Digital Whisper
Technologic Papers

מחקר אפליקציות בסביבת אנדרואיד - המשך

מאת עידן שכטר

הקדמה

במאמר הקודם חקרנו יחד את האפליקציה Shazam Lite בסביבת אנדרואיד, במטרה להתגבר על מנגנון המגביל את השימוש באפליקציה לאיזורים מסויימים בלבד. שאלת המחקר שלנו הייתה דיי מדויקת, ושימוש בכלים שהכרנו סייע לנו להגיע אל הפתרון בצורה יחסית פשוטה. במאמר זה נעמיק את ההיכרות עם אותם כלים בכדי לכתוב צ'יטים למשחק בסביבת Android.

האתגר

המשחק Defender II הינו משחק בקטגוריית ה-Tower Defence (אשר באופן אישי יצא לי לשחק בו לא מעט בתור ילד) בו עלינו להגן על חומה מפני מפלצות המנסות להרוס אותה. כאשר מפלצת תגיע לחומה, היא תפגע בה ותוריד לה נקודות חיים. במידה ונאבד את כל החיים - נפסיד את השלב ונצטרך להתחיל מההתחלה.

במאמר זה נשתמש בכלים שהכרנו מהמאמר הקודם בכדי לכתוב צ'יטים למשחק. אמליץ לקרוא את המאמר לאחר היכרות עם הכלי Frida, שימוש בסיסי ב-ADB, והיכרות עם סביבה וירטואלית שמסמלצת מכשיר אנדרואיד (את כל אלה ועוד ניתן למצוא במאמר הקודם).

קישור להורדת ה-APK אותו נחקור. מאחר והמשחק פרוס על המסך בצורה אופקית, אמליץ לשנות את הגדרות המכשיר הוירטואלי בהתאם.

כשנפתח את המשחק לראשונה נגיע לתפריט בעל אופציות מגוונות, כמו שיפור יכולות, רכישת קשתות חדשות ועוד. נראה שיש הרבה מאד נתונים שנוכל לשחק איתם עוד לפני שבכלל נתערב בלוגיקת ה-"משחקיות" נתון מעניין שישר תופס את העין הוא כמות הזהב והיהלומים שהשגנו עד כה:

זהב ויהלומים משמשים אותנו לשיפור יכולות וכשפים ולרכישה של קשתות חזקות יותר, ונוכל להשיג אותם כאשר נסיים שלבים במשחק בהצלחה. היהלומים נחשבים "נדירים" יותר ונקבל הרבה פחות מהם כאשר נסיים שלב, ונוכל להשתמש בהם כדי לקנות את הקשת הכי חזקה במשחק, מה שלא נוכל לעשות באמצעות זהב.

המטרה הראשונה שלנו היא להבין איך נוכל להוסיף לעצמנו זהב ויהלומים כרצוננו.

שאלת המחקר

כדי למצוא דרך לנצל מנגנון כלשהו, עלינו קודם כל להכיר את המנגנון לעומק. ראשית נרצה להבין איך המשחק שומר את המידע על השחקן שלנו, האם המידע על השחקן שלנו נשמר על שרת מרוחק ונמשך על ידי האפליקציה בעת החתברות? אם כן - נוכל להתערב בתהליך על ידי שימוש ב-Proxy ולנסות לשנות את הנתונים כרצוננו!

אם לא, כלומר, המידע נשמר בצורה מקומית, נוכל להבין היכן הוא נשמר ובמידה והוא לא חתום קריפטוגרפית - אולי נוכל לשנותו בהתאם. על שאלה זו נוכל לענות באמצעות בדיקה פשוטה - ננתק את המכשיר \ אימולטור מהאינטרנט ונעלה את המשחק. אם המשחק עולה עם הנתונים שלנו ואינו דורש מאיתנו להתחבר לאינטרנט, המידע שלנו ככל הנראה נשמר מקומית על המכשיר.

נבצע את הבדיקה הזריזה ונגלה כי המשחק אכן עולה עם הנתונים שלנו, והם ככל הנראה נשמרים על המכשיר. כעת נרצה להבין היכן הנתונים האלה נשמרים ובאיזה צורה!

אפליקציות אנדרואיד ברמת ה-File System

כאשר אפליקציה (APK) מותקנת על מכשיר אנדרואיד, היא מקבלת תיקיה מיוחדת המשתמשת אותה, ורק אותה, כדי לשמור קבצים לבחירתה. את התיקיה המיוחדת נוכל למצוא בנתיב:

/data/data/<package>

כך ש-package מסמל את שם החבילה (במקרה שלנו, שם החבילה הוא com.droidhen.defender2) אם נריץ את הפקודה "ls -l" בתיקיה data/data/, נחשף לרשימה ארוכה של תיקיות בעלות שם עם פורמט קבוע, כך שכל תיקיה שייכת ל-user שונה במערכת (תיקיות השייכות למשתמש "system", שייכות לשירותי מערכת שונים).

הערה: מנגנון זה למעשה מייצר סביבת Sandbox עבור כל אפליקציה, כך שרק ליוזר השייך לאפליקציה הרשאות לכתוב ולקרוא את הקבצים שלה. לדוגמא, אם אפליקציית מסרים מידיים כלשהי שומרת הודעות ופרטים של המשתמש בתיקיה המיועדת לה, אף אפליקציה אחרת לא תוכל לגשת אל אותם קבצים ולהיחשף לתוכן שלהם. (יש לציין שמשתמש בהרשאות גבוהות יוכל לגשת לכל תיקיה שיבחר)

ננווט לתיקיית המשחק שהתקנו ונבחן את הקבצים הנמצאים בה:

```
drwx------ 5 u0_a86          u0_a86          4096 2024-05-17 09:56 com.android.traceur
drwx------ 4 u0_a70          u0_a70          4096 2024-05-17 09:55 com.android.vpndialogs
drwx------ 4 u0_a89          u0_a89          4096 2024-05-17 09:55 com.android.wallpaper.livepicker
drwx------ 4 system          system          4096 2024-05-17 09:55 com.android.wallpaperbackup
drwx------ 4 u0_a118         u0_a118         4096 2024-05-17 09:55 com.android.wallpapercropper
drwxr-x--x 4 u0_a107         u0_a107         4096 2024-05-17 09:55 com.android.wallpaperpicker
drwx------ 7 u0_a112         u0_a112         4096 2024-07-13 19:13 com.android.webview
drwx------ 4 u0_a126         u0_a126         4096 2024-05-17 09:55 com.android.wifi.resources
drwx------ 7 u0_a129         u0_a129         4096 2024-07-05 09:23 com.droidhen.defender2
drwx------ 4 u0_a84          u0_a84          4096 2024-05-17 09:55 com.example.android.livecubes
```

```
:/data/data/com.droidhen.defender2 # ls
cache  code_cache  files  oat  shared_prefs
```

את שמות התיקיות שזה עתה ראינו נפגוש בתיקיית כל אפליקציה שנתקין. למעשה, תיקיות אלו מרכיבות יחד את ה"פורמט" , בו נשמרים קבצי האפליקציה (חשוב לציין - לא מדובר בקוד של האפליקציה [עד לכדי מקרים מסויימים, כמו ספריות Native ועוד דוגמאות שעליהן לא נפרט במאמר זה], אלא בקבצי הגדרות, מסדי נתונים, קבצי מדיה ועוד) והן לרוב ישמשו אפליקציות בצורה דומה.

הכתיבה וקריאה לאותן תיקיות בדרך כלל לא מתבצע על ידי פעולות קריאה וכתיבה פרימיטיביות, אלה יונגשו על ידי API אנדרואידי, כפי שנלמד בעמוד הבא.

ממשיכים באתגר

ננווט אל תיקיית shared_prefs ונמצא שם את הקבצים הבאים:

```
:/data/data/com.droidhen.defender2/shared_prefs # ls
appsflyer-data.xml  save3.xml
```

הקובץ "appsflyer-data.xml" פחות מעניין אותנו והוא כלל הנראה מכיל קונפיגורציה הרלוונטית ל-SDK של AppsFlyer שהאפליקציה משתמשת בו. לעומת זאת, קובץ save3.xml נראה מאד מעניין! נציץ בתוכן שלו:

```
:/data/data/com.droidhen.defender2/shared_prefs # cat save3.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <int name="purchaserate" value="50" />
    <long name="purchaseend" value="1721044800" />
    <int name="bowLock2" value="1" />
    <int name="fireCast" value="13" />
    <int name="magicStone" value="12" />
    <long name="purchasenow" value="1720898895" />
    <long name="tapjoystart" value="1392026401" />
    <int name="battleTime" value="24" />
    <long name="tapjoynowLocal" value="1720898897851" />
    <int name="checkDeviceID" value="1" />
    <int name="gold" value="10" />
    <int name="costCoin" value="0" />
    <int name="discountPicTime" value="-1382983993" />
    <long name="purchasenowLocal" value="1720898895514" />
    <int name="help12" value="1" />
    <int name="costStone" value="1" />
    <int name="help13" value="1" />
    <int name="help11" value="1" />
    <int name="exp" value="125" />
    <int name="killMonster" value="390" />
    <long name="tapjoynow" value="1720898897" />
    <int name="equipBow" value="0" />
    <int name="lightCast" value="0" />
    <int name="tapjoyrate" value="100" />
    <string name="playerName">player</string>
    <int name="level" value="3" />
    <string name="checkValuemagicStone">06d8fd4badd2b8c7e9bda7d725014b12</string>
    <int name="iceCast" value="0" />
    <int name="loseGame" value="5" />
    <string name="deviceID">8c0b9903a8ef250a2080961528</string>
    <long name="tapjoyend" value="1581058801" />
    <long name="purchasestart" value="1720767600" />
    <int name="newPackPicTime" value="-2110432394" />
    <int name="stage" value="11" />
    <string name="checkValuegold">6441b542d9c0e2a58049226de55db908</string>
    <int name="help1" value="1" />
    <int name="help7" value="1" />
</map>
```

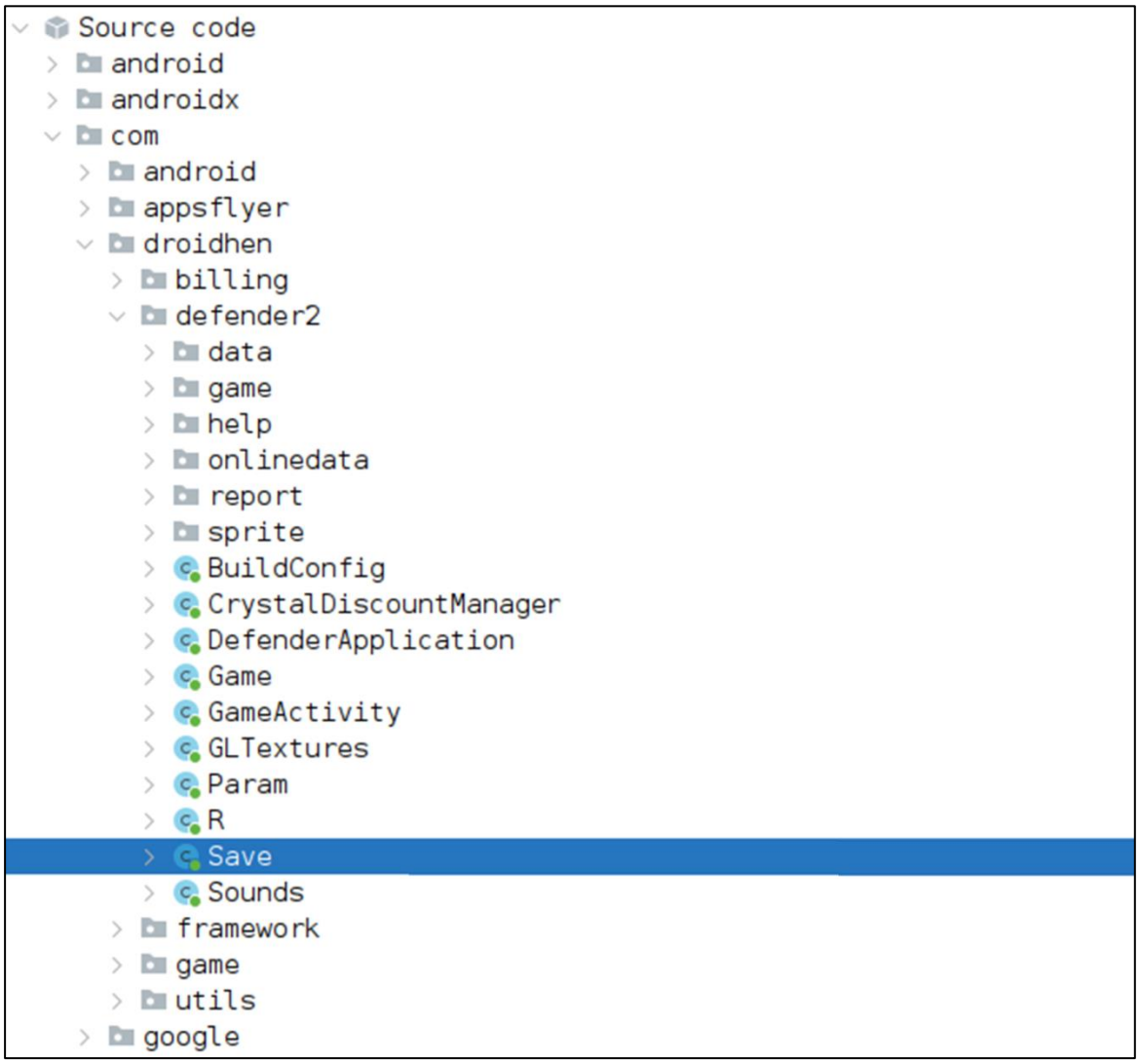
נראה שהקובץ מכיל הרבה מאד ערכים עם שמות מעניינים, הערך תחת השם "gold" זהה לכמות הזהב שיש לנו במשחק! וגם כך הערך תחת השם "magicStone" זהה לכמות היהלומים שיש לנו במשחק. נוכל גם לראות שהשם של השחקן שלנו נמצא בשדה "playerName".

עוד לפני שננסה לשנות ערכים בקובץ, נבחן את הקוד של האפליקציה כדי להבין טוב יותר את השימוש שנעשה באותו קובץ. נפתח את ה-APK באמצעות jadx-gui ונתחיל בעבודה!

נקודה חשובה: ה-APK לא עבר אובפוסקציה ולכן נוכל לראות חלקים רבים בקוד בצורתם הכמעט מקורית - עובדה זו מאד מקלה על תהליך המחקר. כאשר נחקור אפליקציות מתוחזקות ונפוצות, פחות סביר שנמצא את עצמנו במצב הזה.

אגב, במאמר הקודם השתמשנו ב-jadx כדי לבצע decompilation לקוד, ולאחר מכן בחנו אותו באמצעות android-studio, במאמר זה בחרתי להשתמש ב-jadx-gui, החוסך מאיתנו את הצורך להשתמש בתוכנה נפרדת כדי לבחון את הקוד, וכו כן מכיל פיצ'רים נהדרים שיעזרו לנו בהמשך המאמר.

כאשר נעיין בקבצי האפליקציה בעץ הקבצים של jadx-gui, נתקל בקובץ עם שם מעניין:



נשים לב כי במחלקה הממומשת בתוך הקובץ ישנם מספר רב של משתנים סטטיים המחזיקים ערכים המזכירים ערכים שראינו בקובץ save3.xml. נראה שאנחנו במקום הנכון.

במעבר מהיר על קוד המחלקה נשים לב כי רוב הפונקציות המוגדרות בה הן סטטיות, כלומר שהן שייכות למחלקה ולא נצטרך לייצר אובייקט של המחלקה כדי להשתמש בהן. התחושה שלי היא שהמפתח של המשחק התכוון לממש מחלקה בעלת Instance אחד בלבד (Singleton). מה שמרגיש מאד הגיוני כאשר אנחנו חושבים על יישות תוכנתית שהמטרה שלה לשמור ולטעון מידע על השחקן הנוכחי, בייחוד בעיקר כי רק שחקן אחד יכול לשחק במשחק במקביל, ולכן רק שמירה אחת תהיה רלוונטית בכל רגע נתון (או אפילו בכל ריצה של האפליקציה).

קטע הקוד הקצר הזה רומז לנו שהמשחק עצמו אמור לכאורה לתמוך ב-4 שמירות שונות:

```
static {
    int[] iArr = {0, 1, 2, 3};
    SaveFiles = iArr;
    _saves = new SharedPreferences[iArr.length];
}
```

כאמור, SharedPreferences הוא API אנדרואידי המספק ממשק לאחסון מידע קבוע על גבי קבצי XML. קטע קוד זה מרמז לנו שהקובץ "save3.xml" מנוהל על ידי האפליקציה באמצעות SharedPreferences.

SharedPreferences על רגל אחת

<שם באנגלית> הוא ממשק אנדרואידי נפוץ המאפשר אחסון כמויות קטנות של מידע בתצורת key-value. מאחורי הקלעים, SharedPreferences מייצר קובץ xml הנשמר בתיקיית shared_prefs של האפליקציה, ומאשר למפתח לכתוב \ לקרוא ממנו ערכים לבחירתו.

השימוש ב-SharedPreferences מאפשר לאפליקציות לשמור נתונים שונים וחשובים בצורה פרסיסטנטית ולאורך זמן. כמו במקרה שלנו - מפתח עשה שימוש ב-SharedPreferences כדי לעקוב אחר ההתקדמות של המשתמש במשחק.

השימוש ב-SharedPreferences הוא דיי פשוט:

```
// put a value

SharedPreferences sp = getSharedPreferences("file_name", MODE_PRIVATE);
SharedPreferences.Editor editor = sp.edit()

editor.putString("key", "value");
editor.commit();

// read a value

SharedPreferences sp = getSharedPreferences("file_name", MODE_PRIVATE);
String data = sp.getString("password", "")
```

משמעות הדגל MODE_PRIVATE היא שהקובץ שיווצר בתיקיית shared_prefs יוכל להקרא \ להכתב אך ורק על ידי האפליקציה שיצרה אותו. אבל רגע, אמרנו שרק האפליקציה יכולה לגשת לקבצים שלה, אז למה אנחנו צריכים להקשיח את ההרשאות אם הן כבר מוקשחות?

למרות שאין "הבדל משמעותי" מדובר בהרגל בריא. לדוגמא, השימוש בדגל "מסמן" למפתחים אחרים שהקובץ הזה מכיל מידע רגיש ויש לשמור עליו. אפשרות מעניינת נוספת היא שאם בגרסאות עתידיות של אנדרואיד מנגנון ה-sandboxing ישתנה ואפליקציות יוכלו לגשת לקבצים של אפליקציות אחרות, הקובץ עדיין ישאר מוגן.

בהמשך ל-SharedPrefernces, הפונקציה init בעלת החתימה () במחלקה Save מייצרת אובייקט SharedPreferences בצורה מעט שונה ממה שזה עתה ראינו:

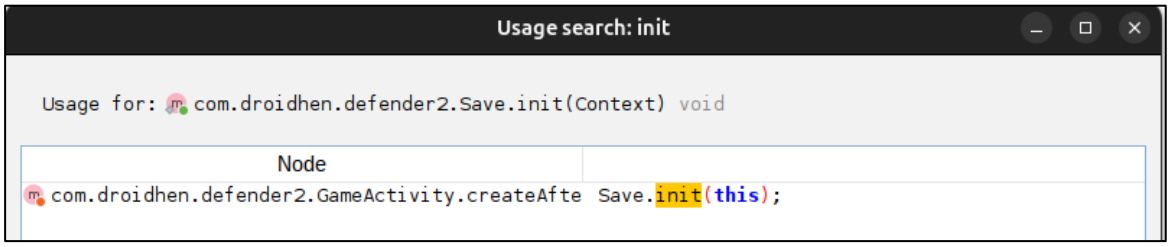
```
private static void init() {
    for (int i = 0; i < SaveFiles.length; i++) {
        _saves[i] = _context.getSharedPreferences("save" + i, 0);
    }
}
```

פונקציה זו נקראת על ידי פונקציה בעלת שם זהה - init, לה החתימה (Context context):

```
public static void init(Context context) {
    _context = context;
    init();
}
```

אז בפועל, פונקציה init אחת מקבלת אובייקט מסוג Context, שומרת אותו במשתנה סטאטי, ולאחר מכן פונקציית ה-init הנוספת משתמשת בו כדי לייצר אובייקט SharedPreferences. מאיפה Context מגיע? נחפש Cross References (או בקיצור X-Refs) לפונקציה init בעלת החתימה (Context context), כלומר נבדוק מי בקוד משתמש בפונקציה הזו.

אם בחרתם להשתמש ב-jadx-gui, סימון הפונקציה עם העכבר ולאחר מכן לחיצה על כפתור X יציג בפניכם את רשימת ה-X-Refs הרלוונטית:




```
Save.init(this);
_glView = (GL2DView) findViewById(R.id.game_view);
hideSystemUI();
_textures = new GLTextures();
Game game = new Game(this, _textures, _handler);
_game = game;
_glView.bindGame(game, _textures);
CrystalDiscountManager.getInstance().updateDiscountRate();
Param.MUSIC_FLAG = Save.loadData(Save.MUSIC_FLAG, 3) == 0;
Param.SOUND_EFFECT_FLAG = Save.loadData(Save.SOUND_FLAG, 3) == 0;
onStartAfterPermissionFinished();
```

```
public class GameActivity extends PurchaseOfflineActivity {
```

הפונקציה מקבלת את this, בקונטקסט הנוכחי הוא אובייקט מסוג GameActivity.

אז מה זה Context?

כשמו כן הוא, Context הוא אובייקט המייצג את המצב הנוכחי של האפליקציה ומהווה Handle לסביבת האפליקציה וליישומים הרלוונטיים שלה. אם נלך בעקבות שרשרת הירושות של המחלקה GameActivity, נראה כי בסוף נגיע למחלקה בשם Activity, המייצגת מסך או ממשק בודד באפליקציה:

```
/* loaded from: classes.dex */
public class SupportActivity extends Activity implements LifecycleOwner, KeyEventDispatcher.Component {
```

למעשה, Activity הוא Context בפני עצמו, מאחר והוא מייצג ממשק / מסך באפליקציה. אפליקציה מורכבת מ-Context-ים רבים אשר חולקים משאבים משותפים (מאחר והם חיים תחת אותה אפליקציה, באותו "עולם") ולכן נוכל להשתמש בהם גם אנחנו.

לאחר שהבנו את הצורה בה האפליקציה משתמשת ב-SharedPreferences, נבחן את הפעולות השונות שהיא מבצעת. אם נסתכל על הפונקציה saveData בעלת החתימה (String str, int i):

```
public static void saveData(String str, int i) {
    saveData(str, i, 3);
}
```

נראה כי הפונקציה מבצעת קריאה לפונקציה בעלת שם זהה המקבלת ארגומנט אחד נוסף, כך שערך ארגומנט זה הוא תמיד 3:

```
public static void saveData(String str, int i, int i2) {
  SharedPreferences[] sharedPreferencesArr = _saves;
  if (sharedPreferencesArr[i2] == null) {
      return;
  }
  SharedPreferences.Editor edit = sharedPreferencesArr[i2].edit();
  edit.putInt(str, i);
  if (str == GOLD || str == STONE) {
      edit.putString(CHECK_VALUE + str, Game.md5(i + Settings.System.getString(_context.getContentResolver(),
                                                                                "android_id")));
  }
  edit.commit();
}
```

הפונקציה בודקת אם האובייקט מסוג SharedPreferences במקום ה-3 מוגדר, ואם לא - היא חוזרת. במידה והאובייקט אכן קיים, הפונקציה כותבת ערך לאותו אובייקט על פי הפרמטרים שהתקבלו. נראה שאף על פי שהפונקציה init מייצרת 4 אובייקטי SharedPreferences, נעשה שימוש רק באחרון ברשימה - "שמירה" מספר 3.

מה שמסביר את שם הקובץ שהוציא אותנו למסע! מגניב!☺

נתעמק בצורה בה הפונקציה מכניסה את הערכים המבוקשים לאובייקט ה-SharedPreferences הרלוונטי:

```
SharedPreferences.Editor edit = sharedPreferencesArr[i2].edit();
edit.putInt(str, i);
if (str == GOLD || str == STONE) {
    edit.putString(CHECK_VALUE + str, Game.md5(i + Settings.System.getString(_context.getContentResolver(), "android_id")));
}
edit.commit();
```

מעניין! נראה שכאשר השדה אליו נרצה לכתוב הוא אחד מהערכים "gold" או "magicStone" (השדות אשר זיהינו שמחזיקים את כמות הזהב והיהלומים של השחקן שלנו) תתבצע כתיבה נוספת לשדה בשם "checkValue" משורשר ל-"gold" או "magicStone", כלומר:

- "checkValuegold" במקרה של "gold"
- "checkValuemagicStone" במקרה של "magicStone"

הערך שיכתב לשדה יהיה 5md של הערך המקורי, משורשר ל-android_id של המכשיר (android_id הינו מזהה ייחודי עבור כל מכשיר אנדרואיד). השם שהמפתח בחר לשדה מרמז שאולי מתבצעת בדיקה מסוימת על הערך המיוחד הזה, כדי לוודא שהערך המקורי שהוכנס לא שונה - מנגנון שמבצע בדיקה על האותנטיות של המידע. לצערי לא מתבצעת בדיקה כלשהי על הערך הזה בעת קריאה, ככל הנראה מדובר במנגנון שלא הוטמע עד הסוף.

אז אחרי שאנחנו בטוחים שהקובץ "save3.xml" מכיל את המידע על השחקן שלנו, וככל הנראה נוכל לשנות אותו בלי בעיה - בואו נעשה זאת! נמשוך את הקובץ באמצעות adb, נשנה את השדות הרלוונטיים ונדחוף את הקובץ הערוך בחזרה לתיקיה של המשחק:

```
from ppadb.client import Client as AdbClient
import xml.etree.ElementTree as ET
from defender import Defender

SAVE_PATH = "/data/data/com.droidhen.defender2/shared_prefs/save3.xml"
LOCAL_FILE_NAME = "save3.xml"


1 usage
def main():
    client = AdbClient()
    device = client.devices()[0]

    device.pull(
        SAVE_PATH,
        LOCAL_FILE_NAME
    )

    tree = ET.parse(LOCAL_FILE_NAME)
    root = tree.getroot()

    for elem in root.iter():
        if elem.get("name") == "gold":
            elem.set( _key: "value",  _value: "9999")
        elif elem.get("name") == "magicStone":
            elem.set( _key: "value",  _value: "9999")

    tree.write(LOCAL_FILE_NAME)

    device.push(LOCAL_FILE_NAME, SAVE_PATH)


if __name__ == "__main__":
    main()
```

נפתח את המשחק מחדש (מאחר ושינינו את התוכן של הקובץ, נרצה שהמשחק יקרא את הקובץ החדש שדחפנו לתיקיה). המשחק התעדכן בהתאם לשינויים שלנו!

חלק ב'

אחרי שהבנו איך נוכל לתת לשחקן שלנו אינסוף זהב ויהלומים, זה השלב להעלות הילוך! באמצעות הכסף שהשגנו לשחקן שלנו נוכל לקנות את כל השיפורים במשחק, בהתאם לשיפורים שנקנה, הנזק שנעשה למפלצות באמצעות כל חץ יעלה, וכמו כן קצב האש שלנו יעלה גם כן. אבל לנו זה לא מספיק!

איך נוכל לגרום לחץ שאנחנו יורים להרוג כל מפלצת במכה? נרצה להבין כיצד מנגנון הירי ממומש, הרי בסוף בוודאות יש פונקציה כלשהי שאחראית על אירוע של "פגיעה" במפלצת. כל מה שעלינו לעשות הוא להבין איך הלוגיקה הזו ממומשת ולשנות אותה לבחירתנו.

נשים לב לתיקיה **maingame** המכילה מספר קבצים עם שמות מעניינים:

קובץ שישר תופס את העין הוא Bow אשר ככל הנראה מחזיק בתוכו את הלוגיקה והפונקציונליות של הקשת במשחק. מאחר והקוד לא עבר אובפוסקציה, נוכל להיעזר בשמות של הפונקציות כדי להתקרב אל מה שאנחנו מחפשים, אבל מה היינו עושים אם לא היו בכלל שמות לפונקציות ולמשתנים? איך נוכל להבין על סמך הקוד מה הרלוונטי עבורנו ובאיזה חלקים כדאי להתמקד?

Objection

הכלי Objection הוא למעשה עטיפה מתוחכמת ונוחה למספר כלים שונים, ביניהם Frida ו-ADB. הוא מאפשר לנו לייצר כמות גדולה של Hooks במהירות, להריץ סקריפטים של Frida בקלות ועוד שלל יכולות מעניינות עליהן ניתן לקרוא בתיעוד של הכלי (הכלי כולל כלים שימושיים גם לעולמת ה-iOS!). פונקציונליות שימושית ש-Objection מציע לנו הוא יצירת Hooks על מחלקה שלמה, כלומר Objection ייצר Hook עבור כל מתודה במחלקה, ויודיע לנו ברגע שאחת מהן נקראת.

נתקין את Objection בקלות על ידי pip:

```
pip3 install objection
```

ואנחנו מוכנים לצאת לדרך!

כדי להזריק את האייג'נט של Objection (בדומה לצורה בה אנחנו מזריקים סקריפט Frida אל תוך תהליך המתקשר עם השרת Frida של המכשיר, אשר מתקשר איתנו) נריץ את הפקודה הבאה:

```
objection -g <package_name> explore
```

במקרה שלנו נריץ:

```
objection -g com.droidhen.defender2.explore
```

ונקבל משהו כזה:

```
user@user-B760M-AORUS-ELITE-AX:~$ objection -g com.droidhen.defender2 explore
Using USB device `Galaxy S10`
Agent injected and responds ok!

     _   _         _   _
 ___| |_|_|___ ___| |_|_|___ ___
| . | . | | -_|  _|  _| | . |   |
|___|___| |___|___|_| |_|___|_|_|
      |___|(object)inject(ion) v1.11.0

     Runtime Mobile Exploration
        by: @leonjza from @sensepost

[tab] for command suggestions
com.droidhen.defender2 on (Samsung: 11) [usb] #
```

ל-Objection מנגנון Auto Bompletion, ונוכל לעיין באפשרויות השונות שהוא מציע בלחיצה על tab.

כדי לעשות Hook על מחלקה שלמה, נכתוב את הפקודה הבא:

```
android hooking watch class "className"
```

במקרה שלנו:

```
android hooking watch class "com.droihen.defender2.game.maingame.bow"
```

ונקבל את ה-output הבא:

```
com.droidhen.defender2 on (Samsung: 11) [usb] # android hooking watch class "com.droidhen.defender2.game.maingame.Bow"
(agent) Hooking com.droidhen.defender2.game.maingame.Bow.draw(javax.microedition.khronos.opengles.GL10)
(agent) Hooking com.droidhen.defender2.game.maingame.Bow.init()
(agent) Hooking com.droidhen.defender2.game.maingame.Bow.shotArrow(float, float)
(agent) Hooking com.droidhen.defender2.game.maingame.Bow.update()
(agent) Registering job 273885. Type: watch-class for: com.droidhen.defender2.game.maingame.Bow
```

קול! עכשיו ברגע שאחת מהפונקציות הרלוונטיות תקרא, Objection יודע לנו על כך.

נתחיל משחק ונשים לב שיש לנו 2 פונקציות שנקראות באופן תמידי, draw ו-update. המטרה של draw היא ככל הנראה לצייר אובייקטים על המסך בהתאם למיקום שלהם. המתודה update אחראית לעדכן את ה-state של האובייקט הרלוונטי, והיא ככל הנראה נקראת על ידי פונקציית update ראשית שמטרתה לעדכן את ה-state של המשחק בכל רגע נתון:

```
this._camera.update();
if (!this._isGameOver) {
    Help.update();
    if (!Help.isShow()) {
        this._monster.update();
        this._mission.update();
        this._arrow.update();
        this._bow.update();
        this._magic.update();
        this._criEffect.update();
        this._panel.update();
        this._wallDefender.updata();
    }
}
```

```
@Override // com.droidhen.defender2.sprite.Scene
public void update() {
```

ברגע שנכנסה לירות חץ בודד, נשים לב שהפונקציה shotArrow נקראת:

```
(agent) [273885] Called com.droidhen.defender2.game.maingame.Bow.update()
(agent) [273885] Called com.droidhen.defender2.game.maingame.Bow.shotArrow(float, float)
(agent) [273885] Called com.droidhen.defender2.game.maingame.Bow.draw(javax.microedition.khronos.opengles.GL10)
(agent) [273885] Called com.droidhen.defender2.game.maingame.Bow.update()
(agent) [273885] Called com.droidhen.defender2.game.maingame.Bow.shotArrow(float, float)
(agent) [273885] Called com.droidhen.defender2.game.maingame.Bow.draw(javax.microedition.khronos.opengles.GL10)
```

הפונקציה מחשבת את הכיוון אליו החץ אמור להיות משוגר, ומדליקה את הדגל toShot וככל הנראה "מכינה את הקרקע" לפונקציה אחרת שתקרא את המידע ותדאג להביא לידי ביטוי את יריית החץ באופן גרפי:

```
public void shotArrow(float f, float f2) {
    double atan = ((float) Math.atan((f2 - this._y) / (f - this._x))) * 180.0f;
    Double.isNaN(atan);
    this._targetAngle = (float) (atan / 3.141592653589793d);
    this._toShot = true;
    this._shotX = f;
    this._shotY = f2;
}
```

הפונקציה עצמה נקראת מתוך הפונקצית update של המחלקה:

```
public void update() {
    if (this._game.isRep() && Report.getReadRep().arrowRep.checkAction(Game.getGameTime())) {
        shotArrow(Report.getReadRep().arrowRep.getX(), Report.getReadRep().arrowRep.getY());
    }
```

כלומר בכל רגע נתון המשחק יבדוק האם הדגל toShot דולק ובמידה וכן הוא יריץ את לוגיקת יריית החץ:

```
public void update() {
    if (this._game.isRep() && Report.getReadRep().arrowRep.checkAction(Game.getGameTime())) {
        shotArrow(Report.getReadRep().arrowRep.getX(), Report.getReadRep().arrowRep.getY());
    }
    float f = this._coolDownTime;
    if (f > 0.0f) {
        this._coolDownTime = f - ((float) Game.getLastSpanTime());
    }
    if (this._toShot) {
```

בתוך התנאי שאליו נכנס במידה והדגל toShot דולק ישנם מספר תנאים נוספים שאין לנו צורך להבין כדי להמשיך במחקר שלנו. נראה שבכל תנאי שאליו נגיע, תקרא מתודה מעניינת בשם shoot על אובייקט מסוג Arrow:

```
public class Bow {
    private int CD;
    private float _angle;
    private Arrow _arrow;
```

```
int i = this._arrow.arrowNum;
if (i == 1) {
    this._arrow.shoot(this._angle);
} else if (i == 2) {
    this._arrow.shoot(this._angle + 1.5f);
    this._arrow.shoot(this._angle - 1.5f);
} else if (i == 3) {
    this._arrow.shoot(this._angle + 3.0f);
    this._arrow.shoot(this._angle);
    this._arrow.shoot(this._angle - 3.0f);
} else if (i == 4) {
    this._arrow.shoot(this._angle + 1.5f);
    this._arrow.shoot(this._angle - 1.5f);
    this._arrow.shoot(this._angle + 4.5f);
    this._arrow.shoot(this._angle - 4.5f);
} else if (i == 5) {
    this._arrow.shoot(this._angle + 3.0f);
    this._arrow.shoot(this._angle - 3.0f);
    this._arrow.shoot(this._angle);
    this._arrow.shoot(this._angle + 6.0f);
    this._arrow.shoot(this._angle - 6.0f);
}
```

המתודה shoot של המחלקה Arrow מבצעת קריאה למתודה אחרת במחלקה בשם add:

```
public void shoot(float f) {
    add(f);
}

private void add(float f) {
    if (this._recycleArrow.isEmpty()) {
        this._usingArrow.add(new BasalArrow(f, this._power, ItemParam.getLevel(24)));
    } else {
        this._usingArrow.add(this._recycleArrow.get(0).init(f, this._power, ItemParam.getLevel(24)));
        this._recycleArrow.remove(0);
    }
}
```

המתודה מייצרת אובייקט חדש בהתאם לתנאי ומוסיפה אותו לרשימה, בייצור של האובייקטים ישנו ארגומנט עם שם מאד מעניין - power, מעניין מה התפקיד שלו...

נכתוב סקריפט Frida בו נבצע hook לפונקציה add ונדפיס את הערך של power בכל קריאה:

```
Java.perform(() => {
    let Arrow = Java.use("com.droidhen.defender2.game.maingame.Arrow");
    Arrow["add"].implementation = function (f) {
            console.log(this._power.value)
            this["add"](f);
    };
})
```

(מאחר ולא מימשנו את הלוגיקה של הפונקציה בתוך ה-Hook ורק הוספנו שורת הדפסה, נקפיד לקרוא לפונקציה המקורית בסוף ה-Hook כדי להימנע מלפגוע בלוגיקת המשחק).

נזריק את הסקריפט באמצעות Frida, נתחיל משחק חדש ונראה מספר חצים:

```
user@user-B760M-AORUS-ELITE-AX:~$ frida -U -f com.droidhen.defender2 -l DEFENDER.js
```

הערך 24 מודפס למסך מספר פעמים:

```
[Galaxy S10::com.droidhen.defender2 ]-> 24
24
24
24
24
24
```

בואו ננסה לשנות את הערך של power למספר גבוהה, ולראות מה קורה (פעמים רבות כאשר נבצע מחקר סטטי ונרצה להבין איפה אנחנו עומדים, האם הגענו לאובייקט מעניין או לפונקציה שחיפשנו, בדיקות דינמיות מסוג זה ייתנו לנו אינפורמציה רבה שלא נוכל להשיג במחקר סטטי בזמן סביר):

```
Java.perform(() => {
    let Arrow = Java.use("com.droidhen.defender2.game.maingame.Arrow");
    Arrow["add"].implementation = function (f) {
            this._power.value = 1337
            this["add"](f);
    };
})
```

נטען את הסקריפט מחדש ונגלה כי המפלצות מתות מפגיעה של חץ אחד! כדי לוודא שאכן הגענו למקום הנכון, נשנה את הערך של power ל-0 ונשים לב שהחצים לא עושים נזק למפלצות. אם נגדיר מספר שלילי, נוכל לראות איך בר החיים של המפלצות גדל!

כדי להכיר טריק Frida נוסף, ננסה להשיג את אותה מטרה בדרך אחרת, מה אם נוכל לעשות Hook על הקונסטרקור של המחלקה Arrow, כך שכל Instance של המחלקה אוטומטית יאתחל את power עם 0?

```
public Arrow(MainGame mainGame, GLTextures gLTextures) {
    this._clearFlag = false;
    this._clearFlag = false;
    this._game = mainGame;
}

public void reset() {
    arrowSkillInit();
    this._clearFlag = true;
    this._power = SkillData.getValue(7, BowData.getAbility(ItemParam.getLevel(24), 0));
    if (this._game.isRep()) {
        return;
    }
    this._power = (int) (this._power + ((r0 * Param.extraDmg) / 100.0f));
}
```

במידה ונרצה לממש את הפתרון בדרך זו, נשים לב כי המתודה reset מאפסת את הערך של power בכל משחק, ולכן נצטרך לבצע עלייה Hook בהתאם.

אף על פי שביצוע Hook לקונסטרקטור של המחלקה מרגיש כמו פתרון מצוין, נבחר לבצע אותו בדרך טיפה שונה - נשתמש ביכולת של Frida לחפש אובייקטים בזיכרון כדי למצוא אובייקט Arrow קיים ולשנות את ערך ה-power שלו.

כדי לעשות זאת, נשתמש ב-Choose על פי הסקריפט הבא:

```
Java.perform(() => {
    Java.choose("com.droidhen.defender2.game.maingame.Arrow", {
        onMatch:function(instance) {
            instance._power.value = 1337
            console.log("edited")
        },
        onComplete:function() {
            console.log("Done")
        }
    })
})
```

הלוגיקה שנגדיר בתוך ה-onmatch event תתרחש עבור כל אובייקט מסוג Arrow שנמצא. הלוגיקה שנגדיר בתוך ה-oncomplete event תתרחש בסוף החיפוש. נשים לב כי הסקריפט ימלא את תפקידו רק כאשר קיים instance מסוג Arrow בזיכרון, אמליץ להריץ את הסקריפט אחרי ריצה של משחק אחד לפחות.

סופר מגניב!

כעת, כשהבנו את העניין, בואו נעבוד באותה דרך כדי לכתוב GODMODE, כך שמפלצות לא יוכלו להוריד לנו חיים כאשר יגיעו לחומה שלנו. קובץ מעניין שישר קופץ לעין הוא Wall בו המימוש למחלקה Wall המכילה מתודות רבות ומעניינות.

גם כאן נוכל להשתמש ב-Objection כדי לראות איזה מתודה נקראת כאשר מפלצות פוגעות בחומה שלנו, אך מאחר ומספר גדול של מתודות נקרא בכל רגע, נתאמץ למצוא במחקר סטטי מתודה שיכולה לעניין אותנו, ונבצע את הבדיקה עליה.

ככל הנראה נחפש מתודה שהמטרה שלה לממש לוגיקה של "נפגעתי", והיא ככל הנראה תקבל משתנה מספרי שמייצג את כמות הנזק שנעשה לה על ידי מפלצת (מאחר ומשחק מפלצות שונות בעלות עוצמה שונה). נראה שהמתודה beHitAct עונה בדיוק על המאפיינים האלה:

```
private void beHitAct(int i, boolean z) {
    this._beHit = true;
    this._hitPoint += i;
    this._isHeavyAttack = z;
    this._colorScale = 0.85f;
}
```

נשתמש שוב ב-Objection, אך הפעם נבצע Hook למתודה בודדת באמצעות הפקודה:

```
android hooking watch class_method "className.method"
```

כמו כן, נוסיף את הדגל dump-args-- כדי ש-Objection יציג בפנינו את הפרמטרים שהמתודה קיבלה:

```
# android hooking watch class_method "com.droidhen.defender2.game.maingame.Wall.beHitAct" --dump-args
(agent) Attempting to watch class com.droidhen.defender2.game.maingame.Wall and method beHitAct.
(agent) Hooking com.droidhen.defender2.game.maingame.Wall.beHitAct(int, boolean)
(agent) Registering job 071150. Type: watch-method for: com.droidhen.defender2.game.maingame.Wall.beHitAct
```

נתחיל משחק ונחכה שמפלצת תגיע אל החומה שלנו. נשים לי כי ברגע שדבר זה יקרה נראה כי הפונקציה נקראה ושערך הפרמטר i הוא 3.

אם נבחן את מד החיים שלנו במהלך המשחק, נשים לב שפגישה של מפלצת תוריד לנו 3 נקודות חיים. כלומר הפרמטר i קובע כמה חיים ירדו לנו ולכן כל מה שעלינו לעשות הוא לדאוג שהמתודה תמיד תקרא כך שהערך של i הוא 0.

נכתוב סקריפט Frida פשוט שמבצע Hook לפונקציה, וקורא ללוגיקת הפונקציה המקורית עם הערך 0 במקום הערך של i:

```
Java.perform(() => {
    let Wall = Java.use("com.droidhen.defender2.game.maingame.Wall");
    Wall["beHitAct"].implementation = function (i, z) {
        this["beHitAct"](0, z)
    };
})
```

מדהים! נראה שהמפלצות כבר לא מורידות לנו חיים!

סיכום

במאמר זה חקרנו משחק אנדרואיד פשוט וכתבנו צ'יטים שיאפשרו לנו להביס את המשחק בקלות בכך ששינינו את קבצי השמירה של המשחק כדי לתת לעצמנו אינסוף זהב ויהלומים והתערבנו בלוגיקת המשחק כדי לשנות אותה לטובתינו. את כל זאת עשינו לצד היכרות עם כלים חדשים ולמידה מעמיקה יותר של כלים שכבר השתמשנו בהם במאמר הקודם.

מקווה שנהנתם!

על המחבר

עידן שכטר, בן 26, אוהב בעלי חיים ואת הים, חוקר בקבוצת NSO.