Technologic Papers

משטחי תקיפה לאפליקציות Android - חלק II

מאת 0x3d5157636b525761

הקדמה

בפרק הקודם דיברנו על מודל האבטחה של אנדרואיד (ממש בשתי מילים), והצגנו כיצד אפליקציות מסויימות עלולות לסמוך על SMS בלי לוודא האם התוכן שלו אותנטי. במאמר זה נמשיך ונציג כיצד RootBrowser - אפליקציה rooted - סומכת על HTTP כדי לבצע פעולות עדכון מסוכנות.

בפרק הקודם

בפרק הקודם דיברנו על מודל האבטחה של אנדרואיד (ממש בשתי מילים), והצגנו כיצד אפליקציות מסויימות עלולות לסמוך על SMS בלי לוודא האם התוכן שלו אותנטי.

במאמר זה נמשיך ונציג כיצד RootBrowser - אפליקציה rooted - סומכת על HTTP כדי לבצע פעולות עדכון מסוכנות.

השתלשלות האירועים

- 11.06.2016 - גילוי הנקודות (שתוצגנה בהמשך) ב-RootBrowser.
- 12.06.2016 - פנייה אל מפתחי האפליקציה. התגובה הייתה שכרגע עובדים על גרסא חדשה שתפתור את הבעיות הנוכחיות, אך אין צפי לזמן הוצאת הגרסא החדשה.
- 02.07.2016 - פנייה נוספת אל מפתחי האפליקציה במטרה לקבל מידע על התקדמות פתרון הבעיה, אך ללא תגובה.
- 08.07.2016 - פנייה שלישית אל מפתחי האפליקציה, גם ללא תגובה.
- 09.07.2016 - פרסום (Public disclosure).

עדכונים באנדרואיד

אנדרואיד מספקת ממשק נוח מאד לעדכן אפליקציות. לכל אפליקציה יש גרסא שמופיעה בקובץ ה-AndroidManifest.xml שלה, וניתן לעדכן גרסאות על ידי הפצה מחדש של האפליקציה מעל ה- Google Play store. עקרונית, כל עוד ה-certificate שחתום על האפליקציה זהה, ניתן לבצע עדכון של האפליקציה.

החסרון במנגנון זה הוא שלעיתים אפליקציות לא מופצות דרך ה-Play store, אלא ב-store-ים אלטרנטיביים או אפילו כקבצי APK. במקרה זה, מפתחי האפליקציה צריכים לחשוב בעצמם על מנגנון עדכון, ולעיתים קרובות מנגנון זה יכול להיות בעייתי מאד. בסוף המאמר נציג מספר כללי ברזל לקוד לעדכון אפליקציה ידני שכזה.

אפליקציות rooted

בפעם הקודמת הזכרנו את מנגנון ההרשאות של אנדרואיד, שבחלקו הגדול מתבסס על יצירת user-ים חדשים עבור כל אפליקציה חדשה. במערכת אנדרואיד (ובכלל במערכות דמויות לינוקס), המשתמש החזק ביותר הוא root (עם ID=0). משתמש זה יכול לבצע כמעט כל דבר:

- דיבוג באמצעות ptrace.
- ביצוע mount.
- ביצוע chown ו-chmod.
- התעלמות מ-DAC-ים על קבצים.

עם זאת, לשם השלמות נציין כי בגרסאות אנדרואיד מתקדמות ישנו מנגנון נוסף בשם SEAndroid (שהוא למעשה התאמה של SE-linux לאנדרואיד) שאליו כפופים כל המשתמשים - גם root. עם זאת, בחלק גדול של ה-Vendor-ים ניתן לכבות אותו (על ידי setenforce) או לחילופין לטעון kernel module שמנוון את המנגנון.

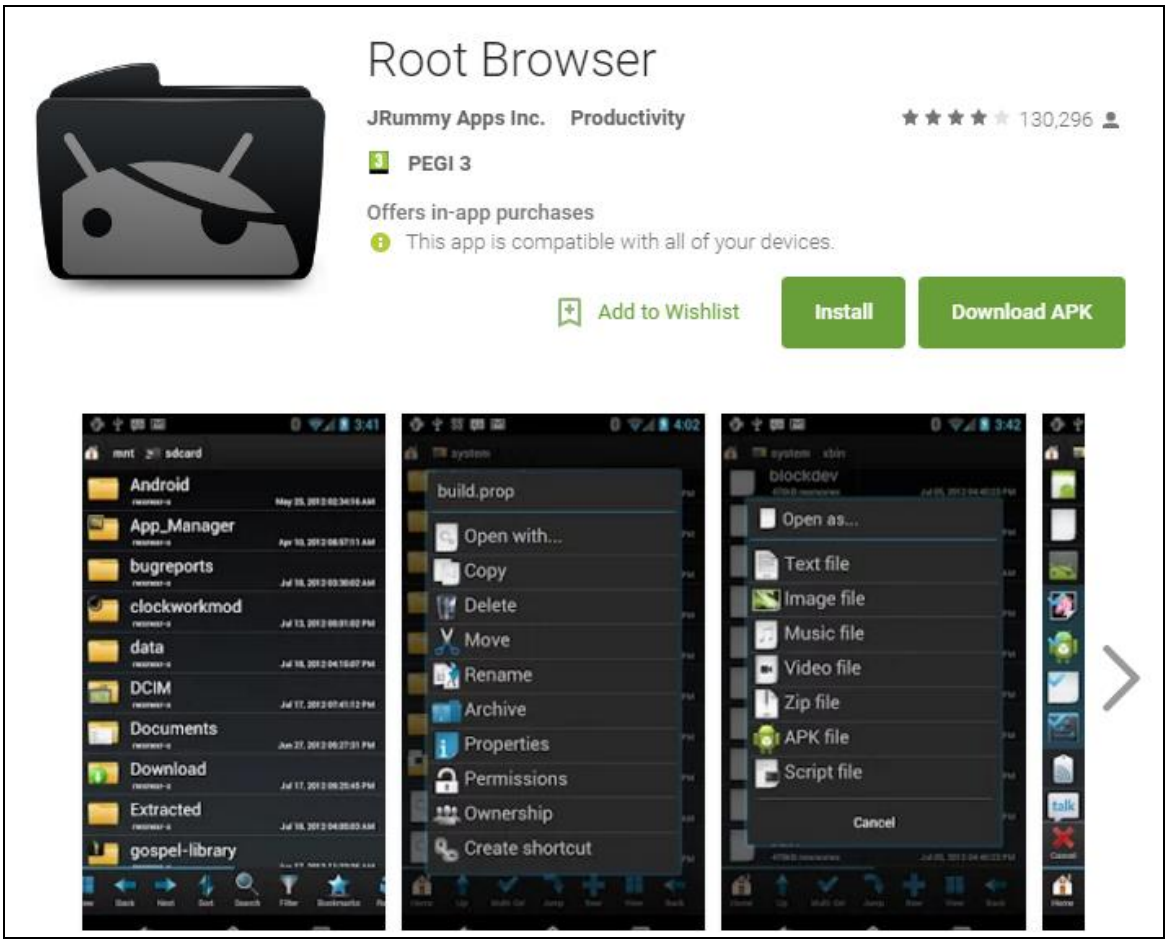
כאשר אומרים שטלפון הוא "rooted" הכוונה היא בדרך כלל לכך שניתן להריץ פקודות על ידי פנייה לבינארי בשם su. בעבר, su זה היה קובץ בינארי רגיל עם דגל setuid דלוק ו-root owner, כך שהרצה שלו תמיד רצה כ-root. מערכות אנדרואיד חדשות לא מכבדות כבר את setuid, ולכן su הוא בדרך כלל בינארי רגיל שמבצע תקשורת מול רכיב בשם sudaemon, שעשה forking מתוך init בשלב מאד מוקדם בעליית המערכת.

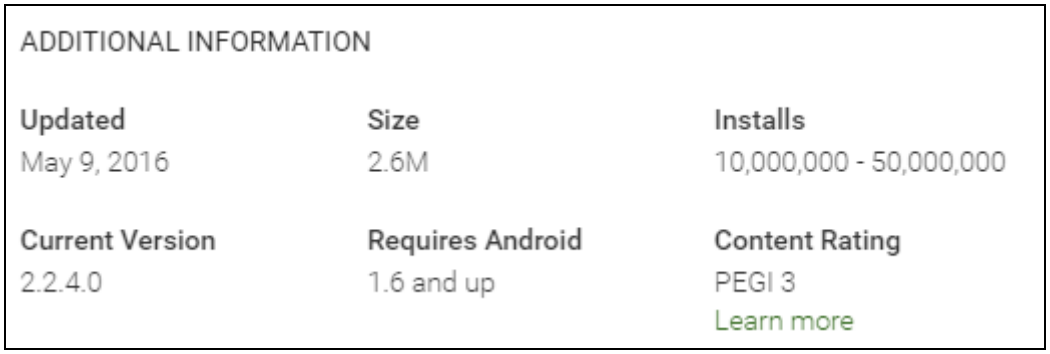
המסקנה החשובה היא ש-rooting נותן כח גדול בידי המשתמש - אפליקציות רגילות יכולות לקרוא ל-API שמאפשר ביצוע פעולות בהרשאות גבוהות. לצערנו, with great power comes great responsibility ולכן מפתחי אפליקציות rooted מחוייבים (במובן מסויים) לסטנדרטים גבוהים יותר (לפי דעתי האישית).

מבט שטחי על RootBrowser

אז, RootBrowser היא אפליקציה rooted שמכוונת לביצוע פעולות עם הרשאות גבוהות על מערכת הקבצים. למשל, אפשר לבצע remounting ל-system partition כך שאפשר יהיה לכתוב עליה (כך ניתן לשלוט באפליקציות system, למשל), אפשר לבצע chown ו-chmod על קבצים כרצוננו וכדומה.

האפליקציה עצמה די פופולרית:





מתחת לפני השטח, RootBrowser משתמשת ב-"ארגז כלים" מיוחד שמכיל פקודות shell נפוצות. מכיוון שלמערכת אנדרואיד יש shell מאד בסיסי, RootBrowser מעוניין בארגז כלים שכזה לביצוע פעולות מחוכמות יותר - ארגז כלים זה נקרא גם busybox.

למי שלא מכיר - busybox הוא בינארי יחיד שמכיל בתוכו מימושים שונים לכלי shell (כגון zip או אפילו ls). כל אחד יכול לקמפל לעצמו busybox כרצונו. באופן מסורתי שמים בדרך כלל link-ים של הפקודות לתוך

busybox. כך למשל, zip מצביע אל busybox, ו-busybox יודע לבצע את הפעולה האחרונה. ניתן להוריד ולשחק עם busybox באתר https://busybox.net.

באופן מסורתי, כאשר אפליקציות מעוניינות לכלול קבצים בינאריים "בתוך הבטן", הן מחזיקות את הקבצים הללו בתור Assets. תחת אנדרואיד, Assets ו-Resources דומים מאד, מלבד כך שניתן לפנות אל Resources על ידי ID מיוחד שמגונרץ לתוך מחלקה בשם R, בעוד שלא ניתן לעשות דבר דומה עם Assets. ה-Assets עצמם נשמרים תחת תיקייה בשם Assets, שנמצאת תחת ה-sandbox של האפליקציה, ולכן "מוגנים" מהעולם החיצון.

לצערי, מפתחי RootBrowser החליטו שהם מעוניינים לכתוב גרסת עדכון מרחוק ל-BusyBox שלהם (והם אכן קימפלו אחד custom של עצמם) - ועדכון זה נעשה מעל HTTP. להלן קטע הקוד הרלוונטי:

```
private void a(String paramString)
{
  File localFile1 = new File(b, paramString);
  File localFile2 = new File(this.f, paramString);
  if (localFile1.exists())
  {
    new i(this, localFile1, localFile2, paramString).start();
    return;
  }
  com.jrummy.file.manager.h.b localb = new com.jrummy.file.manager.h.b("
  http://jrummy16.com/jrummy/rootbrowser/assets/" + paramString, new File(this.c.getFilesDir(), paramString).
  getAbsolutePath());
  localb.a(this.i);
  Message localMessage = this.i.obtainMessage(0);
  Bundle localBundle = new Bundle();
  localBundle.putString("msg", paramString);
  localMessage.setData(localBundle);
  localMessage.setTarget(this.i);
  localMessage.sendToTarget();
  new Thread(localb).start();
}
```

הקוד מקבל שם של asset, בודק אם הוא כבר ירד, ואם לא אז מוריד אותו משרת http רגיל (jrummy16.com). כמובן שביצוע HTTP MitM רגיל (עם mitmproxy למשל) נותן לתוקף שליטה מלאה על ה-busybox שיורד.

מכיוון שהאפליקציה כבר rooted, ניתן למעשה להגיד שהעבודה כמעט הסתיימה - HTTP MitM נותן לתוקף RCE מספיק privileged כדי לגרום לנזק עצום למערכת (ולמעשה רץ תחת root).

מה ניתן היה לעשות?

באופן כללי, העצה הטובה ביותר למפתחי אפליקציות היא לא לממש מנגנונים מורכבים (כגון עדכון או הצפנה) בעצמכם. אנדרואיד יודעת לעדכן באופן מאובטח למדי אפליקציות. אם בכל זאת הייתם מעונינים לממש מנגנון עדכון באפליקציה, הנה מספר טיפים:

1. תמיד יש לוודא שהעדכון לא מתבצע ב-plaintext. בצעו את העדכון מעל תווך מוצפן ו-authenticated שכבר הוכח כבטוח (יחסית) כגון TLS (בגרסא החדשה ביותר כמובן).
2. וודאו שהשרת שממנו מעודכנת הגרסא הוא הנכון. אם מדובר על SSL אז בצעו pinning, אם מדובר על מנגנון אחר אז הכניסו וידוא בדמות חתימה דיגיטלית שייצרתם מראש. ישנם API-ים מובנים לכך ב-Java - נצלו אותם!
3. לא להתקמצן על אורך חתימה דיגיטלית!
4. השתדלו שקוד העדכון שלכם ימומש בשפה high level-ית (ולא, למשל, מעל JNI). הסיכוי ל- Memory Corruption נמוך משמעותית במקרה זה.
5. נסו להמנע משימוש בקבצי zip לעדכון, שכן אלו חשופים לעיתים ל-path traversal.
6. בצעו וידוא על העדכון לפני ההרצה שלו. זה צריך להיות ברור מאליו, אך כבר ראיתי מקרים רבים שבהם לא עשו כך.

מסקנות

1. בתחילת מאמר זה הסברנו קצת על rooting, איך הוא עובד באנדרואיד ומדוע אפליקציות rooted צריכות להיות מוגנות אפילו יותר מאפליקציות רגילות.
2. הראינו דוגמא ל-RootBrowser - אפליקציה rooted שהחליטה לממש מנגנון עדכון משל עצמה - ועשתה זאת תוך חשיפה של משתמשי קצה רבים להשתלטות מרחוק.
3. בסוף המאמר הצגנו מספר קווים מנחים למימוש מנגנון עדכון. הזכרנו במיוחד שכל מנגנון עדכון צריך להיות חתום קריפטוגרפית על ידי ישות אמינה.

אף על פי שסדרת המאמרים תופץ גם בעברית, אני מזמין את הקורא השקדן לקרוא את הפוסט המקורי בבלוג שלי, בכתובת: http://securitygodmode.blogspot.com.