאן.

נתחיל מהבסיס, איך אנדרואיד באמת עובדת?

ראשית נבדיל בין המונחים. כולם מכירים את התרשים המפורסם שבו מפורטים השכבות השונות (הספריות) של המערכת, אך זו לא הדרך שבה היא עובדת (לא בדיוק בכל אופן), אלא הדרך שבה היא בנויה. אם נעלה ונסתכל על עבודת המערכת ממבט הציפור, נבין שהיא עובדת בצורה שונה ממערכות אחרות (לא ממציאה את הגלגל, אך יש פה כמה הברקות).

כדי להבין כיצד היא עובדת יש צורך להבין כיצד קבצי APK עובדים.

למה הכוונה? אנדרואיד כמערכת הינה חבילה של שורות קוד המריצות אפליקציות (או קבצי Apk).
החייגן שלנו הוא אפליקציה (קובץ Apk) שבמקרה הזה, מצורף באופן מובנה לקוד המקור של אנדרואיד.
כאפליקציה אנחנו יכולים גם להסיר אותו (כדי לא ליצור בלאגן גוגל הגדירה את האפליקציות המובנות כ'לא ניתנות למחיקה', אך אין זה אומר שהן באמת לא ניתנות למחיקה, עם ההרשאות המתאימות-תוכלו לבצע הכל). כתוצאה מכך ניתן לבצע שינויים בכל דבר במערכת, לדוגמה: ניתן להסיר את החייגן המובנה ולהתקין במקומו אחד אחר מה-Play (לדוג' Skype).

איך קבצי APK בנויים?

אני לא אתווכח אם המושג אפליקציה הוא נכון או לא נכון, אך בשביל להבין עדיף לקרוא לאפליקציות קבצי APK, כי מה שהם בפועל זה חבילות (Packages), והחבילות האלו יכולות להכיל 4 דברים:

1. Activity - אפליקציה מורכבת מהמון מסכים (Activities) שמשתמשים עוברים בינהם (ולרוב כל מסך מבצע פעולה שונה) המסך הזה נקרא Activity (ברבים: Activities).

כותב האפליקציה בעצם כותב כמה וכמה Activity ועל ידי כפתורים שונים המשתמש מבצע מעבר בינהם ובמעבר מאפליקציה לאפליקציה אתם עוברים מ-Activity ל-Activity.

דוגמה להבנה: נכנסתם לג'ימייל (נכנסתם ל-Activity בתוך ה-Package שנקרא gmail), משם עברתם לדואר הנכנס (עברתם ל-Activity חדש בתוך ה-Package שנקרא gmail), משם נכנסתם לאיזשהו מייל (עברתם לעוד Activity בתוך ה-Package של gmail), ובתוך המייל שכתבו לכם יש כתובת ולחצתם עליה (ומשם עברתם ל-Activity בתוך ה-Package שנקרא maps), וראיתם שבכתובת שבמפה יש מידע נוסף, לחצתם עליו והוא קישר אתכם לויקיפדיה (אז עברתם ל- Activity בתוך ה-Package שנקרא chrome).

- ה-Activity (המסך) מורכב (לרוב) על ידי כפתורים, תיבות טקסט, תמונות וכד', את ה-Activity מעצבים באמצעות מסמך XML שמאפשר לנו לתאר כיצד המסך יראה והיכן יוצב כל רכיב. בעת מעבר מ-Activity אחד למשנהו (לדוג' בתוך אפליקציה-חבילה) ה-Activity הראשון נעצר וה-Activity השני מתחיל (אבל המערכת שומרת את ה-Activity הקודם במחסנית FIFO - האחרון שנכנס הוא הראשון שיוצא).

2. Service - ה-Service עובד ברקע, ולהבדיל מה-Activity אין לו ממשק משתמש (הוא מתואר על ידי שורות קוד בלבד ושקוף למשתמש). ה-Service עוזר לנו לתת שירות לאפליקציה שלנו באה לתת מאחורי הקלעים.

דוגמה להבנה: הדוגמה הכי קלה היא חבילת (אפליקציית) המוסיקה, חבילת המוסיקה מורכבת מכמה Activity, אנחנו עוברים בין השירים ובוחרים שיר להשמעה, כנראה שלאחר מכן גם עוברים ל-Activity חדש שבו ניתן להריץ את השיר אחורה-קדימה, להפסיק, להפעיל וכד'. אבל מה קורה כשאנחנו יוצאים מה-Activity (מהמסך של המוסיקה, ועוברים לדוגמא ל-Activity של משלוח הודעה)

אך רוצים שהמוסיקה תמשיך לנגן ברקע. על המצב הזה ה-Service עונה. ה- Service הינו רכיב הפועל ברקע ובא לתת לנו עזרה כדי לספק שירות כלשהו למשתמש, כמו להמשיך לנגן מוסיקה.

3. Content provider - ה-Content provider מנהל גישה למערך הנתונים של החבילה (האפליקציה) בין תהליך אחד לשני. כאשר מפתח רוצה לגשת לנתונים של אפליקציה אחרת הוא משתמש באובייקט שנקרא ContentResolver, הוא שולח בקשות ל-Content provider שמנגד מקבל את הבקשה, מבצע את הפעולה ומחזיר את התוצאה.

דוגמה להבנה: אנשי הקשר, אנו יכולים לפתח יישום שניגש ל-Content provider של אנשי הקשר על מנת לגשת לנתונים של החבילה (במקרה הזה, רשימת אנשי הקשר שלנו) ולבקש ממנו לקרוא את הרשימה, לשנות אותה וכד'.

4. Broadcast receiver - הרכיב הזה מאפשר למפתח להגיב לאירועי מערכת שונים. מערכת האנדרואיד משחררת הכרזות שונות במהלך פעולתה, לפעמים ההכרזות האלו מיועדות ל-Filter Intent מסויים ולפעמים הן פשוט "נזרקות לאוויר". הרכיב הנ"ל מאפשר לנו "להקשיב" להכרזות הנזרקות במערכת ולבצע פעולות שונות בהתאם להכרזות.

דוגמה להבנה: אני מניח שרובכם מכירים את האפליקציה שמקפיצה חלון Popup קטן ברגע שאנו מקבלים הודעת SMS, האפליקציה הזאת משתמשת ב-Broadcast receiver, היא מאזינה כל הזמן למערכת וברגע שהמערכת מודיע שקיבלה הודעת SMS האפליקציה נכנסת לפעולה ומבצעת כמה פעולות על מנת להקפיץ לכם הודעה על המסך שתציג את תוכן ה-SMS.

איזה אפליקציה נפתחת, מתי, למה ואיך?

כדי להבין קצת יותר לעומק איך אנדרואיד עובדת, אנחנו צריכים להבין למה אפליקציה אחת נפתחת ולא השניה וכיצד עובד המעבר בין Activity של חבילה-א לחבילה-ב.

<u>אז קודם כל, למושגים:</u>

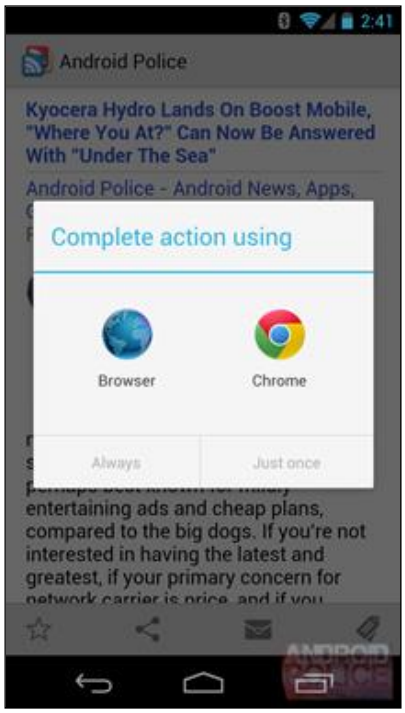
Intent ו-Intent Filter - כל חבילה (אפליקציה) גדולה יחסית תכיל לרוב Activity, Service ו- Broadcast receiver, אלה שלושת מרכיבי הליבה של כל חבילה. שלושת אלה מופעלים באמצעות הודעות הנקראות intents. אובייקט מסוג Intent מייצג כוונה של המשתמש או של מערכת ההפעלה. קיימים 2 סוגים של Intent, הסוג הראשון הוא Intent שמכוון באופן ברור ל-Activity או ל-Component (רכיב חומרתי) ספציפי ונקרא Explicit Intent.

הסוג השני הוא Intent שלא מכוון באופן ברור ואפשר להגיד שהוא פשוט "נזרק לחלל האוויר". במקרה כזה המערכת (Intent Filter) תאתר את ה-Activity (או את הרכיב החומרתי) אליו הוא מכוון בעזרת המאפיינים של ה Intent (שהינם Action, Category ו-Data).

- השימוש הנפוץ ביותר של Intent הינו מעבר בין Activitys (מסכי UI בתוך החבילה).

<u>להלן סיטואציה:</u> במכשיר שלנו יש 2 חייגנים (החייגן המובנה, ו-Skype) ואנו נמצאים ב-Activity של אנשי הקשר, ורוצים להתקשר לחבר. בלחיצה על איש הקשר ולאחר מכן על המספר שלו, עלה לנו חלון קטן שמבקש מאיתנו לבחור דרך איזה אפליקציה אנו רוצים לחייג לאיש הקשר (דרך החייגן המובנה או דרך ה-Skype), למה זה קורה?

כשאנו לוחצים על המספר של איש הקשר אנו שולחים Intent "לחלל האוויר" (ה-Intent לא מיועד באופן ספציפי לאף Activity או רכיב חומרתי. ה-Intent Filter מאתר את ה-Activity (או הרכיב החומרתי) אליו הוא מיועד לפי המאפיינים שהזכרנו לעיל (Action, Category ו-Data).

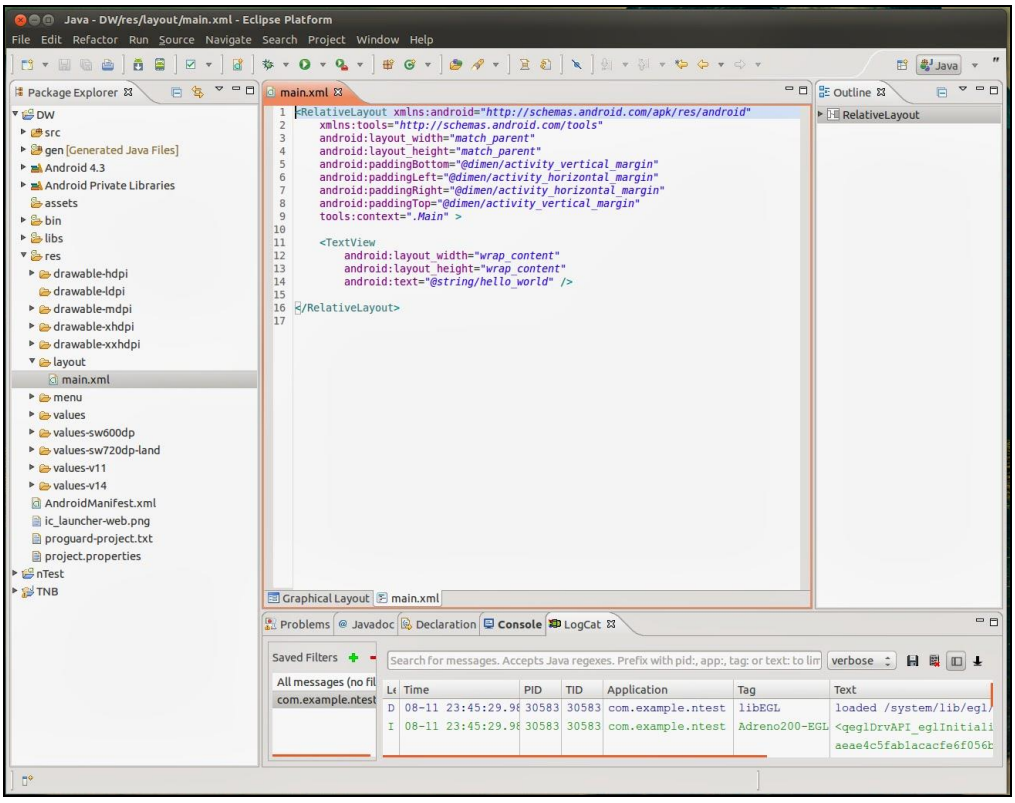


במקרה שלנו: הפעולה (Action) הינה ביצוע חיוג / התקשרות. הקטגוריה (Category) הינה חייגנים. והמידע (Data) הינו המספר שאליו אנו נחייג.

ה-Intent Filter מזהה 2 אפליקציות (חבילות) שעונות על המאפיינים האלו ויכולות להתמודד עם המשימה (החייגן המובנה וה-Skype), והוא נמצא בבעיה, יש 2 אפשרויות, במי לבחור? למערכת אין העדפה, ולכן קופצת לנו הודעת Popup קטנה שמבקשת מאיתנו לבחור כיצד לחייג.

מנקודת המבט של המפתח

בשביל להבין איך החבילות האלו בנויות מנקודת המבט של המפתחים, בניתי אפליקציה פשוטה שמורכבת מ-Activity (מסך אחד). בצד שמאל יש לנו את האקספלורר ובו כל התיקיות והקבצים.



- **התיקייה src:** התיקייה מכילה את כל הקוד וקבצי המקור שפותח בחבילה, בשפת JAVA.

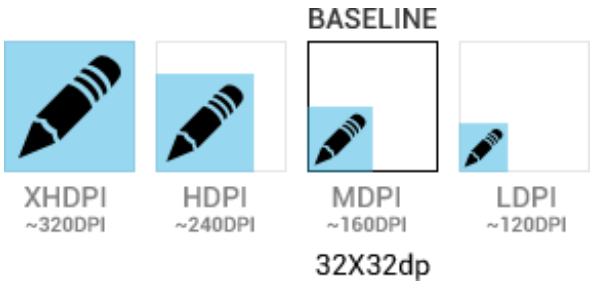
- **התיקייה gen:** המחלקה R היא מחלקה שנוצרת באופן אוטמטי ע"י הפלאגין ADT הכתובה בשפת JAVA (מג'ונרטת באופן עצמאי, אוטומטי). אם נפתח אותה, נראה המון תת מחלקות לפי הקבצים שייצרנו בפרויקט / חבילה שלנו.

- **התיקייה android 4.2:** התיקייה הזאת מכילה את כל החבילות (האפליקציות) המובנות שיש באנדרואיד ואיתן כל המחלקות והשיטות שלהן. כך המפתח לא צריך ליצור כל דבר, אלא מבצע שימוש במחלקות והשיטות הקיימות.

- **התיקייה Assets:** בתיקיית ה'נכסים' משתמשים כדי לאחסן משאבים כדוגמת פונטים (במידת הצורך), קבצי וידיאו, קבצי קול וכו'.

Resources:

- **התיקיות "drawable-hdpi / ldpi / mdpi / xhdpi"** הינן תיקיות שיכילו את כל התמונות שהאפליקציה צריכה. מדובר על אותן תמונות אך ברזולוציות שונות. ולמרות שקיימים עוד המון סוגי רזולוציות באנדרואיד, אלה הבסיסיות ובמידה ולא יצרנו תיקייה לרזולוציה ספציפית, כשהמשתמש יפעיל את האפליקציה המערכת תידע לבחור את התמונה בה הרזולוציה המתאימה ביותר ותתאים את התמונה למסך של המשתמש.



- **בתיקיית "layout"** נמצא קובץ ה-Activity (המסך) שיצרתי, ומתוארים על ידי קבצי XML. קבצי ה-XML שנמצאים ב-layout בעצם מגדירים את סידור המסך.

- **תיקיית ה-"Menu"** מכילה קבצי XML של התפריטים. מי שמשתמש באנדרואיד מכיר את כפתור השלוש נקודות המסמל את מקש האופציה / אפשרויות. אז בדיוק על זה מדובר, פה המפתח יגדיר את המקש הזה והאופציות שלו.

- **תיקיית ה"values"** מכילה קבצי XML עם ערכי מחרוזות. בעצם כל מחרוזות הטקסט שבאפליקציה, ובכך עבודת התרגום לשפות השונות הופכת לקלה יותר מכיוון שהשפות לא צמודות לUI (במילים אחרות: לא Hard coded).

- **AndroidManifest.xml:** הקובץ "AndroidManifest.xml" מכיל את כל המידע הרוולנטי על החבילה: החל משם החבילה (חבילת ה-JAVA) של האפליקציה (המשמש כמזהה יחודי לאפליקציה ב-Play), תיאורים של הרכיבים מהם האפליקציה בנויה (ה-activities, services, broadcast receivers, content providers), הסברים על המחלקות והמתודות השונות שהאפליקציה יכולה לבצע (לדוגמה ה-Intent שאיתם האפליקציה יכולה להתמודד), הצהרה על גרסת אנדרואיד מינימאלית שעליה האפליקציה יכולה לרוץ, פירוט של הספריות בהם האפליקציה עושה שימוש, ולבסוף הצהרות של ההרשאות שהאפליקציה (חבילה) צריכה (ובמידת הצורך גם ההרשאות שאפליקציות אחרות צריכות על מנת לעבוד מולה).

Take a look inside

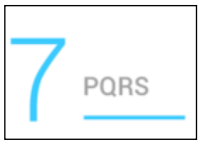
אחרי שפתחנו את הקובץ נקבל תיקייה בשם Phone (בחרתי לקרוא לתיקייה בשם המקורי של החבילה), אך פה נראה שהקובץ שבנוי בצורה קצת שונה ממה שראינו לעיל (מהזווית של המפתח), עכשיו אנו רואים את הפרויקט / חבילה בתצורה הסופית שלה (לאחר שעברה קימפול ע"י הקומפיילר של ה-Dalvik מה-bytecode, שהגיע כתוצאה מהקימפול של Java code).

בואו נתחיל לשחק:

נפתח את ה-Activity של החייגן, אז כמו שאמרנו ה-Activity יימצא בנתיב res/layout, ונפתח את הקובץ dialpad.xml. להלן השורה של הלחצן 7:

```
<ImageButton android:id="@id/seven"
android:src="@drawable/dial_num_7_wht"
android:contentDescription="@string/description_image_button_seven"
style="@style/DialpadButtonStyle" />
```

ואת התמונה של הכפתור לקחתי מהתיקייה drawable-hdpi וניתן לראות (לפי ההפניה ב XML שלעיל) שהשם שלה הוא dial_num_7_wht (להלן התמונה של הקובץ).



כמו שאמרנו, מחרוזות הטקסט באפליקציה נכתבות במסמך שונה, בקובץ XML בשם strings, וקובץ זה יהיה בתיקייה values (לכל שפה ישנה תיקיית values משלה, בצירוף קוד המדינה ב-2 תווים). ה- strings.xml בשפה העברית יהיה בתיקייה values-iw.

להלן חלק מתוכן הקובץ (בירוק מסומן התיאור של הלחצן 7).

```
<string name="description_image_button_five">חמש</string>
<string name="description_image_button_six">שש</string>
<string name="description_image_button_seven">שבע</string>
<string name="description_image_button_eight">שמונה</string>
<string name="description_image_button_nine">תשע</string>
<string name="description_image_button_star">כוכבית</string>
<string name="description_image_button_zero">אפס</string>
<string name="description_image_button_pound">סולמית</string>
<string name="description_dial_button">חייג</string>
```

ובדרך זו אנו יכולים לבצע עוד מגוון שינויים, כגון: לשנות את הרקע של האפליקציה, לשנות את האייקונים והתמונות השונות, אם האפליקציה אינה בעברית - לתרגם או לשנות את התרגום הקיים וכד'.

בנוסף, יש לציין כי במערכת עצמה ניתן לשנות מגוון נוסף של דברים:

- להוסיף / לשנות פונטים, הנמצאים בנתיב: /system/fonts .
- התראות מכשיר, רינגטונים, צלילי מערכת (הקליק של המצלמה, סוללה עומדת להיגמר, חיבור לדוק וכד') בנתיב: /system/media/audio .
- אנימציית ההפעלה של המערכת, נמצאת בנתיב: /system/media והעריכה שלה הינה פעולה פשוטה ביותר (להסבר מפורט יותר: לחצו כאן).
- אם רוצים שאפליקציה מסויימת לא תהיה ניתנת למחיקה (כמו אפליקציות מערכת), פשוט תעבירו את הקובץ Apk שלה לנתיב: /system/app. (אפליקציות רגילית מותקנות בתיקייה DATA/app בעלת ההרשאה rwxrwx—x, בעוד לתיקייה בנתיב system/app יש הרשאות rwxr-xr-x).
- אם רוצים לשנות את כפתורי המגע (במכשירי הנקסוס), אלה הן תמונות הנמצאות ב-SystemUI.apk. (להסבר מפורט יותר: לחצו כאן).

The interesting part

כשפתחנו את ה-Apk היו לנו בתיקייה כמה קבצים ותיקיות, אחת מהתיקיות המעניינות היא תיקיית ה-smali, שם נמצא כל הקסם. Smali אלה קבצי bytecode (לא bytecode של JAVA, אלא bytecode של dalvik). בשפת הסף המתאימים ל-Dalvik (המכונה הוירטואלית של אנדרואיד), (ניתן להגיד שהם המקבילים ל-assembly).

מכיוון שאפליקציית החייגן גדולה ומורכבת, בואו נפריד את הקובץ DW.apk שבניתי בשביל ההדגמה מהסעיף הקודם ("מנקודות המבט של המפתח"), זהו קובץ פשוט שלא עושה כלום חוץ מלהציג Activity אחד ובו רשום Hello world!, את הקובץ ניתן להוריד מכאן.

ניתן לראות שהספרייה smali מכילה תיקיות משנה המגדירות את את המזהה היחודי, אצלי הוא com.example.dw. ניכנס לתיקייה smali>com>example>dw.

בתיקייה dw ניתן לראות שני סוגים של קבצים, כאלו עם הסימן "$" בשם וכאלו בלי הסימן:

- הקבצים ללא סימן ה-$ הינם class רגיל בשפת JAVA.

- סימן ה-$ בשם הקובץ מסמן כי זוהי מחלקת JAVA **פנימית** בקובץ שבאה לפני ה-$ (כלומר הקובץ R$id.smali הינו מחלקה פנימית, class, בקובץ R בשם id).

הקבצים ללא סימן ה-$ הם הקובץ R.smali והקובץ Main.smali.

הקובץ Main.smali זהו ה-Activity שלנו (שמעוצב באמצעות קובץ XML בנתיב res/layout), והקובץ R.smali זהו קובץ שנוצר באופן אוטומטי וממפה את המשאבים של האפליקציה, ז"א כשהמפתח רוצה לקרוא לדוגמא לכפתור, ל-string, ל-layout, או לתמונה מסויימת (מהתקייה drawable), הוא קורא לו מתוך המחלקה R.

כעת, נסתכל בתוך הקובץ Maim.smali:

```
class public Lcom/example/dw/Main;
.super Landroid/app/Activity;
.source "Main.java"

# direct methods
.method public constructor <init>()V
  .locals 0

  .prologue
  .line 7
  invoke-direct {p0}, Landroid/app/Activity;-><init>()V

  return-void
.end method

# virtual methods
.method protected onCreate(Landroid/os/Bundle;)V
  .locals 2
  .parameter "savedInstanceState"

  .prologue
  .line 11
  invoke-super {p0, p1}, Landroid/app/Activity;->onCreate(Landroid/os/Bundle;)V

  .line 12
  const/high16 v0, 0x7f03

  invoke-virtual {p0, v0}, Lcom/example/dw/Main;->setContentView(I)V

  .line 13
  return-void
.end method

.method public onCreateOptionsMenu(Landroid/view/Menu;)Z
  .locals 2
  .parameter "menu"

  .prologue
  .line 19
```

```
  invoke-virtual {p0}, Lcom/example/dw/Main;->
getMenuInflater()Landroid/view/MenuInflater;

  move-result-object v0

  const/high16 v1, 0x7f07

  invoke-virtual {v0, v1, p1}, Landroid/view/MenuInflater;->
  inflate(ILandroid/view/Menu;)V

  .line 20
  const/4 v0, 0x1

  return v0
.end method
```

נחלק לחלקים את הקוד:

בשלוש השורות הראשונות ישנם הצהרות של ה-class.

משורה 5 עד 14 אנו רואים את המתודה של ה-constructor (הבנאי).

משורה 16 עד 32 רואים את המתודה onCreate (אחראית למה שקורה בעת הפעלת ה-Activity).

משורה 34 עד הסוף רואים את המתודה onCreateOptionMenu (שאחראית על השורה העליונה ב-Activity, הפנאל).

הבנאי מופיע כביכול משום מקום, אך זה בגלל שהמחלקה תמיד תהיה מורחבת (extends) מהמחלקה Activity (ניתן לראות זאת גם בשורה השניה, לפי ההפניה של super ל-Activity), ולכן תירש את הבנאי של Activity. אם נסתכל על המתודה onCreate, נוכל לראות כי היא אינה שונה מהקבילה ב-Java.

המתודה protected, השם שלה הוא onCreate, והיא מקבלת פרמטר בשם sacedinstanceState מסוג Bundle (במידה וישנם כמה פרמטרים, ההפרדה בינהם תתבצע באמצעות נקודה פסיק), ובסוף המתודה מחזירה void.

קל לזהות כי סוגי האוייבקטים המוחזרים מתחילים ב-L וכתובים במרחב המלא. המשתנים לעומת זאת, מופיעים בפורמט הבא:

- **V** - מציין void.
- **Z** - מציין boolean.
- **B** - מציין byte.
- **S** - מציין short.
- **C** - מציין char.
- **I** - מציין int.
- **J** - מציין long (64 ביט).
- **F** - מציין float.
- **D** - מציין double (64 ביט).

אלה המשתנים הבסיסיים, ומי שמתעניין יותר יכול להמשיך לקרוא כאן.

שורה לאחר מכן, אנו רואים "locals." ומספר, שורה זאת נותנת הוראה ל-Dalvik VM בכמה רשומות להשתמש. בנוסף, ה-smali משתמש ב"v" וב"p" עבור רשומות מקומיות או רשומות של פרמטרים (בהתאמה).

ה-opcode של Dalvik די ברורים, אך יש המון כאלה. לכן אציין פה את החושבים ומי שרוצה לראות את הרשימה המלאה יכול להיכנס לכאן.

- {vx, vy, …} nvoke-super - מפעיל את המתודה ב class של ההוראה באובייקט vx ומעביר את הפרמטר/ים vy.
- {vx, vy, …} nvoke-virtual - מפעיל את המתודה הווירטואלית באובייקט vx ומעביר את הפרמטר/ים vy.

נכניס הודעה קופצת (Toast) עם הטקסט Hack, אך נצטרך להכניס אותה ב-smali, והאמת שאין לי מושג איך לכתוב ב-smali, אז להלן השורה ב-Java:

```
Toast.makeText(getApplicationContext(), “Hack”,
Toast.LENGTH_SHORT).show();
```

נכניס אותה לאפליקיה, נקמפל, נייצא, נפתח בעזרת apktool, ונראה את הקוד smali שקיבלנו, הוא אמור להיות כזה:

```
  invoke-virtual {p0}, Lcom/example/dw/Main;-
>getApplicationContext()Landroid/content/Context;

  move-result-object v1

  const-string v2, "Hacked!"

  const/4 v3, 0x0

  invoke-static {v1, v2, v3}, Landroid/widget/Toast;-
>makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/wi
dget/Toast;

  move-result-object v1

  invoke-virtual {v1}, Landroid/widget/Toast;->show()V
```

ונכניס אותו בשורה 33, לאחר ה-"line 13." ולפני ה-"return-void".

(אנו מכניסים את ה-Toast תחת המתודה onCreate, כי אנו רוצים שההודעה תקפוץ ישר כשנפעיל את האפליקציה). **לא לשכוח להעלות את המספר שליד ה-"locals." שכן אנו נשתמש בעוד רשומה**. נשמור, נבנה את האפליקציה מחדש, ונחתום אותה (שוב, חזרו למדריך הזה).

והעבירו את האפליקציה למכשיר, התקינו אותה, ובדקו שזה עובד.

סיכום

לסיום, למדנו איך מערכת ה-Android עובדת מאחורי הקלעים, ראינו איך קבצי Apk בנויים מנקודת המבט של המפתח, פתחנו קבצי Apk, שיחקנו עם קבצי XML, וביצענו קצת הנדסה לאחור (Reverse Engineering) על ידי התעסקות בקבצי smali.

אני מקווה שהצלחתי לחדש, להעשיר בידע, והכי חשוב - שנהנתם!