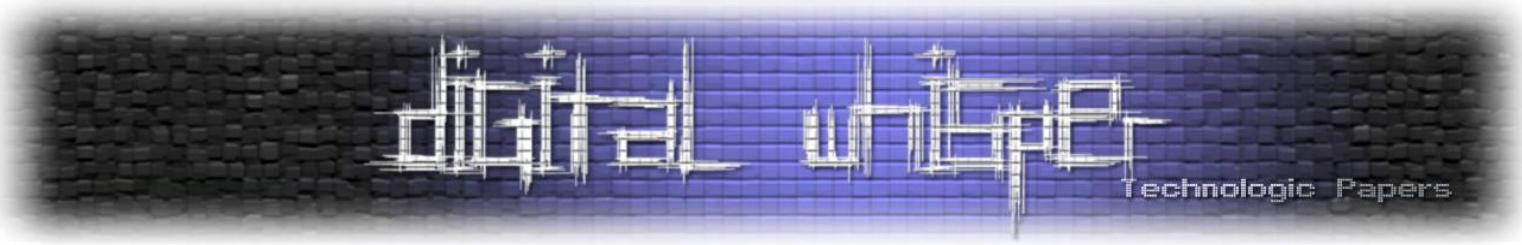


Android Firmware Modding

מאת אשי ורד (A.I.V Dev)

הקדמה

כבר מספר שנים שאני מתעסק מתוך תחביב בעריכת קושחות של טלפונים, במאמר זה נתמקד באנדרואיד (בכל זאת, מישהו היום מתעניין ב-Symbian או ב-Windows Mobile?). איך עושים את זה? משתנה בין מכשירים, יצרנים, דגמים, גרסאות אנדרואיד ודגם ה-Chipset. אחרי שסיימתי פרויקט על מכשיר של סמסונג, Galaxy A06, כתבתי פוסט קצר ולא מפורט בכלל בבלוג שלי, עם סקירה שטחית בלבד של הצעדים שעשיתי. הפוסט ההוא נועד למי שכבר בתוך התחום ומבין מה כתבתי שם. מאז למדתי עוד כמה שיטות לעשות דברים - וגם נדבר עליהם במאמר כאן, וחוץ מזה, המאמר הזה אמור להיות שער לכל מי שירצה לצלול לנבכי העולם המרתק הזה.

במאמר אני נותן את העקרונות הכלליים לפרויקט מהסוג הזה, אבל לא ארד לפרטים ספציפיים שמשתנים בין יצרנים, דגמי Chipset שונים, או גרסאות אנדרואיד שונות, שדברים יכולים להיות שונים בצורה ניכרת. לכן, אני ממליץ לחפש ברשת כיצד לבצע Root למכשיר שלכם, כדי להבין מה לעשות בדיוק.

למען האמת, בהתחלה תכננתי לכתוב ולהדגים על מכשיר ספציפי, אבל הבנתי שזה לא מאוד יעיל, בגלל ההבדלים הגדולים בין המכשירים השונים, ושעדיף להסביר את העקרונות.

וכאן, אני כותב כמה אזהרות שנכונות לאורך כל הדרך ולכל מכשיר, ואני לא הולך לחזור עליהן בהמשך המאמר:

א. אנחנו מתעסקים ברכיבים קריטיים של מערכת ההפעלה. בסבירות גבוהה מאוד נגיע (זה קורה גם לי אחרי שנים בתחום) למצבים שהמכשיר לא נדלק. כמעט תמיד אפשר לצרוב קושחה מקורית ולפתור את הבעיה, אך יש מקרים נדירים שבהם הבעיה יותר מורכבת. במקרים בודדים ממש גם הרסתי מכשירים לחלוטין. תהיו זהירים, ותעשו הכול על אחריותכם בלבד.

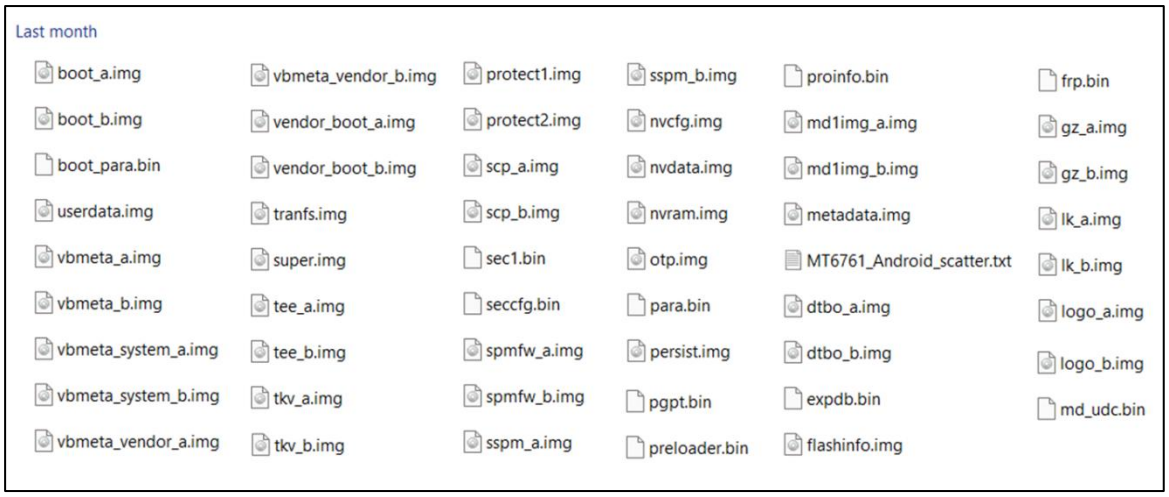
ב. שימו לב שכל מידע שיהיה על המכשיר ימחק, ושהמכשיר יהיה פחות מאובטח, סביר להניח שאפליקציות בנקאיות וכדומה לא ירוצו עליו בלי משחקי חתול ועכבר של זיופי הגנות והגנות נגד זיופי הגנות וכו'. בקיצור, לא מומלץ כמכשיר ראשי.

ג. כמובן, אף אחד לא ייתן לכם אחריות על המכשיר אחרי התעסקות עם הקושחה. זה מבטל את האחריות שיש לכם על המכשיר. ליצרנים מסוימים (כמו סמסונג) יש הגנה שמזהה שהמכשיר עבר עריכה אפילו אם נצרוב מחדש את הקושחה המקורית, נאפס את המכשיר וננעל בוטלואדר. שימו לב, יש כאן דברים בלתי הפיכים.

החלטתם להמשיך? מעולה. Let's do it!

המחיצות השונות בקושחה

אם נסתכל על הקושחה שלנו (אותה או נמצא ברשת, או נגבה [להלן: נשאב] מהמכשיר. אין אתר אחד מרכזי להורדת קושחות, ואין כלי אחד לשאיבה של הקושחה. משתנה מאוד בין יצרני מכשירים, יצרני ה-SoC ועוד.), אנחנו נראה בה הרבה קבצי img[1]. לדוגמא, צילום מסך של קושחה של מכשיר סיני בשם Tecno Pop 6 Pro BE8:



יש פה גם לא מעט קבצי bin, זה לא הבדל מהותי, סתם תלוי איך החלטנו לקרוא לקובץ. כשעורכים קושחה, לרוב עובדים על 3 מחיצות בלבד: Boot, Recovery ו-System.

בדוגמא שצירפתי מחיצות Recovery ו-System לא מופיעות, וכן חלק מהמחיצות מופיעות פעמיים - למשל boot_a ו-boot_b. אסביר על נקודות אלו בהמשך, כרגע נסביר מה תפקיד כל מחיצה מהמחיצות האלו. (כמובן, יש לנו גם את מחיצות ה-VBMETA, עליהם נדבר בשלב הבא) אני לא אאריך בהסבר הטכני על תפקידה של כל מחיצה, כי אנחנו עסוקים כאן יותר ב-Reversing מאשר בלימוד מעמיק של ה"תאוריה" מאחורי אנדרואיד. למי שרוצה ללמוד לעומק את אנדרואיד, אני ממליץ על המאמר המעולה של שיא תדמור (בגיליון 117 של Digital Whisper): Diving into Android.

לענייננו, רק חשוב להבין שמחיצת boot כשמה כן היא - המחיצה שאחראית על העלייה של המכשיר. אם מחיצה זו נפגמה, המכשיר לא יידלק. היא מכילה את הקרנל וה-Ramdisk. בעריכה דרך המכשיר לא ניתן להתעסק איתה, בעריכה דרך המחשב יש כל מיני כלים, אני ממליץ על Android Image Kitchen או MagiskBoot.

[1] שימו לב שקושחות של מכשירי סמסונג או של מכשירים מבוססי SoC מתוצרת Unisoc (מה שמכונה SPD, קיצור של שמה הקודם של החברה) ארוזות בצורה קצת אחרת, אך אם "נפתח" את האריזה נקבל את אותו המבנה.

מחיצת Recovery היא מחיצה למקרה חירום; מכילה מערכת לינוקס מינימלית עם UI בסיסי ומאפשרת לבצע פעולות מסוימות, כגון איפוס המכשיר, התקנת עדכון ע"י קובץ דרך adb או כרטיס זיכרון (כמובן שקובץ העדכון חייב להיות חתום על ידי היצרן!), יצירת Log-ים למערכת וכדומה. אפשר למצוא לא מעט Recovery מקוסטמים, כאשר TWRP הוא הנפוץ ביותר, שמרחיבים מאוד את האופציות. הבסיס הוא כמובן, התקנת עדכונים לא חתומים, אך ממש לא רק. בעריכה דרך המכשיר לא ניתן להתעסק איתה, בעריכה דרך המחשב יש כל מיני כלים, אני ממליץ על Android Image Kitchen או MagiskBoot.

על 2 מחיצות אלו אני לא מפרט כי לא ניגע בהן כלל, מעבר לתהליך הרוט, בו ניגע בהמשך. למקרה שתרצו לערוך אותן (למשל לערוך סקריפטי rc[2] שחלקם נמצאים שם, או את ה-default.prop - הדומה מאוד ל-build.prop שב-System), תדעו שהכלים לפרק ולארוז אותן, כמו גם המבנה שלהם, די זהים.

מחיצת System היא מערכת ההפעלה עצמה. בין השאר - אך בפירוש לא רק - היא כוללת את החלקים שהמשתמש פוגש: האפליקציות, ה-UI וכדומה. מטבע הדברים, זו המחיצה בה אנחנו נתעסק ברוב הזמן.

אז למה מחיצת System לא מופיעה?

בגרסאות אנדרואיד מודרניות (החל מ-10) מחיצת System, יחד עם מחיצות vendor ו-product, שזהות לה במבנה ומאוד דומות בתפקיד (בגדול, System כולל רכיבי אנדרואיד מ-AOSP, Vendor כולל רכיבים של יצרן ה-SoC ו-Product של יצרן המכשיר) כלולות יחד בתוך מחיצת super. בעריכה דרך המכשיר זה לא מאוד משנה. בעריכה דרך המחשב... אנחנו נצטרך לפרק ולארוז את מחיצת ה-super, באמצעות כלים כמו lpunpack ו-lpmake הרשמיים של גוגל, או סקריפטים ותוכנות המבוססים עליהם.

כדי לערוך מחיצות אלו (להלן - וזה לא שם מקובל או נפוץ, אלא רק קיצור שלי לשם הנוחות - מחיצות המערכת) אני הכי אוהב את הדרך הזו - שדורשת לינוקס, אך כמובן אפשרי גם באמצעות WSL.

למה מחיצת Recovery לא מופיעה? כי בהרבה מקרים היא כבר כלולה במחיצת ה-boot.

לגבי A\B - רוב המכשירים היום תומכים בשני "סלוטים" לקושחה - סלוט A וסלוט B. ככה שכשמורידים עדכון מערכת, הוא נצרב לסלוט שלא בשימוש, ואז כשנפעיל מחדש את המכשיר, בעצם נעבור לסלוט המעודכן. לא נעמיק כאן בנושא זה. לפני שמתחילים לעבוד על הקושחה, שווה לבדוק על איזה סלוט המכשיר נמצא כדי שנדע על מה לעבוד. נשתמש בפקודה הזו במצב fastboot (קצת יותר הסבר על המצב הזה בהמשך המאמר):

```
fastboot getvar all
```

ונחפש את השורה:

```
current-slot
```

[2] הסבר בהמשך...

וכך נדע מה הסלוט הפעיל. מחיצת super תמיד יש רק אחת, אבל בתוכה מחיצות ה-System כפולות במקרה של מכשיר שתומך ב-A/B.

כמו שוודאי ראיתם, די קיצרתי בפירוט הכלים לעריכת המחיצות השונות במחשב. זה מכיוון שאני משתמש בדרך כלל בדרך נוחה יותר - עריכה דרך המכשיר עצמו, וזו הדרך שנדבר עליה לאורך המאמר. היתרון הבולט שלה הוא, שחלק מהשינויים אפשר לראות ממש בלייב, ואפילו אלו שלא, בסך הכול צריך הפעלה מחדש, ולא צריבה מחדש של הקושחה.

ADB

ADB (Android Debug Bridge) הוא כלי רב עוצמה לגישה למכשיר. יש פקודות ADB, כגון adb pull ו-adb push (שמשמשות להעברת קבצים בין המכשיר למחשב), אך השימוש הנפוץ ביותר הוא גישה ל-Shell הפנימי של המכשיר.

כאבטחה בסיסית, כמובן שאין גישת ADB למכשיר סתם ככה. צריך להפעיל את הגישה בתפריט נסתר באנדרואיד - תפריט אפשרויות למפתחים. כדי להפעיל את התפריט הזה, נלחץ 7 פעמים על מספר ה-Build בהגדרות > אודות המכשיר, עד שנקבל הודעה "אין צורך, אתה כבר מפתח". נחזור אחורה ונכנס בתפריט ההגדרות לכרטיסיה "מערכת". נראה שנוספה לנו שם אופציה - תפריט מפתחים.[3]

נכנס לתפריט ונפעיל ניפוי באגים באמצעות USB. (אפשר גם ניפוי באגים אלחוטי, דרך WIFI, אבל זה קצת יותר מורכב).

עכשיו נחבר את המכשיר למחשב ונוריד את תוכנת ADB למחשב. בשלב הזה נצטרך גם דרייברים (אלא אם אנחנו בלינוקס.) חפשו את הדרייבר המתאים למכשיר שלכם.

נפתח את ה-CMD (או הטרמינל. יאללה, לינוקס וזה...) ונריץ adb devices. במצב תקין, יופיע לנו במחשב המכשיר שלנו. בשלב הזה, יקפוץ לנו פופ-אפ במכשיר, לאשר את החיבור. אם לא נאשר, לא יופיע המכשיר במחשב, אלא תופיע שגיאה.

כדי להיכנס ל-Shell הפנימי של המכשיר נריץ adb shell.

נהדר! אנחנו ב-Shell של המכשיר.

[3] תיארתי כאן לפי מכשירים מבוססי AOSP - שזה רוב המכשירים הפשוטים. במכשירים מחברות עם ממשק משתמש משלהם, למשל סמסונג, העיקרון דומה אך קצת שונה.

מכיוון שאנדרואיד רצה על קרנל לינוקס, זהו Shell יוניקסי די סטנדרטי:

```
C:\Users\Ashi\Documents\ADB_AppControl\adb>adb devices
* daemon not running; starting now at tcp:5037
* daemon started successfully
List of devices attached
Q3LY250730008334        device


C:\Users\Ashi\Documents\ADB_AppControl\adb>adb shell
k6789v1_64:/ $ ls
acct        config         dev              linkerconfig  odm          product                sys
apex        d              etc              lost+found    odm_dlkm     sdcard                 system
bin         data           img              metadata      oem          second_stage_resources  system_ext
bugreports  data_mirror    init             mnt           postinstall  shared_pref_backup     vendor
cache       debug_ramdisk  init.environ.rc  oat           proc         storage                vendor_dlkm
k6789v1_64:/ $
```

[שימו לב לפלט של ls - הוא מציג את כל התיקיות הקיימות בשורש של המכשיר. כדי להיכנס לאחסון הפנימי, נריץ cd sdcard]

אבל עדיין, אנחנו לא יכולים לעשות מה שאנחנו רוצים. אין לנו גישה למשתמש ה-Root כלל. בשביל זה, נגיע לשלב הבא, השרשת (Rooting) המכשיר והשגת הרשאות מלאות על המכשיר שלנו.

היכרות עם מנגנוני האימות

חשוב לדעת, שישנם שני מנגנוני אימות שהקושחה שצרובה על המכשיר מקורית ולא עברה שינויים:

1. נעילת בוטלואדר - לרוב מתייחסים למנגנון זה כמנגנון ההגנה היחיד, ומכלילים את כל תהליך ביטול האימות תחת השם "פתיחת בוטלואדר". בוטלואדר נעול משמעותו שהמכשיר לא יעלה באופן תקין אם צרובה עליו קושחה לא מקורית (ובתוכנות צריבה מסוימות, אף יחסום צריבה של קושחה לא מקורית), אך הוא אינו אחראי בעצמו על בדיקת תקינות הקושחה, וחשוב לדעת את זה. הוא בסך הכול בודק את החתימה של מחיצת ה-VBMETA, האם היא נחתמה עם המפתח הפרטי שהוגדר לו. במכשירים מסוימים, כגון גוגל פיקסל, ניתן לפתוח את הבוטלואדר, להגדיר למכשיר מפתח פרטי חדש (לא זה של המפעל), לצרוב קושחה שנחתמה על ידי מפתח זה, ולנעול בחזרה את הבוטלואדר. כך ניתן לצרוב רום מקוסטם, אך לשמור על מנגנוני האבטחה של המכשיר.

2. Android Verified Boot - או בראשי התיבות המקובלים - AVB. זהו המנגנון שמאמת את שלמות הקושחה. מחיצת ה-VBMETA אחראית עליו. במחיצה זו מאוחסן המידע על המחיצות השונות שאותם המנגנון צריך לאמת, כולל החתימות. מצורף פלט לדוגמא מכלי רשמי של גוגל בשם AVBTool לקריאה ויצירה של מחיצות VBMETA. כשהבוטלואדר נעול, הוא מאמת, כמו שאמרנו את מחיצת ה-VBMETA בלבד, ולכן אי אפשר להחליף אותה במחיצה עם חתימה אחרת. מנגנון ה-AVB הוא זה שמאמת את שלמות המחיצות האחרות. בדוגמא שלפנינו אפשר לראות שה-VBMETA הראשי לא מאמת באופן ישיר חלק מהמחיצות, אלא מאמת מחיצות VBMETA משניות שמאמתות אותם.

גם לאחר פתיחת הבוטלואדר, מנגנון ה-AVB עדיין פעיל! אם נצרוב רום מקוסטם, רוט, וכדומה - נצטרך לכבות אותו, או לחתום את השינויים שלנו וליצור מחיצת VBMETA חדשה. כיבוי מנגנון ה-AVB מתבצע בדרכים שונות בין מכשירים.

```
Minimum libavb version:   1.0
Header Block:             256 bytes
Authentication Block:     320 bytes
Auxiliary Block:          2752 bytes
Public key (sha1):        fb86b6a7475f8ae82ee4f731a1c8e86ee27a3399
Algorithm:                SHA256_RSA2048
Rollback Index:           2
Flags:                    0
Rollback Index Location:  0
Release String:           'avbtool 1.1.0'
Descriptors:
    Chain Partition descriptor:
      Partition Name:          boot
      Rollback Index Location: 3
      Public key (sha1):       090c5e80cc8d61dc5378625c74af4daa2033ab23
    Chain Partition descriptor:
      Partition Name:          vbmeta_system
      Rollback Index Location: 2
      Public key (sha1):       8c5586ce094272bc59ede02e91e6d3132dab523d
    Chain Partition descriptor:
      Partition Name:          vbmeta_vendor
      Rollback Index Location: 4
      Public key (sha1):       3b11d91d311f5eb26cd538705f88dcad85da9d9d
    Prop: com.android.build.dtbo.fingerprint -> 'TCL/T408DL/Gflip6_TF:11/RP1A.200720.011/KEE7:user/release-keys'
    Hash descriptor:
      Image Size:            53964 bytes
      Hash Algorithm:        sha256
      Partition Name:        dtbo
      Salt:                  c83987f11437da6abe42c27f0f93c4c6d501d8c68cfe6ed1820c5f484a51a3fc
      Digest:                00d78e0a2e99189cf563fd21b2db631cdc1baab0c0cf66eb4a679f4b00c1505c
      Flags:                 0
```

לסיכום שלב זה:

במצב של בוטלואדר נעול, אי אפשר להחליף מחיצת VBMETA (למעשה, אפשר, אך צריך שתהיה חתומה באותו המפתח של המחיצה המקורית) או להשבית את מנגנון ה-AVB. אך ברמה התאורטית, אין אימות על מחיצות אחרות, אך אז אימות ה-AVB ישבר והמכשיר לא יעלה באופן תקין.

מסקנה פרקטית: אם המחיצות שלנו שונות מהמקוריות, אך תואמות ל-VBMETA - הן יעבדו. כמובן, זה אפשרי רק במצב שיש לנו את המפתח הפרטי המקורי של המפעל כדי לחתום VBMETA חדש.

במצב של AVB מופעל - אי אפשר לשנות את המחיצות שמאומתות על ידו, גם אם הבוטלואדר פתוח.

מסקנה פרקטית: כדי לשנות את המחיצות, אנחנו חייבים לכבות את AVB ולא רק לפתוח את הבוטלואדר.

כל זה מאפשר לעדכן בקלות יחסית את המכשיר - אין צורך בכל עדכון לעדכן מחדש את הבוטלואדר, אלא רק לחתום את המחיצות וליצור VBMETA חדש שחתום עם המפתח שמאומת בבוטלואדר.

הערה חשובה: פתיחת בוטלואדר כרוכה באיפוס של המכשיר; זהו מנגנון הגנה מובן מאוד - מצד אחד אין דרך לפתוח את הבוטלואדר וככה לצרוב מערכות ערוכות בכל מיני רמות שיאפשרו לדלג על הסיסמא, ומצד שני גם אי אפשר לפתוח בוטלואדר ולצרוב מערכת עם רוגלות בלי להשאיר עקבות, אפילו אם הצלחנו להסתיר את האזהרות שמוצגות בכל הדלקה במכשיר שהבוטלואדר שלו נפתח. מעבר לזה, ביחד עם מנגנון

FRP נגד גניבות, שאחרי איפוס דורש חיבור לחשבון הגוגל האחרון שהיה מחובר במכשיר (גם במכשיר עם בוטלואדר נעול וקושחה תקנית ומקורית), זה מונע למעשה פתיחת בוטלואדר לא לגיטימית. כמובן, בחלק מהמכשירים יש דרכים לעקוף דרישות אלו... אבל אני מדבר כרגע על הדרך הרשמית. לכן, לפני פתיחת הבוטלואדר, אני ממליץ לוודא שאין מידע חשוב על המכשיר ולנתק את כל החשבונות מהמכשיר (גם גוגל, וגם חשבונות אחרים [למשל חשבונות של יצרן המכשיר, בסגנון Mi Account] שגם משמשים לעיתים לאימותים בסגנון FRP).

Rooting

אז פתחנו את הבוטלואדר, וכיבינו את מנגנון ה-AVB. עכשיו נראה איך אנחנו יכולים להשיג גישה למשתמש ה-Root ובעצם לקבל שליטה מלאה על המכשיר שלנו. לפני כן, חשוב לי לנפץ שני מיתוסים שמשום מה נפוצים ברשת ביחס לרוט:

א. מאפשר להתקין ROM-ים מקוסטמים. זה לא נכון. רוט יכול לאפשר עוד דרך צריבה (דרך פקודת dd, שעוד נדבר עליה), אבל חוץ מזה אין קשר. אולי חוץ ממכשירים חריגים מאוד, אפשר לצרוב ROM מקוסטם גם בלי רוט.

ב. בשביל רוט חייבים ריקוברי מקוסטם כדוגמת TWRP. שוב, לא נכון. כאן אומנם יש יותר קשר - אם יש TWRP אפשר לצרוב דרכו את מג'יסק כקובץ zip של עדכון והוא כבר משריש ונשאר רק להתקין את ה-APK של מג'יסק במכשיר בסוף, אבל זה רק קיצור תהליכים. אבל לא נקדים את המאוחר...

כמו שכתבתי למעלה, באנדרואיד אין דרך רשמית להסלמת הרשאות וקבלת גישת רוט. ההרשאות הכי גבוהות שאפשר לקבל בצורה לגיטימית הן הרשאות Device Owner, והן מוגבלות מאוד וזו פרוצדורה שלמה להתקין אפליקציה עם הרשאות אלו. אפשר להקביל את זה למצב ב-Windows - אין משתמש עם הרשאות מלאות. ב-Windows, אפילו משתמש ה-Admin מוגבל. אותו הדבר באנדרואיד; המשתמש יכול לעשות הרבה דברים, כולל התקנת והסרת תוכנות, אבל אין סיכוי שהוא יגע במערכת ההפעלה עצמה, אפילו עם הוא קיבל את ההרשאה הלגיטימית הכי גבוהה.

כדי להשיג גישת Root, נצטרך לערוך את הקושחה. אין קיצורי דרך, אין אפליקציות One-Click-Root (חוץ ממכשירים ישנים מאוד או במכשירים ספציפיים שנמצא בהם פירצת 0Day). זוהי כמובן דרישת אבטחה בסיסית, תחשבו מה היה קורה אם אפליקציה פשוטה הייתה יכולה להשיג הרשאות Root - היה הופך לחסר משמעות לחלוטין את העובדה שאין גישה למשתמש ה-Root במכשיר. אבל אין צורך להיבהל, יש דרך פשוטה מאוד.

תכירו - Magisk. מג'יסק, מעבר לפתרון רוט, הוא כלי רב עוצמה עם הרבה אפשרויות:

1. הגישה הכללית של מג'יסק - Systemless root, כלומר, רוט ללא צורך לפגוע במחיצת ה-System. תבינו לבד עד כמה זה נוח, מחיצת ה-System בעצמה לא נפגמת מעולם. אפליקציית מג'יסק עורכת רק את ה-Ramdisk כדי להשריש (Rooting) את המכשיר. ה-Ramdisk לרוב נמצא במחיצת boot, לעיתים ב-Recovery. עוד נדבר על זה.

2. כחלק מגישת ה-Systemless, מג'יסק מספקת גם דרך מבריקה ל"הזרקת" שינויים למחיצת ה-System ללא שינוי בפועל: מודולי מג'יסק. אלו הם קבצי zip עם קבצים וסקריפטים להשפעה על המערכת, אני ממליץ לקרוא את הדוקומנטציה של מודולי מג'יסק להבנה מעמיקה יותר, כאן אסביר בקיצור.

הרעיון העקרוני של המודולים הללו הוא שמג'יסק עושה mount לתיקיה במחיצת data/ אליה מחולצים קבצי ה-zip של המודולים ככה שהמכשיר מתייחס לשינויים שמתבצעים בתיקיה כאל שינויים במחיצות ה-System. כמו כן, מג'יסק מריץ סקריפט שנמצא בקובץ ה-zip בשלב ההתקנה, ככה שאפשר להוסיף שם פקודות נצרכות. תוך כדי כתיבת המאמר נתקלתי בסוג נוסף וחדש יחסית של מודולים - מודולי Zygisk, שמאפשרים להריץ קוד ב-C++ ישירות ל-Zygote. לא ניסיתי את זה מעולם, אז אין לי כל כך איך להרחיב על זה. אולי במאמר אחר מתי שהוא.

3. שליטה למי יש הרשאת רוט - הגדרה חשובה שקיימת גם בכלי רוט אחרים - לא כל מי שינסה לקבל גישת רוט במכשיר שלכם יקבל, אלא יוצג לכם פופ-אפ לבחירה אם לאשר או לדחות את הבקשה. אתם יכולים גם להגדיר שכל בקשה תיענה תמיד בחיוב (לא מומלץ כלל כמובן!) או לדחות תמיד (אם מטרתכם היא, לדוגמא, רק המודולים).

4. עד גרסה 23 של מג'יסק, היה בתוך מג'יסק אופציה שנקראה MagiskHide שכשמה כן היא - הסתירה את הרוט מאפליקציות, ככה שאפליקציות מאובטחות, כגון אפליקציות בנקאיות, לא יזהו שאבטחת המכשיר נפגעה וירוצו על המכשיר.

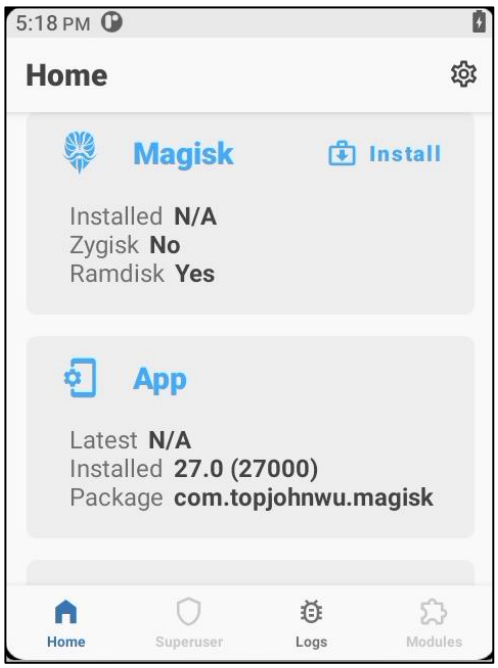
ואז... אז גוגל גייסה את John Wu, המפתח הראשי של מג'יסק, לצוות האבטחה של אנדרואיד. מתחילים לזהות את הבעיה? אומנם גוגל הסכימה שהוא ימשיך לפתח את מג'יסק, למרבה המזל, אך דרשה שיסיר את MagiskHide. המפתח הדגיש גם עוד נושא-הסתרת הרוט הפכה להיות משחק חתול ועכבר, של בדיקות אבטחה, זיוף התוצאות ובדיקות האם התוצאות מזויפות, וחוזר חלילה, והוא מעדיף להקדיש את זמנו לפיתוח מג'יסק ולא לבזבז את הזמן על זה. במקום זה, הוא הציג כלי חדש - Zygisk DenyList - שמאפשר לבחור איזה אפליקציות ירוצו במעין סביבה מבודדת משאר המערכת, ואכן לאפליקציות ברמת אבטחה נמוכה זה מספיק, אך אפליקציות ברמה גבוהה עדיין מזהות רוט - כי מג'יסק לא מזייף תוצאות של בדיקות. מפה התפתח עולם חדש של מודולי מג'יסק לזיוף תוצאות והסתרת רוט (למשל, ראו כאן) - וכמו שניבא ג'ון וו, זה הפך למשחק חתול ועכבר ממש.

כמובן, היה גם מי שלקח את הקוד של מג'יסק (קוד פתוח, אחרי הכול), עשה fork וניסה לתחזק את MagiskHide. בתחילה, היה זה משתמש בגיטהאב בשם HuskyDG, אבל כעבור כמה זמן הוא מחק לחלוטין את הפרויקט. משתמש אחר תחזק את ה-fork, אבל כיום גם הוא במצב קריאה בלבד ולא מתוחזק. לסיכום: יש במג'יסק גם אופציה של הסתרת הרוט (באמצעות DenyList ומודולים משלימים) אך זה מסובך מבעבר.

כמובן, למרות גישת ה-Systemless שהזכרנו למעלה, מג'יסק נותנת גישת רוט מלאה, ומאפשרת גם לערוך באופן ישיר את הקושחה, שזה מה שאנחנו נעשה.

נוריד את אפליקציית Magisk מדף הגיטהאב הרשמי, נתקין אותה ונפתח אותה. אומנם אני משתמש בגרסה לא הכי עדכנית (27) אבל זה בגלל שבגרסאות החדשות יש קצת בעיות עם מודולים. מניח שיתוקן מתי שהוא, אז ממליץ לכם לנסות קודם את ההכי עדכני.

נראה מסך כזה:

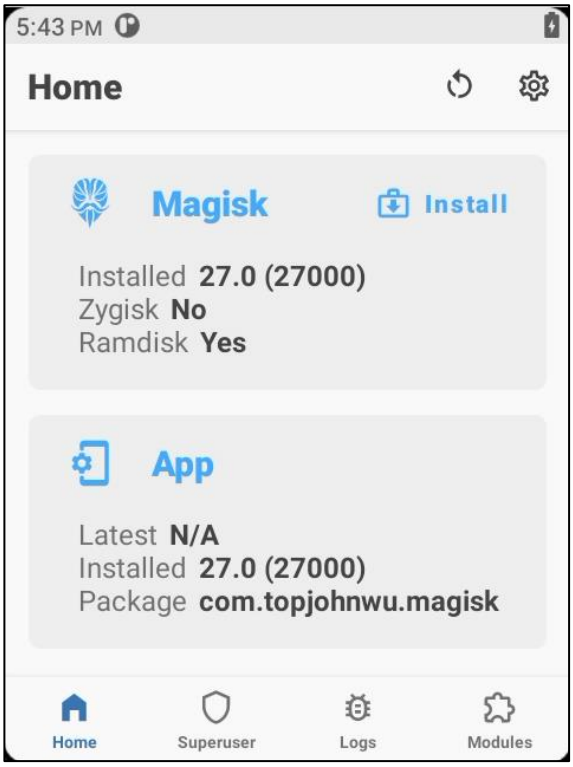


בחלק הראשון, פרטים על שירות מג'יסק, בחלק השני על האפליקציה. כמו שאנחנו רואים - שירות מג'יסק לא מותקן במכשיר וה-Ramdisk מסומן ב-Yes, כלומר הוא נמצא, כרגיל, במחיצת boot. אם כתוב no, תצטרכו את התהליך המתואר כאן לעשות גם למחיצת recovery.

אם כן, אנחנו צריכים להתקין את שירות מג'יסק. אבל כמו שהסברתי למעלה, כמובן, אי אפשר להתקין סתם ככה על מכשיר פעיל דרך אפליקציה. אז אפליקציית מג'יסק עורכת את קובץ ה-boot.img מהקושחה. נעתיק אותו למכשיר, ונבחר במג'יסק באופציה Install (עם הסמל של המזוודה והחץ כלפי מטה). במסך שיופיע נסמן את האופציה Select and Patch a file, ומיד יפתח לנו סייר הקבצים של המכשיר. נבחר את קובץ ה-boot.img ונלחץ על LET'S GO. יפתח מסך שיראה את התקדמות תהליך עריכת מחיצת ה-boot.

נחכה לסיום, אמור להיות פלט כזה:

נלך לתיקיית ההורדות שלנו, יהיה שם קובץ ששמו מתחיל במילים magisk_patched. זהו ה-boot המתוקן. נעביר אותו למחשב ונצרוב לפי ההוראות בהמשך המאמר. אחרי הצריבה נפעיל את המכשיר, נכנס לאפליקציית מג'יסק, יתכן שיופיע לנו חלון לאשר התקנה נוספת. נאשר. אם תופיע הודעה שהמכשיר יופעל מחדש בעוד 5 שניות, נחכה שיופעל מחדש, אם יפתח מסך בחירת דרך ההתקנה (שאגב, נראה בו כבר יותר אופציות ולא רק את Select and Patch file), זו תקלה של מג'יסק שקורית לפעמים, נתעלם:



כעת אפשר לראות:

א. כעת, בסעיף Installed, במקום N/A כתובה גרסת מג'יסק.

ב. בסרגל הניווט למטה, הלחצנים Superuser - שפותח לנו מסך שיראה את כל האפליקציות שקיבלו הרשאות רוט, ו-Modules - שפותח את מסך ניהול המודולים, הפכו לשחורים (במקום אפורים) ומגיבים ללחיצות.

נפתח adb, נקליד adb shell ואז su (מקבילה ל-sudo בלינוקס). נאשר בפופ-אפ שקופץ על מסך הטלפון את הענקת ההרשאות ל-Shell, והנה, אנחנו בטרמינל עם הרשאות root!

הכנה של מחיצות המערכת לקריאה וכתיבה (RW)

יש לנו רוט, אבל אם ננסה לשנות משהו במחיצות המערכת דרך ה-Shell נקבל שגיאה שזו מחיצה לקריאה בלבד. אפשר לפתור את זה עם פקודת remount:

```
mount -ro remount,rw /system
```

או משהו בסגנון הזה (משתנה בין מכשירים). כמובן, דורש הרצה כ-root.

בדרך כלל, כשהפקודה הזו מספיקה, סיירי קבצים עם הרשאות רוט יעבדו גם בלי להריץ אותה ידנית - הם בעצמם מריצים במכשיר את הפקודות האלו.

אך הרבה פעמים, לא הפקודה הזו ולא אפליקציות סיירי רוט למיניהם יעזרו, כי המערכת מוגדרת לקריאה בלבד בדרכים יותר מתוחכמות, למשל בגלקסי A06 מחיצות המערכת משתמשות במערכת הקבצים erofs (Enhanced Read-Only File System), שכשמה כן היא, לקריאה בלבד, והפקודה הזו פשוט לא משפיעה.

כמו כן, הרבה פעמים מחיצות המערכת מלאות עד אפס מקום באופן מילולי - לא נשאר בהם נפח פנוי, ולעיתים רבות אם נמחק רק קבצים לא קריטיים לפעילות המכשיר זה לא יספיק. (במיוחד לאור העובדה שחלק מהמקום ישתחרר רק אחרי הפעלה מחדש). כדי לפתור את בעיה זו יש לשנות את מערכת הקבצים של מחיצות המערכת, ההגדרות של מחיצת ה-super, ולהגדיל את גודל המחיצות. נשמע מפחיד? בצדק, יש סיבה. לעשות את כל זה ידני זה הרבה עבודה. אבל שוב, יש לנו כלי מאוד נוח: מודול המג'יסק RO2RW.

שימו לב! הכלי הזה דורש כמה וכמה ג'יגות פנויות באחסון הפנימי (לא כרטיס זיכרון) של המכשיר.

המודול הזה מוסיף כלי למכשיר בשם RO2RW שאותו נריץ דרך ה-Shell (דרך אפליקציות טרמינל כגון Termux, או דרך adb shell מהמחשב). הכלי הזה יגבה את מחיצת ה-super (עם פקודת dd), יפרק אותה, ישנה דברים לפי ההגדרות שלנו (למשל, יגדיל את המחיצות, ייתן לנו אופציה להסיר אפליקציות, וכדומה) ויארוז את ה-super בחזרה כשהמחיצות ניתנות לכתיבה.

נוריד את המודול, ונעתיק אותו (ללא חילוץ! את קובץ ה-zip) לאחסון של המכשיר שלנו. במסך הראשי של אפליקציית מג'יסק נלחץ על Modules (הסמל של חתיכת פאזל) ונבחר Install from storage. יפתח סייר הקבצים, נבחר את קובץ ה-zip ונאשר את ההתקנה.

בסיום ההתקנה, נתבקש להפעיל מחדש את הטלפון. אם ההתקנה נכשלת, אז כמו שאמרתי, כדאי להשתמש בגרסת מג'יסק ישנה יותר, כמו 27 בה אני משתמש.

נריץ ב-Shell את הפקודה ro2rw כרוט, ונראה מסך כזה:

```
- Aka RO2RW/EROFS2RW/F2FS2RW
******************
- StableBeta-v3.7.2.1
******************
- By @LeeGarChat
******************
- ARCH: arm64
- Active slot: _a
- Force start: false

- Paths directories:
- TMP_NEO = /data/local/TMP_NEO
- TMP_IMGS = /data/local/TMP_NEO/imgs
- NEO_OUT_LOGS = /data/media/0/NEO.LOGS
- Main_dir = /data
- OUT_SUPER_DIR = /data/media/0/RO2RW_SUPER


- Super found
- Found system mapper block



- You want make/install new RW super or check
  RW and free size?

  1) [Make/Install]
  2) [Check free size]
  3) [EXIT]
 - Select num:
```

כמובן, נבחר 1. בשלב הבא, נצטרך לבחור בכמה להגדיל את המחיצות. אני בדרך כלל בוחר את אופציה 4 - אבל לא מסיבה מיוחדת כל שהיא. יופיע מסך לאישור - נלחץ על 1. (Continue).

בשלב הבא, תופיע שאלה אם להשבית את הצפנת נתוני המשתמש בקושחה. לא מומלץ בעיני, אז נלחץ על 1:

```
**==================================***

- Do you also want to install the Disable Force
  Encryption (DFE) patch? WARNING: Only Use This
  if You Know What You're Doing. Else Press "SKIP"

  1) [SKIP]
  2) [Yes]
     If your data is encrypted you need to format
     data before running it into the system with
     new super
  3) [No]
     If your data is not encrypted and you have used
     DFE before and the ROM is not patched with DFE
     (ROM is encrypted), you need to format data
     before running into the system with new super
  4) [EXIT]
 - Select num:
```

עכשיו יתחיל תהליך פירוק, העתקת ועיבוד המחיצות שייקח זמן. נמתין.

בסיום התהליך נתבקש ללחוץ על כל מקש להמשך. ובכן, לא באמת כל מקש - לפחות כשזה בשימוש דרך adb במחשב - עובד. מקש אנטר עובד. בשלב הבא תוצג שאלה אם אנחנו רוצים דרך הסקריפט למחוק אפליקציות מסוימות. נבחר 1, להמשך, אך שוב 1 כדי לדלג על השלב הזה. עכשיו תוצג שאלה אם ליצור גיבוי של ה-super המקורי. זה יוסיף הרבה זמן, אך אם אין לנו את המקורי, זה קריטי. אם יש, נבחר 2 ונדלג. בשלב הבא - בין אם בחרנו לגבות ובין אם לא, נצטרך לבחור באיזה צורה אנחנו רוצים את ה-super. נבחר ב-1. נמתין לסיום יצירת ה-super, ותוצג לנו שאלה האם אנחנו רוצים לבטל את אימות המחיצות או ליצור קבצי vbmeta תואמים. עדיף לבטל את האימות. הקובץ שנוצר יהיה במיקום הזה באחסון הפנימי:

```
RO2RW_SUPER/super.rw.sparse.fastboot.img
```

נעתיק למחשב, ונצרוב לפי ההוראות שאציג בהמשך.

מבט כללי על מחיצות המערכת

נסתכל רגע על מבנה קלאסי של מחיצת ה-System. כאמור, גם מחיצות vendor ו-product עושות שימוש באותו מבנה עקרוני:

Name	Size	Packed Size	Mode	Modified	Created	Accessed
apex	370 941 952	369 860 608	drwxr-xr-x	2009-01-01...	2009-01-01...	2009-01-01...
app	285 205 645	284 827 648	drwxr-xr-x	2009-01-01...	2009-01-01...	2009-01-01...
bin	45 650 168	46 067 712	drwxr-x--x	2009-01-01...	2009-01-01...	2009-01-01...
etc	8 232 843	9 265 152	drwxr-xr-x	2009-01-01...	2009-01-01...	2009-01-01...
fonts	77 854 040	78 225 408	drwxr-xr-x	2009-01-01...	2009-01-01...	2009-01-01...
framework	181 571 996	182 255 616	drwxr-xr-x	2009-01-01...	2009-01-01...	2009-01-01...
lib	86 392 434	87 199 744	drwxr-xr-x	2009-01-01...	2009-01-01...	2009-01-01...
lib64	183 692 539	184 258 560	drwxr-xr-x	2009-01-01...	2009-01-01...	2009-01-01...
media	9 407 906	9 416 704	drwxr-xr-x	2009-01-01...	2009-01-01...	2009-01-01...
priv-app	57 958 442	57 962 496	drwxr-xr-x	2009-01-01...	2009-01-01...	2009-01-01...
product	0	0	lrw-r--r--	2009-01-01...	2009-01-01...	2009-01-01...
system_dlkm	952	4 096	drwxr-xr-x	2009-01-01...	2009-01-01...	2009-01-01...
system_ext	0	0	lrw-r--r--	2009-01-01...	2009-01-01...	2009-01-01...
usr	2 769 341	3 514 368	drwxr-xr-x	2009-01-01...	2009-01-01...	2009-01-01...
vendor	0	0	lrw-r--r--	2009-01-01...	2009-01-01...	2009-01-01...
build.prop	8 199	12 288	-rw-------	2009-01-01...	2009-01-01...	2009-01-01...

פתחתי את קובץ ה-Img עם 7zip, אך זה לקריאה בלבד - אי אפשר לערוך ככה את המחיצות.

apex - תיקיה שמכילה מודולי APEX, כלומר רכיבים קריטיים מהמערכת שגוגל הולכת ומעבירה (מאז אנדרואיד 10) למבנה מודולרי; מודולי APEX מתעדכנים דרך שירותי גוגל, ברקע, ללא התערבות המשתמש או יצרן המכשיר. מאפשר עדכונים של רכיבים מהירים ואוניברסליים של חלקים רגישים במערכת. מעין "הרחבה" או המשך של העקרונות של Project Treble. ישנם כמובן גם מודולי APEX של היצרן, שמתעדכנים דרכו ולא דרך גוגל, אך עדיין זה יתרון - עדכון של חלקים קריטיים במערכת בלי צורך בעדכון כל ה-ROM. גם אם נצליח לערוך מודול APEX (בעיקרון, התיקיה הזו מוגנת יותר משאר התיקיות במערכת ונשארת לפעמים לקריאה בלבד גם אחרי RO2RW וכדומה), הוא עלול להתעדכן ברקע ובעצם לחזור למקור... לכן במקרה שאנחנו נרצה לערוך משהו שנמצא כמודול APEX, צריך למצוא דרך אחרת. למשל, ה-Daemon של ADB באנדרואיד 14 הוא מודול APEX, וכשרציתי לבטל בגלקסי A06 את האופציה ל-ADB זה קצת סיבך אותי. הרחבתי על זה בבלוג שלי.

app - כשמה כן היא; מכילה את האפליקציות של המכשיר (כגון סייר הקבצים, נגן המוזיקה, וכדומה). אפליקציות שדורשות הרשאות ברמה גבוהה יותר יהיו ב-priv-app - עליה נסביר בהמשך.

bin - קבצים בינאריים, בקוד נייטיבי, שרצים ישירות על קרנל הלינוקס של המכשיר (ולא אפליקציות שרצות על גבי Android Runtime). לדוגמא: wpa_suppcliant הוא הקובץ שאחראי על ה-WIFI.

etc - כשמה כן היא: מכילה עוד הרבה קבצים (ותתי תיקיות) שאחראים על קונפיגורציות של המערכת. למשל, תיקיית permissions שאחראית על ההרשאות, או קובץ ה-hosts, ועוד.

fonts - כשמה כן היא.

framework - כשמה כן היא, מכילה את ה-framework של המכשיר, כלומר את ממשקי ה-API, הגדרות UI, שירותים בסיסיים וכדומה. הקבצים בה נמצאים בפורמט jar או בפורמט dex ודומיו - מקומפל מראש כדי לחסוך זמן ריצה.

lib/lib64 - מכילות ספריות לשימוש האפליקציות. בגדול, יש 3 סוגי ספריות:

1. ספריות Java עדכניות - הרי כל האפליקציות רצות על Java, ויש צורך שהמכשיר יתמוך בספריות Java.
2. ספריות מערכת - לרוב, ספריות לינוקס סטנדרטיות, למשל עבור Bluetooth. יש גם ספריות מיוחדות לאנדרואיד, כמובן, כמו libart שאחראית על Android Runtime - סביבת הריצה של אנדרואיד.
3. ספריות יצרן - ספריות לתקשורת בין רכיבי החומרה, כגון צ'יפ ה-GPS, ומערכת ההפעלה.

media - כשמה כן היא, מכילה את קבצי המדיה שבשימוש במכשיר. למשל; אנימציית ההדלקה, צלילים, וכדומה.

priv-app - קיצור של Privileged apps. כאן יהיו אפליקציות ש:

1. צריכות הרשאות גבוהות מהרגיל. לא שיש להם הרשאת רוט, אך יש הרשאות רגישות שרק אפליקציות שנמצאות בתיקיה הזו יכולות לקבל. למשל, הרשאה של התקנת אפליקציה (בפועל, ולא רק קריאה ל-Package Manager המובנה של אנדרואיד או שימוש בפקודות shell).
2. אפליקציות שנמצאות בתיקיה הזו יכולות לקבל הרשאות על בסיס קבצי XML בתיקיית etc/permissions, ואז הן יקבלו את ההרשאות גם ללא התערבות המשתמש.

נרחיב יותר על תיקיה זו בהמשך.

system_ext/system_dlkm/odm - מחיצות מערכת נוספות על הסטנדרטיות (system, vendor,product). לא קיימות בכל מכשיר. לא נפרט עליהם, כי הם זהות במבנה ובתפקיד העקרוניים לשאר מחיצות המערכת.

vendor/product - מחיצות מערכת.

לעיתים חלק ממחיצות המערכת לא ממומשות כמחיצות ממש אלא רק כתיקיות בתוך ה-system, ולכן נמצא אותן (או את חלקן) כתיקיות בתוך ה-system, אך לרוב יהיו אלו תיקיות ריקות - שכשיהיו במכשיר (ולא בפירוק על המחשב) יתפקדו כקיצור דרך למחיצה האמיתית.

usr - מתפקדת בעיקרון כמו תיקיית usr בלינוקס סטנדרטי.

קבצי prop - קבצים שמכילים הגדרות שונות למערכת ההפעלה, בצורה של flags שבנויים משם (name) וערך (value).

למשל:

```
ro.build.version.sdk=34
```

ה-name הוא ro.build.version.sdk - כלומר, גרסת ה-sdk של המכשיר. ה-value הוא 34, כלומר, sdk 34 שהוא אנדרואיד 14. אפשר לשנות חלק מהגדרות אלו, ונדבר על זה בהמשך.

מטבע הדברים, רוב השינויים בקושחות ערוכות (אני מדבר כאן על קושחות שעברו עריכה, ולא על ROM-ים חדשים שנבנו עבור המכשיר ישירות מ-AOSP, כגון LineageOS) מתבצעים בתיקיות האפליקציות (app/priv-app). כמו כן, מתבצעים שינויים בקבצי ה-prop, ולעיתים גם בתיקיית etc. מעבר לזה, אין הרבה שינויים.

APK Reversing

לפני שניגע בנושא המוזכר בכותרת - עריכת קבצי ה-APK, נסביר רגע איך להוסיף ולהסיר אפליקציות מהקושחה. למחוק, זה קל. פשוט... למחוק. חשוב כמובן לשים לב שאנחנו יודעים טוב מה אנחנו מוחקים, אחרת המכשיר עלול בקלות להיהרס.

יש דרך מאוד נוחה למצוא מה ה-APK של אפליקציה מסוימת. התקינו על המכשיר את האפליקציה הזו, הפעילו אותה ותנו את ההרשאות שהיא מבקשת. עכשיו תופיע לכם על המסך, בחלק העליון, מעין חלונית עם 2 שורות. בשורה הראשונה נראה את ה-PackageName של האפליקציה ובשניה - את הנתיב המדויק (בתוך האפליקציה) של האקטיביטי שאנחנו נמצאים בו.

נריץ ב-Shell של המכשיר (דרך adb, או דרך אפליקציית טרמינל כלשהי - רק שבאפליקציית טרמינל זה ידרוש הרצה כרוט) את הפקודה הבאה

```
pm path your.package.name
```

ונראה פלט כזה:

```
/system/app/YourApp/YourApp.apk
```

שהוא כמובן הנתיב לקובץ ה-APK.

להוסיף זה גם די קל; יוצרים בתוך התיקיה שאליה אנחנו רוצים להוסיף את האפליקציה (app או priv-app) תיקיה בשם של קובץ ה-apk שלנו ואליה מעתיקים את קובץ ה-apk.

בסוף נקבל נתיב כזה:

```
/(selected_system_partition)/(selected_folder)/MyApp/MyApp.apk
```

לתיקיה שיצרנו צריכות להיות הרשאות 755 ולקובץ ה-apk הרשאות 644. זה נכון לאפליקציות רגילות, יש מקרים חריגים כמובן.

הוספת אפליקציה ל-priv-app:

כמו שאמרתי, אפשר לתת הרשאות מראש לאפליקציות שנמצאות בתיקיה הזו על ידי קבצי xml בתיקיית /(selected_system_partition)/etc/permissions.

על מנת להבין את זה יותר טוב, נסתכל על מודול המג'יסק של AuroraServices.

בנתיב /system/etc/permissions/ בתוך ה-zip, נראה קובץ xml שזה הקוד שלו:

```xml
<?xml version="1.0" encoding="utf-8"?>
<permissions>
    <privapp-permissions package="com.aurora.services">
        <permission name="android.permission.DELETE_PACKAGES"/>
        <permission name="android.permission.INSTALL_PACKAGES"/>
    </privapp-permissions>
</permissions>
```

כמו שאפשר לראות, בשורה השלישית יש את ה-PackageName של האפליקציה, ובשורה השלישית והרביעית את שמות ההרשאות שאנחנו רוצים להעניק לאפליקציה. (במקרה הזה, התקנה והסרת אפליקציות - כאמור, הרשאות שבכלל אפשר לקבל רק אם האפליקציה נמצאת ב-priv-app).

את ההרשאות שהאפליקציה דורשת נוכל לראות בקובץ ה-AndroidManifest.xml שלה. ההרשאות של קובץ ה-XML ב-/system/etc/permissions צריכות להיות 644.

עכשיו - עריכת קבצי ה-apk. חשוב לדעת, שברוב מוחלט של המכשירים עם גרסאות אנדרואיד מודרניות, ישנה הגנה שמאמתת את החתימה של אפליקציות המערכת. אנחנו יכולים להוסיף אפליקציות משלנו, אבל אם נערוך אפליקציית מערכת, לרוב המכשיר לא יידלק. יש דרכים רבות לבטל הגנה זו, ובגלל שהם משתנות בין גרסאות האנדרואיד השונות לא נדון בהן כאן.

אני לרוב משתמש בכלי בשם Mt Manager, שהוא כלי רב עוצמה, אך חלק גדול מהפונקציות שלו מוגבלות למשתמש בתשלום, שלמי שמתעסק בזה הרבה משתלם מאוד, אבל כאן במאמר בחלקים שהוא דורש תשלום אשתמש בכלים אחרים, חינמיים. כמו כן, הכלי הזה מזוהה משום מה כווירוס ברוב האנטי וירוסים. ממחקר שעשיתי בעבר לא מצאתי בו משהו חשוד באמת, אבל... All at your own risk.

ראשית, נראה איך לפרק (Decompile) את קובץ ה-APK. בדרך כלל משתמשים בכלי בשם ApkTool או בתוכנות שמבוססות עליו, אם עושה בעיות אפשר להשתמש בכלי מאוד דומה בשם APKEditor. (שני הכלים דורשים Java מותקן במחשב).

לפירוק, נריץ:

```
java -jar apktool.jar d appName.apk
```

נראה על המסך פלט שמפרט את שלבי הפירוק, התהליך לוקח קצת זמן.

לקימפול מחדש נריץ:

```
java -jar apktool.jar d app-debug.apk
```

ונראה פלט דומה.

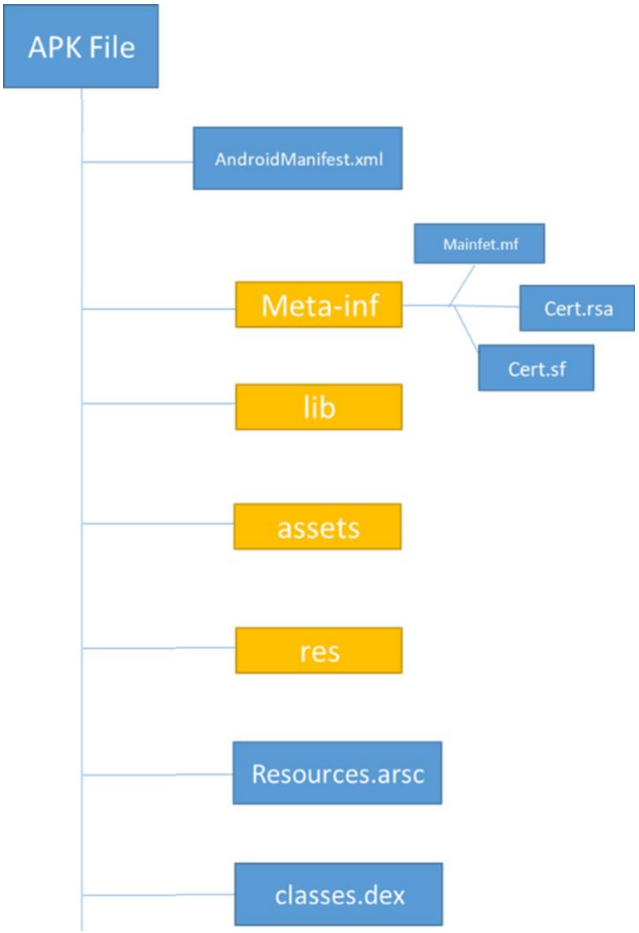
ApkTool הוא כלי מאוד נפוץ, ולכן הסברתי עליו, אך גם עם הרבה באגים, לכן הבאתי גם את האופציה של APKEditor.

שימו לב שבסיום העריכה הקובץ לא חתום; זה אומר שא"א להתקין אותו. נצטרך לחתום אותו מחדש - הכי קל עם Mt Manager, אבל כדי לחתום עם מפתח משלנו ולא הדיפולטיבי, צריך חשבון בתשלום. יש עוד כלים כמובן ברשת. לא אאריך בזה כרגע, כי אפליקציות מערכת לא חותמים אחרי עריכה, וזה הנושא שלנו.

ועכשיו, נצלול להסבר על מבנה קובץ ה-APK.

הסבר כללי על מבנה קובץ APK

כך נראה עץ התיקיות של קובץ APK:



[תיקיות בכתום, קבצים בכחול. נכון גם לתרשימים להלן]

AndroidManifest.xml - קובץ זה מכיל מידע קריטי על האפליקציה כמו שם החבילה, פעילויות (activities), הרשאות (permissions) ועוד.

META-INF/ - תיקיה זו מכילה קבצים הקשורים לאימות והחתימה הדיגיטלית של האפליקציה.

- **MANIFEST.MF** - קובץ זה מכיל רשימה של כל הקבצים ב-APK עם חתימותיהם.
- **CERT.SF ו-CERT.RSA** - קבצים אלו מכילים את החתימה הדיגיטלית של האפליקציה.

lib/ - אופציונאלי. תיקיה זו מכילה ספריות קוד (native libraries) באפליקציות מערכת, לרוב, תיקיה זו מחולצת ונמצאת או ליד קובץ ה-apk בתיקיה שלו, או בתיקיות ה-lib הראשיות של המערכת.

assets/ - אופציונאלי. תיקיה זו מכילה קבצים נוספים שמשתמשי האפליקציה יכולים לגשת אליהם בזמן ריצה. למשל, קבצי סאונד.

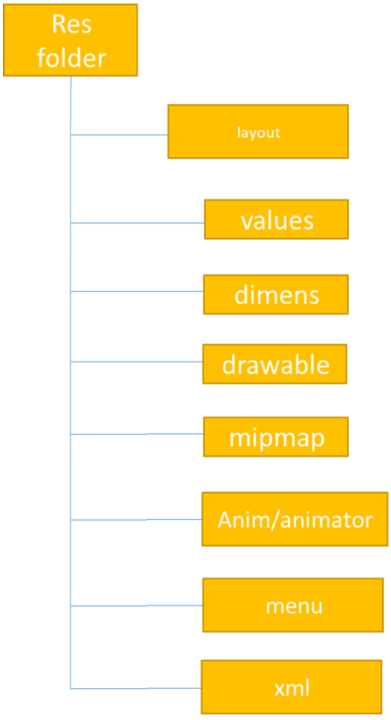
/res - תיקיה זו מכילה את כל משאבי האפליקציה כמו קבצי תמונה, קבצי XML, פריסות (layouts) ועוד.

classes.dex - קובץ זה מכיל את הקוד הבינארי של האפליקציה בפורמט Dalvik Executable (DEX). או בשם המקובל - קוד smali.

resources.arsc - קובץ זה מכיל מידע על המשאבים של האפליקציה בצורה מכווצת. (ועורכי ה-APK מבוססי APKTOOL, מחלצים את התיקיות והקבצים שבתוכו לתוך תיקיית res).

מה שמעניין אותנו לרוב אלו קבצי classes.dex ו-resources.arsc, וכן תיקיית res.

אני אאחד את ההסבר על תיקיית res ועל קובץ resources.arsc, מכיוון שגם ברמת קוד המקור וגם ברוב עורכי ה-APK, לאחר הפירוק, תכולת הקובץ הזו נמצאת בתיקיית res. אומנם, באפליקציית Mt Manager שבה אני משתמש לרוב הקובץ עומד בפני עצמו, אך העקרונות יהיו אותו דבר. זהו המבנה:



res/layout - תיקיה זו מכילה קבצי XML המגדירים את הפריסות (layouts) של המסכים באפליקציה. כל קובץ XML מגדיר את מבנה רכיבי הממשק על המסך, בהתאם ל-Activity אליו הוא משוייך.

res/values - תיקיה זו מכילה קבצי XML המגדירים ערכים שונים כמו מחרוזות, צבעים, סגנונות ועוד.

קבצים בתיקיות res/values: בתוך תיקיה זו, יהיו תיקיות של השפות השונות, ובתוכם קבצי ה-XML השונים. values-en - אנגלית values-ar - ערבית values-iw - עברית. לפעמים מוסיפים גם את האזור values-en-rUS - אנגלית ארה"ב values-iw-rIL - עברית ישראל.

strings.xml - מחרוזות טקסט (קבצי השפה). עכשיו אנחנו יודעים איך לתרגם אפליקציות...

colors.xml - הגדרות צבעים.

dimens.xml - הגדרות גדלים וכדומה.

styles.xml - הגדרות סגנונות עיצוב כלליים באפליקציה.

res/drawable - קבצי גרפיקה כמו תמונות וקבצי XML המגדירים צורות וציורים מורכבים. לעיתים יהיו כמה תיקיות, בהתאם לפריסות מסך שונות של מכשירים נתמכים.

res/mipmap - אייקונים של האפליקציה בגדלים שונים עבור מכשירים שונים. בדרך כלל ישנם תת-תיקיות כמו mipmap-xxhdpi, mipmap-xhdpi, mipmap-hdpi, mipmap-mdpi ו-mipmap-xxxhdpi.

res/anim ו**res/animator-** - מכילות קבצי XML המגדירים אנימציות ואנימטורים שמשתמשים בהם באפליקציה.

res/menu - קבצי XML המגדירים תפריטים שמופיעים באפליקציה, כגון תפריטי אפשרויות ותפריטי קונטקסט.

res/xml - קבצי XML כלליים שניתן להשתמש בהם להגדרות שונות של האפליקציה, כמו SharedPrefences, הגדרות אבטחה ועוד.

אחרי תהליך הפירוק, קבצי ה-XML מתקבלים קריאים לחלוטין, והעריכה שלהם די קלה. אז לא אאריך כאן - פשוט צריך ידע בפיתוח אפליקציות לאנדרואיד כדי להבין את ה-XML.

קובץ ה-Dex

הקובץ הזה מכיל את החלק הכי מעניין מבחינתנו - הקוד האמיתי של האפליקציה, ועריכה שלו היא מורכבת - אבל גם יכולה להביא לתוצאות משמעותיות מאוד. בדוגמא שאביא, אנחנו מבטלים תלות של אפליקציה בכמה דגלים ב-build.prop ובכך שאפליקציות אחרות יהיו גם הן במכשיר.

במכשיר Qin F22Pro יש אפליקציית שלט אינפרא אדום מובנית מראש. הבעיה היא, שאם עושים root ומוחקים חלק מה-Bloatware שהיצרן הכניס לקושחה - האפליקציה לא עובדת, בכניסה אליה מוצגת שגיאה "System environment error!" ומיד האפליקציה נסגרת. אז החלטתי לפצח את ההגנה הזו ולערוך את האפליקציה ככה שתעבוד בכל מקרה.

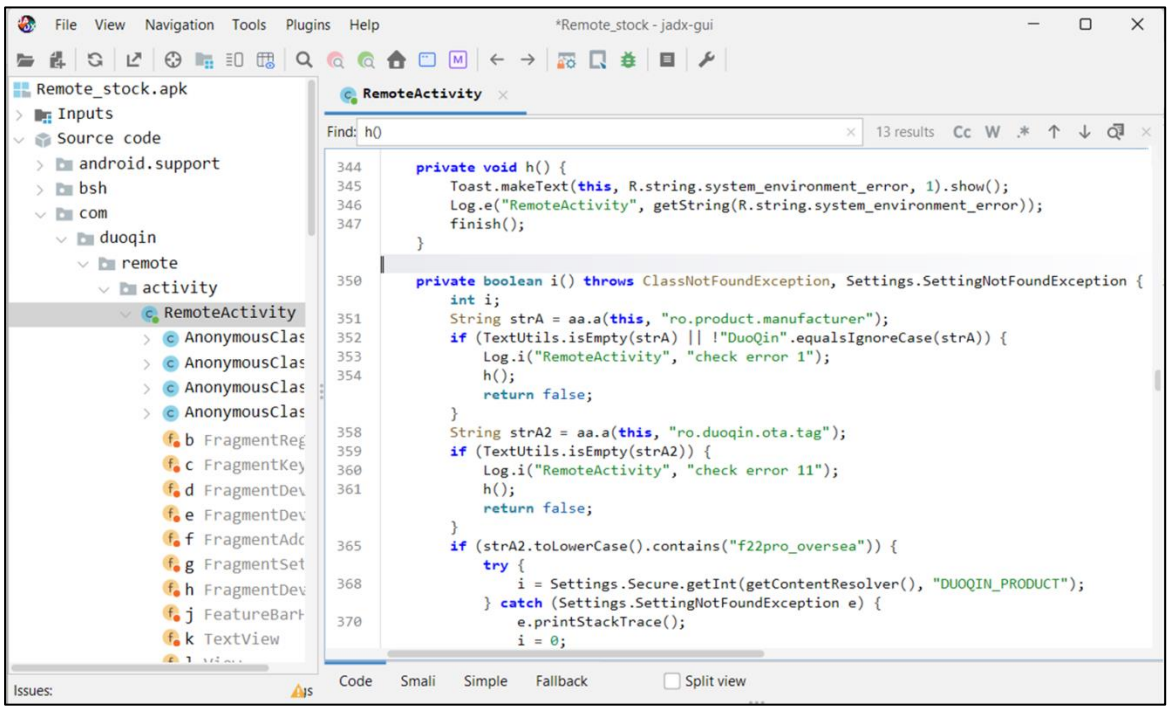
הטכניקה שהשתמשתי בה היא כזו:

1. מציאת הסטרינג הרלוונטי של הודעת השגיאה בקובץ ה-APK.
2. מציאת ה-Class שגורם להצגת השגיאה.
3. המרת קוד ה-Smali לקוד Java. כדי שאבין מה קורה בדיוק בקוד. בכל זאת, קוד Smali הוא לא כל כך קריא, לעומת קוד Java פשוט.
4. עריכת קוד ה-Smali וביטול הבדיקה שגורמת לשגיאה. (חובה לערוך ברמת ה-Smali, כי קוד ה-Java שמתקבל לא תקין ב-100% באמת ולא ניתן לקימפול בעצמו).

שלב ראשון - עם Mt Manager: פתחתי את קובץ ה-APK, נכנסתי לקובץ ה-Resources.arsc וחיפשתי את הסטרינג. מצאתי אותו, אז העתקתי את ה-id שלו, שבמקרה שלנו הוא 7f0a004a. חיפשתי את ה-id הזה בקובץ ה-Classes.dex ומצאתי אותו בנתיב com/douqin/remote/activity/RemoteActivity. צריך לבדוק גם איפה בדיוק בתוך הקובץ הזה, כי ה-id הוא ייחודי לעריכה דרך Mt Manager, ואנחנו עוברים כלי. אז אנחנו נגלה שזה נמצא במתודה h.

מעולה, עכשיו נעבור לעבוד במחשב - כי Smali2Java ב-Mt Manager היא אופציה למשתמשים בתשלום. נשתמש להמרה ב-JadxGUI.

נפתח את קובץ ה-APK, ננווט לנתיב הנכון ונחפש את h. והנה, נהדר, מצאנו קוד שנראה רלוונטי!



מה קורה כאן בעצם?

1. מוצגת הודעה מסוג Toast למשתמש עם הסטרינג הזה.
2. נשלחת שגיאה ב-Logcat עם הסטרינג הזה.
3. "האקדח המעשן" - פקודת Finish שמובילה לסגירת האפליקציה.

שימו לב לשם המחלקה - h - שלא מלמד שום דבר על תוכן ותפקיד המחלקה. זה בדרך כלל אופייני לקוד שעבר ערבול (obfuscation) שמיועד להקשות על פענוחו. כנראה של-Douqin, יצרנית המכשיר, היה משום מה מאוד חשוב לשמור על האפליקציה הזו שתעבוד רק כשאפליקציות אחרות קיימות במכשיר, למרות שהיא חינמית לחלוטין. עוד נקודה שחשוב לשים אליה לב היא, שבמחלקה כאן לא מתבצעת הבדיקה עצמה. רק מתבצעות פקודות מסויימות לאחר שהבדיקה כבר התבצעה. אז החלטתי לבטל את פקודת ה-Finish.

קוד ה-Smali הוא מסורבל וארוך, אז קיצרתי תהליכים ושאלתי את ChatGPT מה בדיוק למחוק ב-Smali (כמובן שניתן לעשות ידני, אבל אני חושב שאם יש כלים שמקצרים לנו את העבודה, אחרי שהבנו מה לעשות, כדאי להשתמש בהם) אז צריך להסיר את השורות האלו במתודה h:

```
.line 347
invoke-virtual {p0}, Lcom/duoqin/remote/activity/RemoteActivity;->finish()V
```

(את העריכה עצמה אני עושה ב-MT Manager).

אז מחקתי את השורה, שמרתי את השינויים והכנסתי את הקובץ הערוך למכשיר.

פה חיכתה לי הפתעה מתסכלת - האפליקציה אומנם לא נסגרה - אך הגיעה למסך לבן ולא עבדה. זה אומר, שבעצם, כמו שזיהינו המחלקה h היא רק מה שקורה אחרי שהאפליקציה זיהתה שמשהו לא תקין. ובעצם - ההשבתה האמיתית של האפליקציה מתרחשת בשלב מוקדם יותר. הדרך היא לחפש קריאות ל-h בקוד ה-Java, אבל כאן זה במקרה יותר פשוט. תחזרו לתמונה מ-Jadx למעלה? מיד אחרי h, מופיע המשתנה הבוליאני i, שמאותחל כ-true או false לפי בדיקות מסויימות (הכוללות בדיקה אם כמה PackageName קיימים או לא, שמניסיוני, אם הוא לא קיים האפליקציה אכן לא עובדת). ובכן, ככל הנראה הדרך היא שהמשתנה יחזיר תמיד True.

ChatGPT כדי לכתוב את ה-Smali כמו שצריך, ו... זה אכן עובד!

למעוניינים, קבצי ה-APK של האפליקציה, המקורית והערוכה, נמצאים בגיטהאב שלי, כאן.

עריכות נוספות (סקריפטי RC, hosts, build.prop)

את עיקר העבודה שעושים בדרך כלל בקושחות ערוכות כבר הסברתי, עכשיו אני אוסיף כאן על עוד כמה דברים. אתחיל מהדברים הפשוטים:

קובץ ה-hosts - נמצא ב-system/etc/. כמו במערכות הפעלה אחרות, אנחנו יכולים גם באנדרואיד להגדיר כאן שדומיינים מסוימים יפנו לכתובות ip מסויימות. השימוש הקלאסי כמובן הוא חסימת כתובות מסויימות, לרוב פרסומות. יש אפליקציות מאוד נוחות לזה, כולל תיעוד הכתובות שאפליקציות אחרות ניגשות אליהן. כלי מאוד נוח כזה הוא AdAway. אם תרצו לערוך ידנית את קובץ ה-hosts, הוא פשוט קובץ טקסט. לחסימת פרסומות וכדומה מרשימות מוכנות, אם אתם לא רוצים להשתמש באפליקציות, אני ממליץ על הפרויקט הזה בגיטהאב. יש לו רשימות גדולות של כתובות, מחולקות לפי קטגוריות (פרסומות. רשתות חברתיות. וכדומה), ומוכנות במבנה של קובץ hosts.

קובץ ה-build.prop - הסברנו עליו כבר למעלה. נמצא בכמה מיקומים במחיצות ה-system, משתנה בין מכשירים שונים. אך הראשי נמצא ב-system/build.prop. גם הוא ניתן לעריכה כקובץ טקסט פשוט. ניתן לערוך חלק מהערכים הקיימים כבר, וניתן גם להוסיף ערכים. למשל, הוספה פופלארית יחסית היא:

```
debug.sf.nobootanimation=1
```

שמבטל את אנימציית ההדלקה, והדלקת המכשיר הופכת למהירה יותר.

סקריפטי rc

אלו סקריפטים שרצים במערכת בהדלקת המכשיר, או כאשר מתרחשים אירועים מסוימים - למשל חיבור כבל usb. התחביר שלהם אינו תחביר sh רגיל, ולכן הדרך הרגילה לשימוש בהם היא כתיבת סקריפט rc מינימלי, שטוען סקריפט sh שכתוב כסקריפט sh סטנדרטי, ומריץ את הפקודות שבתוכו. נדגים כאן הרצה של סקריפט בעת הפעלת המכשיר.

שימו לב, שכל הסקריפטים צריכים להיות במבנה Unix ולא Windows. אני משתמש כדי לערוך ב-Notepad++ ואז בתפריט ה-Edit אני בוחר EOL Conversion ו-Unix. בנוסף, שימו לב שאני מביא כאן את קטעי הקוד, אך כדי לשמור על המבנה, אני מצרף אותם גם בקישור הזה.

הסקריפט הראשון שלנו יהיה סקריפט rc, שרץ אוטומטית בהפעלת המכשיר. בשביל זה, נשים אותו בתיקיית /system/etc/init - זה יגרום לכך שהוא יופעל אוטומטית בהדלקת המכשיר. נשמור אותו בשם bootscript.rc (לדוגמא, השם יכול להיות גם משהו אחר). ותחת הרשאות 755. הקוד שלו הוא:

```
service bootscript /system/etc/bootscript.sh
    oneshot
    seclabel u:r:magisk:s0

on property:sys.boot_completed=1
    start bootscript
```

כמו שאפשר לראות, הבלוק הראשון מגדיר את השירות - הרצת סקריפט ה-sh שלנו, מריץ אותו (דרך מג'יסק) בהרשאות SELinux מקילות מאוד, ככה שיש לו הרשאה דומה לרוט, והבלוק השני מפעיל אותו מיד בהדלקת המכשיר.

כעת, ניצור את סקריפט ה-sh. כמו שראינו, בחרתי להריץ את הסקריפט מ-/system/etc/bootscript.sh, אז שם נשים את הקובץ - וגם הוא, בהרשאות 755. זה הקוד שלו:

```
#!/system/bin/sh
set -e
BROWSER_FLAG=/data/adb/.browser_hidden
LOGFILE=/data/local/tmp/bootscript.log
exec >>"$LOGFILE" 2>&1
echo "---- browser hide check at $(date) ----"
echo "Waiting for browser app to become available..."
while true; do
  if pm list packages | grep -q com.android.browser; then
    echo "Browser app detected. Disabling and hiding..."
    pm disable com.android.browser
    pm hide com.android.browser
    echo "Browser disabled and hidden successfully at $(date)" >> "$BROWSER_FLAG"
    break
  fi
  sleep 1
done
```

בדוגמא הזו, שהיא די ברורה, אנחנו מגדירים קובץ FLAG שיעזור לנו לדעת אם הפעולה אכן בוצעה, קובץ log שישמור תיעוד של הפעולות, ומשביתים ומסתירים את הדפדפן מהמכשיר.

גיבוי השינויים וצריבה על מכשירים אחרים

בשלב הזה נשלים חובות מהשלבים הקודמים - איך לצרוב את המחיצות שלנו. אך לפני זה, נלמד איך לגבות את השינויים שעשינו במכשיר כדי לצרוב על מכשירים אחרים - כמובן, אך ורק מאותו הדגם בדיוק.

חשוב לציין גם, שבשלב זה אציג שיטות אוניברסליות ל(כמעט) כל המכשירים והיצרנים, אך הם לא בהכרח הכי מהירות או נוחות.

גיבוי השינויים:

כמובן, אפשר פשוט אחרי כל השינויים להריץ את ro2rw ולבחור לגבות את ה-super "המקורי", שבעצם מה שיש עכשיו על המכשיר זה כבר עם כל השינויים. אם אנחנו מעדיפים לעשות את זה ידנית, נשתמש בפקודת dd. (גם כן, דורש רוט).

לדוגמא:

```
dd if=/dev/block/by-name/super of=/sdcard/super.img
```

בסיום התהליך, ה-super ישמר בתיקיה הראשית של האחסון המקומי שלנו, בשם super.img. יתכן שבמכשיר שלכם הנתיב הראשון (למחיצת super על המכשיר) יהיה קצת שונה.

אני ממליץ לפני גיבוי הקושחה וצריבתה על מכשירים אחרים, להפעיל מחדש את המכשיר, לאפס אותו וכדומה, כדי לוודא שכל השינויים עובדים כנדרש והמכשיר פועל תקין ולא רק "מכוח האינרציה" של דברים שנשארו ב-data מהקושחה המקורית.

צריבה:

ביחד עם adb, יש כלי צריבה בשם fastboot. נחבר את המכשיר למחשב, נריץ adb reboot bootloader כדי להיכנס למצב הצריבה, נחכה שהמכשיר באמת יהיה במצב fastboot ואז נריץ את fastboot:

```
fastboot devices
```

אם לא מוצג מזהה של מכשיר מחובר - זו תקלת דרייברים או משהו דומה.

אחרי שהמכשיר מחובר, נצרוב עם הפקודה:

```
fastboot flash super c:\users\username\super.img
```

כשכמובן, הנתיב ל-super.img צריך להיות נתיב לאיפה שממוקם באמת ה-super.img שלכם.

יתכן שבמחיצות כמו vbmeta, boot וכדומה הוא יכתוב שאין מחיצה כזו - זה בגלל שצריך לפעמים לצרוב בהתאם לסלוט, במכשירי A/B. במקרה כזה נצרוב לסלוט הפעיל, למשל boot_a:

```
fastboot flash boot_a c:\users\username\magisk_patched-xxxx.img
```

סיכום

וואו! זה היה מסע ארוך במיוחד! המאמר הזה בעצם מסכם ידע שצברתי במשך זמן רב, עם הרבה ניסוי ותעייה, לא הכול בתחומים האלו ממש מתועד כמו שצריך... בטח שלא בעברית, אבל גם באנגלית לא ממש.

אין הרבה חומר בעברית באופן כללי על התחום הזה, אפילו בסיסי ולא מקיף. המקורות שאני מכיר הם פורום מתמחים טופ - פורום טק ישראלי עם קטגוריית אנדרואיד מרשימה, והבלוג שלי, שהתחלתי לאחרונה לכתוב בו על הנושא (אני חושב שאני לא כל כך ייחודי, ומן הסתם יש עוד בלוגים ואתרים שמסבירים, ואשמח להכיר אותם. אם אתם מכירים - תשלחו לי!) אם אתם גם כן מתעסקים בתחום - אשמח להכיר אתכם.

בשפות אחרות יש עוד כמה אתרים מומלצים, ושמתי קישורים אליהם בנספח הקישורים בסוף המאמר.

זה המאמר הראשון שאני מפרסם ב-DigitalWhisper, ואני שמח מאוד להיות חלק מהפרויקט המרשים הזה. כתיבת מאמר היא פרויקט דורש הרבה זמן ומאמץ, אבל בעיני, זוהי חוויה מעניינת ומשתלמת מאוד. מעבר לצד החווייתי שבדבר, כתיבת המאמר הכריחה אותי לסדר לעצמי דברים בראש ולהבין לעומק דברים שעד עכשיו למעשה הבנתי רק בצורה שטחית, וזה רווח גדול מאוד.

אפשר לפנות אלי או בדף צור הקשר באתר שלי, או במייל: aiv.firmwares@gmail.com

קישורים ומקורות מידע

- הפוסט בבלוג שלי על גלקסי A06:

https://aiv-dev.com/he-IL/2025/04/21/GalaxyA06/

- לקריאה נוספת על מחיצות באנדרואיד:

https://source.android.com/docs/core/architecture/partitions

- דוקומנטציית מודולי מג'יסק:

https://topjohnwu.github.io/Magisk/guides.html#magisk-modules

- KitsuneMagisk - ה-fork שמשמר את MagiskHide וננטש:

https://github.com/1q23lyc45/KitsuneMagisk

- התיעוד הרשמי של מודולי APEX:

https://source.android.com/docs/core/ota/apex

- הגיטהאב של APKEditor:

https://github.com/REAndroid/APKEditor

- פורום XDA Developers, פורום מקצועי (באנגלית) בנושאי מובייל בפרט וטק בכללי:

https://xdaforums.com/

- פורום 4PDA, פורום מקצועי (ברוסית) בנושאי מובייל בפרט וטק באופן כללי. אני יודע שרוסית אומר שרובנו לא יודעים לקרוא... אבל מומלץ מאוד. בשביל מה יש גוגל טרנסלייט? אגב, אם קיבלתם שגיאת 404 בהורדת קובץ משם - זה רק בגלל שאין לכם שם משתמש:

https://4pda.to/forum/index.php?act=idx

- Jtech forums - פורום חרדי אמריקאי (באנגלית), וכמו הקודמים, עוסק באופן כללי בטק, אך ממוקד בעיקר במובייל:

https://forums.jtechforums.org/