

---

# מבוא למחקר אפליקציות

מאת עידן שכטר

---

## הקדמה

מחקר בעולם הסייבר הינו עולם דינמי, מגוון ומרגש, המציב בפני חוקרים אתגרים המתחדשים בהתאם לעולם טכנולוגי הדוהר קדימה. אחד מעולמות התוכן הרלוונטיים ביותר בימינו, הוא עולם ה-Mobile, מעצם הפיכתם של מכשירים סלולריים לחלק בלתי נפרד מחיינו.

במאמר זה נחשף לעולם מחקר האפליקציות בסביבת Android, נכיר כלי מחקר פרקטיים ונכנס לראשו של חוקר בתהליך מציאת פתרון לבעיה. תוכן המאמר מהווה בסיס מעולה לכל המעוניין לצלול אל עולם מחקר האפליקציות בסביבת Android ולפרוח ממנו לתחומים נוספים...

חשוב לציין, כי המאמר רלוונטי לכל המתעניין בתחום! אך לצורך חוויה מיטבית, אמליץ על היכרות עם שפת Java ועקרונותיה הבסיסיים וכך גם לגבי JavaScript, כמו כן ידע בפרוטוקולי תקשורת ושליטה במערכות מבוססות Linux.

## הקמת סביבה

על מנת לעסוק באתגר, עלינו להקים סביבת מחקר שבין היתר תאפשר לנו להתקין ולהריץ את ה-APK הרלוונטי. לשם כך, נצטרך מכשיר המריץ Android עם הרשאות Root המאפשר USB Debugging, או לחילופין, Emulator כלשהו המספק את אותו מענה.

במאמר זה לא נעסוק בהשגת הרשאות Root על מכשיר Android, אך נוכל למצוא שלל מדריכים באינטרנט בהתאם למכשיר שבידינו. ע"מ להדליק את פיצ'ר ה-USB Debugging במכשיר נפעל ע"פ הצעדים הבאים:

1. Settings ← About Phone ← Software Information ונלחץ על Build number מס' פעמים, עד אשר נראה הודעה המציינת כי הדלקנו בהצלחה את ה-Developer mode במכשיר.

2. נחזיר אל ה-Settings של המכשיר, כעת נראה לשונית חדשה בשם "Developer options", נכנס אל הקטגוריה החדשה, נגלול עד ל-Debugging, ונדליק את אופציית ה-USB debugging.



או במידה ומכשיר שכזה אינו בהישג יד, ניתן להשתמש באחד מהפתרונות הבאים:

1. יצירת מכשיר וירטואלי מבוסס ARM ב-[Android Studio](#)
2. יצירת מכשיר וירטואלי באמצעות Genymotion, והתקנת ARM Translation (מאחר וה-Image ש-Genymotion מייצר מחכה ארכיטקטורת x86, בעוד ה-APK מיועד לארכיטקטורת ARM)
  - a. מורידים Genymotion [מהקישור](#), מתקינים, ויוצרים מכשיר וירטואלי עם Android 9.
  - b. מתקינים [ARM Translation](#) על המכשיר הוירטואלי (גוררים את ה-Zip הרלוונטי (for\_9.0...)) אל המסך של המכשיר הוירטואלי, או לסירוגין משתמשים ב-adb install.

תוכנה נוספת שנצטרך היא [Android Studio](#), בו נשתמש כדי לקרוא ולחקור את קוד ה-APK הרלוונטי.

## האתגר

כעת, כשיש לנו Setup מוכן, נוכל להתקין את ה-APK הרלוונטי על המכשיר שברשותינו, ובהזדמנות זו גם נכיר את הכלי הראשון שנפגוש במאמר זה, והוא: ADB.

### Android Debug Bridge (ADB)

מדובר בממשק המספק נתיב תקשורת אל מול מכשירי Android המאפשר ביצוע פעולות מגוונות, כגון דיבוג מרחוק, צפייה בלוגים, הנגשת Shell וכו'.

במאמר זה, נשתמש בכלי בעיקר כדי לבצע פעולות בסיסיות כמו הנחה ומשיכה של קבצים, אך אמליץ בחום לחקור את הכלי לעומק ולהכיר את היכולות השונות שלו!

ADB מגיע כחלק מ-Android SDK Platform Tools, אותו נוכל להוריד ולהתקין בהתאם למערכת ההפעלה שלנו מהקישור [כאן](#).

הפקודה הראשונה אותה נכיר, היא adb devices, אשר מציגה בפנינו רשימה של כל מכשירים ה-Android המחוברים למחשב שלנו (פיזיים ווירטואליים כאחד):

```
user@user:~$ adb devices
List of devices attached
RFCR10H6BDA    device
```

נוכל לראות כי ישנו מכשיר מחובר אחד בעל המזהה RFCR10H6BDA. מאחר ושינו רק מכשיר אחד מחובר, השימוש במזהה אינו רלוונטי, לעומת זאת, במקרה בו ישנם מספר מכשירים מחוברים, נוכל להשתמש בדגל -s על מנת לציין את ה-UUID של המכשיר עליו נרצה לבצע את הפעולה המבוקשת.

לאחר שראינו שהמכשיר שלנו מחובר, נרצה להתקין את ה-APK של Shazam Lite על ידי שימוש בפקודה  
:install

```
user@user:~$ adb install com.shazam.android.lite_1.1.0-170321-101040_minAPI9\(\arm64-v8a\)\(nodpi\)_apkmirror.com.apk  
Performing Streamed Install  
Success
```

במידה והיו מספר מכשירים מחוברים, היינו מריצים את הפקודה על דגל -s בצירוף ה-UUID של המכשיר  
הרלוונטי:

```
user@user:~$ adb -s RFCR10H6BDA install com.shazam.android.lite_1.1.0-170321-101040_minAPI9\(\arm64-v8a\)\(nodpi\)_apkmirror.com.apk  
Performing Streamed Install  
Success
```

במידה והאפליקציה הותקנה בהצלחה, נוכל לראות את האייקון שלה בדף הבית / רשימת היישומים של  
המכשיר.

פקודות adb שימושיות נוספות:

- קבלת ממשק Shell על המכשיר:

```
adb shell
```

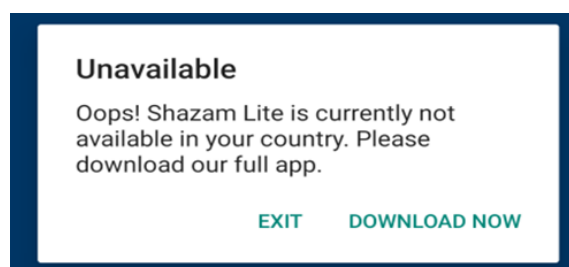
- "העלאת" קובץ למכשיר:

```
adb push <source> <destination>
```

- "משיכת" קובץ מהמכשיר:

```
adb pull <source> <destination>
```

כשנססה להריץ את האפליקציה, נתקל בשגיאה הבאה:



ובכן, נראה שהאפליקציה לא נתמכת במדינה שלנו - באסה. אבל מאחר ואנחנו חוקרים, לא ניתן לשגיאה  
הזו לעצור אותנו, המטרה שלנו - לזהות את המנגון הרלוונטי ולמצוא דרך לעקוף אותו!

## הבנת הבעיה

אנו מבינים כי קיים מנגנון מסויים באפליקציה, שמטרתו לזהות את המדינה בה נמצא המכשיר, ולאנוף את השימוש באפליקציה בהתאם, מכאן עולה השאלה - כיצד המנגנון עובד?

אפשרות אחת היא שה"אכיפה" מבוצעת בצד הלקוח, כלומר, האפליקציה מבצעת את הבדיקה בעצמה, ומציגה את ההודעה הרלוונטית בהתאם.

אפשרות נוספת, היא שהבדיקה מבוצעת בצד שרת, כלומר, האפליקציה אוספת נתונים מסויימים אודות המכשיר, שולחת אותם לשרת, והוא זה שמחליט איך צריך לפעול בהתאם לנתונים. לבסוף, השירות ישיב לאפליקציה עם ההחלטה שהתקבלה, וזאת תציג את ההודעה בהתאם.

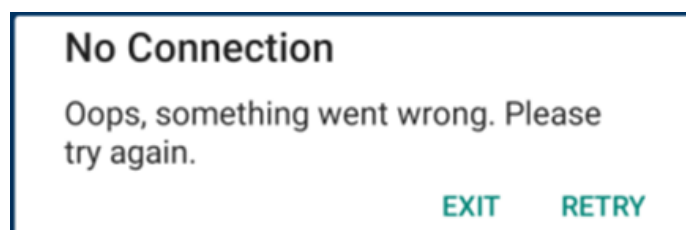
היתרון המשמעותי של האפשרות השנייה, בה השרת הוא זה שמבצע את הבדיקה, היא שאופן הפעולה של המנגנון אינו חשוף למשתמש, אלא רק המידע הנשלח \ מתקבל מהמנגנון, דבר המקשה על הבנת המנגנון, ולכן עשוי להקשות על מציאת פתרון.

נתבונן בפתרונות העומדים בפנינו:

- מנגנון ממומש בצד לקוח ← נזהה את המנגנון באפליקציה ← נשבש את אופן הפעולה שלו לטובתנו ← ניצחון
- מנגנון ממומש בצד שרת ← נזהה את המידע הנשלח \ מתקבל מהמנגנון ← ננסה להבין מה המידע ש"מפליל אותנו" (במידה ולא נצליח לזהות את המידע, נצטרך למצוא דרך להבין כיצד "מידע לגטימי" [כזה שעבורו המנגנון יחזיר תשובה חיובית] נראה, ולהשתמש בו, או לחילופין, לייצר אותו בעצמנו) ← נשנה את המידע המפליל הנשלח לשרת ונחליף אותו במידע לא מפליל ← ניצחון.

רגע לפני שננסה להבין מול איזו סיטואציה אנו נמצאים, נתעכב על נקודה חשובה - כחוקרים, נרצה לחקור כמה שפחות ונשאף למצוא פתרון הפשוט \ מהיר ביותר (אם הדבר אכן אפשרי). לכן, רגע לפני שנרוץ לפתרונות קצת יותר "מורכבים" כמו לרוורס את ה-APK (אל דאגה, גם לשם נגיע!) נחשוב על דרך פשוטה לענות על השאלה הבאה - האם המנגנון ממומש בצד שרת, או בצד לקוח?

פעולה פשוטה שיכולה לעזור לנו לענות על שאלה זו, היא לנסות לפתוח את האפליקציה כשאנחנו לא מחוברים לאינטרנט, מה שיכול להכניס אותנו ל-"Flow" שונה באפליקציה, כלומר, לגרום לאפליקציה להתנהג אחרת (לדוגמא, להציג הודעה שונה):





ואכן נגלה שנקבל שגיאה שונה. אבל רגע! האם זו הוכחה מספיק טובה? מה אם קיים באפליקציה מנגנון שקודם כל בודק אם אנחנו יכולים תחילה לגשת לשירות, ורק אז מבצע את הבדיקה הרלוונטית? (כלומר, הבדיקה תבוצע בצד הלקוח, אבל רק אחרי שווידא שהוא יכול לגשת לשירות) - זהו בהחלט תרחיש אפשרי!

על מנת לפתור את הסוגיה, נפעל בצורה הבאה - נבדוק האם המנגנון ממומש בצד שרת, ובמידה ונראה שלא, נוכל להסיק שהוא ממומש בצד הלקוח.

על מנת לבצע בדיקה זו, נצטרך למצוא דרך להאזין לבקשות שהאפליקציה שולחת לשירות, וכאן נכנס לתמונה HTTP Proxy.

## MITMProxy

פעמים רבות, כחלק מהליך מחקרי, נרצה לנתח את ההתנהגות הרשתית של אפליקציה מסויימת, בין היתר, כמו במקרה שלנו, על מנת להבין מנגנון העומד בדרכנו בצורה טובה יותר. את יכולת זו נשיג על ידי שימוש בשרת Proxy, זהו למעשה שרת ה"עומד" בין האפליקציה לבין השירות שלה, כך שהוא מקבל את ההודעות ישירות מהאפליקציה, ושולח אותן בשם האפליקציה לשרת, בדרך זו, נוכל לראות את הבקשות הנשלחות לשרת ואת התשובות להן (גם מעל HTTPS!) (כמובן, שגם פעולה זו לא בדיוק פשוטה, ותדרוש מאיתנו מספר מניפולציות נוספות).

למזלנו, לא נצטרך לכתוב שרת כזה בעצמנו ונוכל להשתמש בכלים קיימים, הכלי בו נשתמש הוא MITMProxy. MITMProxy הינו HTTP Proxy חינומי, מבוסס קוד פתוח, אינטראקטיבי, המציע ממשק (CLI & Web) קל (Lightweight) ופשוט לשימוש.

נוכל להשתמש בו כדי לבחון את הבקשות (HTTPS) הנשלחות מהאפליקציה לשירות, ולחפש בהן מידע שעשוי לעזור לנו בהבנת המנגנון (כמו כן, הרגישו חופשי להשתמש בפרוקסי המועדף עליכם).

([התקנת MITMProxy](#), הרצתו על ידי הפקודה `mitmproxy / mitmweb`, אמליץ להשתמש ב-`mitmweb` מאחר והממשק שלו חברותי יותר). על מנת לנתב את התעבורה שיוצאת מהמכשיר שלנו דרך MITMProxy, נצטרך לחבר את המכשיר איתו אנחנו עובדים לשרת שהקמנו, ולשם כך נצטרך שהמחשב איתו אנחנו עובדים והמכשיר בו אנחנו משתמשים ישבו על אותה רשת (במידה ומדובר באימולטור, זהו המצב הדיפולטי איתו אנחנו עובדים, ולכן כל מה שנצטרך לעשות זה להגדיר את כתובת ה-Proxy בהתאם ל-Host הרלוונטי) במידה ואנחנו עובדים ממכשיר פיזי, נוכל ליצור נקודת גישה אלחוטית במחשב איתו אנחנו עובדים, ולהתחבר אלייה ע"י המכשיר הרלוונטי.



הדבר האחרון שעלינו לעשות הוא [להתקין את ה-certificate של mitmproxy](#) על המכשיר איתנו אנחנו עובדים. לאחר שקינפנו והרצנו את MITMProxy בהצלחה, נריץ שנית את האפליקציה שנית.

רגע, מוזר! נראה שאנחנו שוב מקבלים שגיאת התחברות, כזו הזזה לשגיאה שקיבלנו כשאר פתחתנו את האפליקציה ללא חיבור לאינטרנט, אז מה בעצם קרה כאן?

## SSL Certificate Pinning

מנגנון אבטחה נפוץ בקרב אפליקציות מובייל, שבא לחזק את השימוש ב-SSL Certificates, הינו SSL Pinning. בפועל, המנגנון מוודא שה-Certificate שהאפליקציה מקבלת מהשירות איתו היא מנסה ליזום סשן מאובטח, הוא אכן ה-Certificate האותנטי של השירות, או במקרה שלנו, Certificate השייך לשירות של Shazam Lite, מאחר וחיברנו את האפליקציה לשרת Proxy, מנגנון ה-SSL Pinning המוטמע באפליקציה ימנע ממנה ליצור סשן מאובטח, מאחר וה-Certificate שתקבל הוא לא ה-Certificate של השירות. כלומר, "נועץ" את ה-Certificate המקורי של השירות (ומכאן השם).

אז איך נתגבר על מנגנון שכזה? בפועל, המנגנון מבצע בדיקה פשוטה - השוואה של ה-Certificate שקיבל מהשירות איתו הוא מנסה ליצור סשן מאובטח, אל ה-Certificate המקורי של השירות, המוטמע בקוד בדרך כלשי (לדוגמא, משווה ערכי Hash) פעולה זו צפויה לחזיר שתי תוצאות אפשריות - ה-Certificates זהים, או שלא, כלומר - True / False (כלומר, שחרור הנעץ - Unpinning).

כל מה שעלינו לעשות הוא לזהות את המימוש של המנגנון, ולגרום לו להחזיר את התשובה הרצויה עבור כל קלט שיקבל, אך איך נעשה זאת? במעמד זה נכיר כלי רב עוצמה, שבין היתר יפתור עבורנו את הבעיה בקלות, הלו הוא Frida!

## Frida

פרידה הינה כלי רב עוצמה המאפשר ניתוח ושינוי של יישומים בזמן ריצה (Dynamic Analysis) על ידי ביצוע Hook לפונקציות, (שינוי מימוש של פונקציה) קריאת וכתובה של ערכים בזיכרון ועוד. מדובר בכלי מחקרי לעולמת ה-iOS ו-Android כאחד (כולל תמיכה בספריות Native) אף על פי שראוי לכתוב על Frida מאמר שלם בפני עצמו, לא נצלול לעומק יכולות הכלי, אלא נשאר בגבול הגזרה שרלוונטי תרגיל זה.

בפועל, Frida מורכבת מ-2 חלקים, האחד הוא שרת הנמצא על המכשיר (קובץ בינארי) אשר מבצע מניפולציות בתהליך הרלוונטי. והשני, הוא הקליינט המנגיש ממשק סקריפטינג עשיר התומך במספר שפות, וכמו כן כלים נוספים (frida-ps, frida-trace, etc) (בפועל, הקליינט והכלים שצויינו הם ישויות נפרדות, אך הם חלק מחבילת ה-frida-tools). הקליינט והסרבר מבצעים תקשורת דו צדדית על מנת לבצע פעולות שונות, בפועל, הקליינט ישלח לשרת את הפעולה שנרצה לבצע (לדוגמא, סקריפט שנרצה להריץ), השרת יריץ אותו, ויחזיר לקליינט את הפלט הרלוונטי.

**הערה:** השם Frida מגיע מצירוף המילים Free IDA - יוצר הכלי, Ole André Vadla Ravnås, שאף ליצור אלטרנטיבה חינומית ל-IDA, התשלב יכולות ניתוח סטאטיות ודינאמיות כאחד, אך עם הימים הפוקוס השתנה לניתוח דינאמי בלבד. כיום, הכלי מתוחזק על ידי חברת NowSecure / אמליץ לקרוא על הכלי באתר הבית: <https://frida.re/docs/home>.

כדי להתקין את Frida נריץ את הפקודה:

```
pip install frida-tools
```

נוודא שההתקנה אכן הצליחה על ידי:

```
frida --version
```

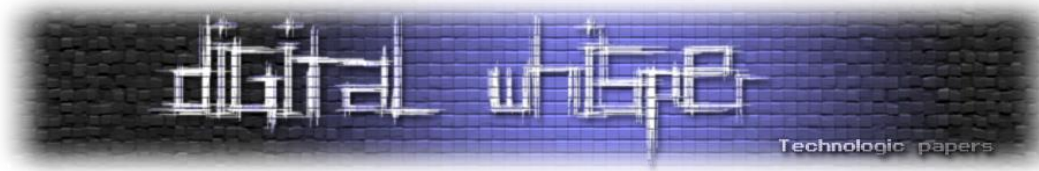
בימים של כתיבת מאמר זה, גרסאת ה-Frida העדכנית ביותר הינה 16.0.7, והיא זו שתשמש אותי בפתרון האתגר, אך השימוש בכלי רלוונטי גם לגרסאות ישנות יותר, וכמו כן לגרסאות עתידיות!

כעת, נתקין את החלק השני, הסרבר. נכנס לדף ה-releases ב-repository של Frida ב-Github, ונחפש את קובץ השרת:

```
frida-server-{version}-android-{arch}.xz
```

כך ש-arch מייצג את הארכיטקטורה של המכשיר בו אנחנו משתמשים (אם מדובר במכשיר פיזי, נבחר ב-arm64, אך אם מדובר באימולטור, נבחר ב-x86). את version נתאים לגרסאת ה-client שהתקנו, זו שנקרא בהרצת הפקודה:

```
frida --version
```



**כמה מילים על תאימות:** נוכל להשתמש בצמד שרת + לקוח, כל עוד הם חולקים את אותה גרסת major, כלומר, שרת בגרסא 16.x.x יעבוד אל מול קליינט מגרסא 16.x.x

כפי שהבנו, Frida מאפשרת לנו בין היתר לבצע Hook לפונקציות, וזה בדיוק מה שאנחנו צריכים! במידה ונוכל לזהות את הפונקציה שמבצעת את בדיקת ה-Certificate, נוכל לשנות את ערך החזרה שלה בהתאם!

מאחר ומדובר בפעולה דיי נפוצה הממומשת על ידי ספריות מוכרות, לא נצטרך למצוא את הפונקציה בעצמנו, ונוכל להשתמש בסקריפטים קיימים (אל דאגה, נבצע Hook באמצעות Frida בהמשך)

סקריפט המבצע SSL Unpinning לאפליקציות Android נוכל למצוא ב-Codeshare של Frida (שבין היתר מכיל מגוון סקריפטים נוספים):

<https://codeshare.frida.re>

על מנת להריץ את הסקריפט הלוונטי, נשתמש בפקודה הבאה:

```
frida -U -f com.shazam.android.lite -l <script_name>
```

Frida תריץ את האפליקציה עבורנו תוך כדי ביצוע Hook לפונקציות המממשות מנגנוני SSL Pinning בספריות רלוונטיות מוכרות, נוכל לזהות בהדפסות הסקריפט שזוהי שימוש ב-TrustManager והותאם מעקף בהתאם (מאחורי הקלעים, Frida תבצע Hook לפונקציות הרלוונטיות).

אם נבחן את הסקריפט, נראה כי ה-Hook מבוצע על ה-Constructor של המחלקה "SSLContext" המשמשת את האפליקציה ליצירת חיבור מאובטח (TLS). אותו Constructor מקבל כפרמטר אובייקט מסוג TrustManager המכיל את ה-Certificate של השירות (אותו אנחנו "נועצים"), שבו יוכל להשתמש כדי לבצע את הבדיקה הרלוונטית בהמשך הריצה. על מנת "להתחמק" מתרחיש זה, נספק ל-Constructor אובייקט TrustManager ריק, לחילופין, נוכל להחליף את אובייקט ה-TrustManager בערך null לקבל תוצאה זהה.

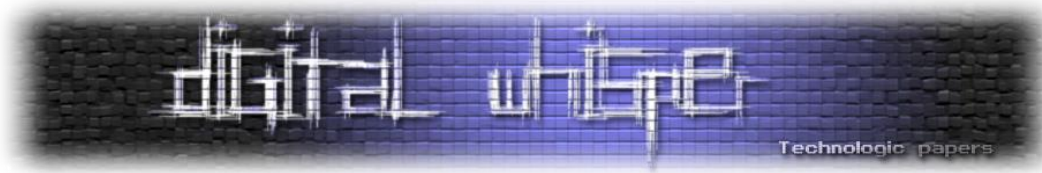
הופה! אנחנו מקבלים את השגיאה הראשונה שוב! פתרנו את הבעיית ה-SSL Pinning. אם נביט ב-mitmproxy, נזהה בקשת HTTPS הנשלחת לשרתי האפליקציה, ומכילה JSON מעניין מאד ב-Body:

```
POST https://amp.shazam.com/configuration/v1/configure
...
```

נשים לב כי ה-Status code של התשובה שאנחנו מקבלים מהשרת הוא 403, כלומר, Forbidden, מה שמהווה רמז ענק לעובדה שהבקשה בה אנו דנים, היא הבקשה ש"מפלילה אותנו".

השגיאה שאנו מקבלים, זו שאומרת כי האפליקציה לא זמינה במדינה הנוכחית שלנו, היא רמז ענק לחלק המפליל בבקשה, נוכל לראות כי נשלחים 3 פרמטרים מעניינים, שקשורים באופן ישיר למדינה בה אנחנו נמצאים: Country, mobileCountryCode, ו-mobileNetworkCode.





משהו אומר לי שהפתרון שלנו קשור סביבם.

לדוגמא, ניתן לראות כי השדה "country" מחזיק את הערך "il" כלומר Israel, בנוסף השדה "[mobileCountryCode](#)" מחזיק את הערך 425, חיפוש מהיר בגוגל ילמד אותנו שמדובר במספר המזהה מדינה בעולם תקשורת המובייל, ובהתאם למקרה שלנו, מדובר במספר המזהה של ישראל.

אם נצא מנקודת הנחה שערכים אלה "מפלילים" אותנו, כל שנצטרך לעשות הוא להבין עבור אילו ערכים הבדיקה עוברת בהצלחה, נשלח אותם לשרת, וניצחנו!

חיפוש מהיר בגוגל מלמד אותנו ש-Shazam Lite נתמכת במספר מצומצם של מדינות, בין היתר בהודו. נחפש בזריזות את ה-country code של הודו, וכמו כן את ערך ה-mobileCountryCode. הם "in" ו-"404" בהתאמה (נוכל להשתמש ב-404, 405, 406 עבור mobileCountryCode).

נשתמש בפיצ'ר שימושי ב-mitmproxy (וכמו כן ב-proxies מוכרים נוספים) המאפשר לנו לערוך בקשות, ולשלוח אותן לשרת בשנית, באמצעות פיצ'ר זה נוכל לערוך את השדות החשודים ולראות את תגובת השרת.

כיצד נערוך את הבקשה? על ידי סימן העיפרון שנמצא בצד ימין של הבקשה: נשנה את ערכי "country" ו-"mobileCountryCode" בהתאם, ונשלח את הבקשה בשנית ע"י לחיצה על כפות ה"Replay" בצד שמאל למעלה, או לחליפין על ידי לחיצה על R. וואלה! אנחנו מקבלים 200 מהשרת! **ניצחנו!**

אבל רגע, אומנם הבנו מה הערכים שמפלילים אותנו והצלחנו לקבל תשובה חיובית על הבקשה שלנו, אך מדובר בבקשה לה ביצענו Replay ולא בבקשה עצמה הנשלחת ע"י האפליקציה. בשלב זה נחזור אל כלי עוצמתי שהכרנו לפני מספר דקות, הלוא היא Frida.

אחד הדברים המגניבים ש-Frida מאפשרת לנו לעשות, הוא היכולת לבצע Hook לפונקציות בפשטות ובמהירות, על ידי כתיבת סקריפטים פשוטים ב-Javascript (בדומה ל-Script ה-SSL Unpinning שבו השתמשנו). ביצוע Hook לפונקציה למעשה מאפשר לנו לשנות את המימוש של לחלוטין, לדוגמא במקרה שלנו,

פונקציה היוצרת JSON עם ערכים מסויימים - נוכל לשנות את הדרך בה הפונקציה יוצרת את ה-JSON הרלוונטי על ידי הכנסת ערכים לבחירתנו.

כאן אפשר למצוא מדריך מעולה לשימוש ב-Frida בעולמות ה-Android:  
<https://node-security.com/posts/android-hooking-in-frida>



בפועל, על מנת ליצור Hook לפונקציה, כל מה שנצטרך הוא לדעת מה שם ה-Class שמממש אותה, מה השם שלה בתוך אותו Class, ומה החתימה שלה (הפרמטרים וערך החזרה). אז איך נוכל למצוא את הפונקציה הזו? איך בכלל נוכל לקרוא את הקוד של האפליקציה? לצורך כך נכיר כלי נוסף, והוא jadx.

## Jadx

Jadx הינו כלי חינמי מבוסס קוד פתוח המסוגל לבצע די-קומפילציה לקבצי APK, ואף מספק ממשק UI לצפייה בקוד שעבר די-קומפילציה. במקרה שלנו, נשתמש ב-Jadx ע"מ לחלץ את קבצי ה-Java המרכיבים את האפליקציה, ונטען אותם ב-Android Studio על מנת לבחון אותם ולאתר את הפונקציה הרלוונטית.

את Jadx נוכל להוריד מה-Repository הרשמי ב-Github ולהתקין אותו ע"פ ההסבר שנמצא ב-Repository:  
<https://github.com/skylot/jadx>

לאחר שהתקנו jadx בהצלחה, נבצע די-קומפילציה ל-APK של האפליקציה כך:

```
jadx -e -deobf <path_to_apk>
```

כמה מילים על הדגלים בהם נשתמש:

“-e” - מייצא את הקבצים בצורה שבה כל Class נמצא בקובץ משלו, דבר המדמה בצורה טובה יותר את קוד המקור (מבחינת קבצים), ומאפשר לנו לבצע Import אל תוך Android Studio בתור פרוייקט.

“-deobf” - נבקש מ-Jadx לנסות לשחזר שמות של מחלקות ופונקציות שעברו אובפוסקציה (בפועל, jadx ייתן לפונקציות ומחלקות שמות קצת יותר "קריאים", דבר שעשוי להקל עלינו בקריאת הקוד)

נטען את הקוד ה-Decompiled ע"י File >> Open, כעת אנחנו ערוכים ומוכנים לריוורס האפליקציה.

## Static Analysis

כאשר אנו ניגשים למחקר סטטי של אפליקציה, בדגש על מצב בו אנו מחפשים מרכיב נקודתי וספציפי, או פונקציה במקרה שלנו, עולה השאלה - איך נמצא את הפונקציה? הרי מדובר בכל כך הרבה קוד! כדי שנוכל לדייק את עצמנו, או לפחות לצמצם את "משטח" החיפוש, נחשוב על Point Entry, כלומר, נקודה בקוד שיהיה לנו דיי קל למצוא, שקשורה באופן כזה או אחר לפונקציה שאנחנו רוצים למצוא.

לדוגמה במקרה שלנו, האפליקציה שולחת בקשה לשרת, בקשה זו מכילה ב-Body שלה JSON עם המידע אותו נרצה לשנות, כנראה שיש פונקציה שאחראית על בניית ה-JSON הזה, אולי נוכל לאתר אותה על ידי חיפוש מחרוזות שמופיעות באותו JSON, כמו לדוגמה "mobileCountryCode" (דוגמה נוספת היא לחפש את קטע הקוד שמרכיב את הבקשה הרלוונטית על ידי חיפוש ה-URL או אחד מה-Headers).

**נקודה חשובה לגבי מחלקות פונקציות ומשתנים שעברו אובפוסקציה:** בפועל, אחרי ש-jadx מבצע decompilation ל-APK, הוא ייתן שמות פשוטים וחסרי משמעות למחלקות פונקציות ומשתנים, בדרך כלל מדובר באותיות בודדות. כאשר נשתמש בדגל ה-debof, השמות יהפכו למעט יותר קריאים, שילוב של ספרות ואותיות, אך מאחר ושמות אלה עשויים להשתנות בין תוצאות דיקומפילציה שונות, נתבסס על השמות המקוריים, אותם ניתן לזהות ע"י שורת ההערה הנמצאת מעל לכל משתנה \ פונקציה \ מחלקה.

נלחץ ctrl+shift+f על מנת לפתוח את דיאלוג החיפוש, ונחפש את הסטרינג הרלוונטי. מעולה! נראה שיש רק תוצאה אחת! לחיצה כפולה על התוצאה תביא אותנו ל-Class בו ה-String הרלוונטי מופיע. נראה כי אותו String מזהה Attribute של מחלקה בשם:

com.shazam.android.lite.e.a.b.a.a.c0686c

```
/* renamed from: a */  
public final String f1654a;
```

בנוסף לאותו שדה, נוכל לראות שדות נוספים עם השמות "model", "mobileNetworkCode", "os" נראה קצת מוכר לא? מדובר בשדות שמופיעים ב-JSON שאנחנו מחפשים!

עושה רושם שמחלקה זו מייצרת אובייקט המכיל את הערכים שמעניינים אותנו, אם נמצא את החלק בקוד שמייצר את האובייקט הרלוונטי, ומזין אותו בערך "mobileCountryCode", נוכל למצוא את מקור הערך הרלוונטי!

```
public C0685b(String str, String str2, String str3, String str4, C0684a c0684a, C0686c c0686c) {  
    this.f1654a = str;  
    this.f1655b = str2;  
    this.f1656c = str3;  
    this.f1657d = str4;  
    this.f1658e = c0684a;  
    this.f1659f = c0686c;  
}
```

```

package com.shazam.android.lite.p054e.p055a.p057b.p058a.p059a;

import com.p017b.p018a.p019a.InterfaceC0263c;
/* renamed from: com.shazam.android.lite.e.a.b.a.a.b */
/* loaded from: classes.dex */
2 usages
public final class C0685b {
    1 usage
    @InterfaceC0263c(m566a = "inid")

    /* renamed from: a */
    public final String f1654a;
    1 usage
    @InterfaceC0263c(m566a = "platform")

    /* renamed from: b */
    public final String f1655b;
    1 usage
    @InterfaceC0263c(m566a = "language")

    /* renamed from: c */
    public final String f1656c;
    1 usage
    @InterfaceC0263c(m566a = "country")

    /* renamed from: d */
    public final String f1657d;
    1 usage
    @InterfaceC0263c(m566a = "application")

    /* renamed from: e */
    public final C0684a f1658e;
    1 usage
    @InterfaceC0263c(m566a = "device")

```

נשים לב כי Attribute זה מוגדר על ידי ה-Constructor, וליתר דיוק, מדובר בפרמטר השני. נלחץ ctrl+left click על ה-Constructor של המחלקה, ואוטמטית נקפוץ לשימוש היחיד שיש בו בקוד (במידה והיה נעשה בו שימוש יותר מפעם אחת, היינו מקבלים את רשימת כל המופעים).

נשים לב כי נגיע לשורת ד"י ארוכה, המכילה שרשור של מספר פונקציות ומשתנים בעלי שם לא אינפורמטי (גם כאן מדובר באובפוסקפציה), אך לנו זה לא משנה, נוכל "להתעלם מהרעש" ולהתרכז באובייקט שמעניין אותנו - C0686c:

```

(), new C0686c(this.f14881.mo176e(), this.f14881.mo174g(), this.f14881.mo173h(), new C0687d(str: "Android", String.valueOf(this.f14881.mo175f()))));

```

כפי שראינו, הערך שיקבל ה-Attribute "mobileCountryCode" הוא השני ב-Constructor, אותו ערך המגיע מפונקציה בשם "mo174g" מאובייקט מסוג "com.shazam.android.lite.e.a.l InterfaceC0720l /". אם ניכנס לקובץ ה-Class של אותו אובייקט, נראה כי הקובץ מכיר חתימות לפונקציות ללא המימוש שלהן וזאת מאחר ומדובר בממשק שיש לממש.



אז איך נמצא את ה-Class שמממש את הממשק הרלוונטי? נחזור לדיאלוג החיפוש (ctrl+shift+f) ונחפש את הטקסט הבא:

“implements InterfaceC0720I”

כך נוכל למצוא את כל המחלקות המממשות את אותו ממשק, ולשמחתנו, יש רק אחת כזו! “C067b \ com.shazam.android.lite.d.b” נוכל לראות ב-Constructor של המחלקה שימוש ב-API מעניין מאד, נראה שאנחנו קרובים לפתרון!

```
1 usage
public C0607b(Application application) {
    TelephonyManager telephonyManager = (TelephonyManager) application.getSystemService( name: "phone");
    if (telephonyManager.getSimState() == 5) {
        this.f1525b = telephonyManager.getSimOperator();
        this.f1527d = telephonyManager.getSimCountryIso();
    } else {
        this.f1525b = null;
        this.f1527d = null;
    }
    this.f1526c = Locale.getDefault();
}
```

נריך חיפוש זריז על TelephonyManager ונמצא את הדף האינפורמטיבי [הבא](#).

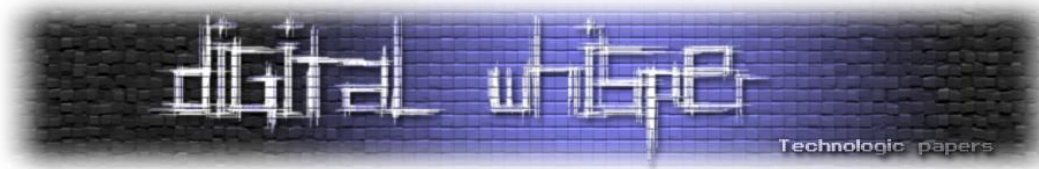
נקרא על הפונקציות "getSimOperator" ו"getSimCountryIso" ונראה שהן מחזירות בדיוק את מה שאנחנו מחפשים. נוכל לראות כי הפונקציה "b" מחזירה את התוצאה של "getSimCountryIso", כלומר את הערך "country" ב-JSON הרלוונטי, והפונקציה "g" מחזירה את הערך "mobileCountryCode" שמגיע כחלק מהקריאה ל-getSimOperator.

נבצע בדיקה ונכתוב סקריפט Frida פשוט שמבצע Hook ל-2 הפונקציות הרלוונטיות, ומדפיס את ערך החזרה שלהן:

```
Java.perform(function()
{
    console.log("Script loaded")
    var theClass = Java.use("com.shazam.android.lite.d.b")

    theClass.b.implementation = function()
    {
        console.log("b ret - " + this.b())
        return this.b()
    }

    theClass.g.implementation = function()
    {
        console.log("g ret - " + this.g())
        return this.g()
    }
})
```



אז מה הסקריפט עושה? (האובייקט הנקרא Java בו אנו משתמשים כדי לקרוא לפונקציות כמו perform או use הוא למעשה reference לסביבת הריצה ה-Java-ית של האפליקציה הרלוונטית), תוכן הסקריפט שלנו נמצא בתוך פונקציית perform, שהיא חלק חשוב מה-Javascript API של Frida, פונקציה זו למעשה יוצרת את Context הריצה שלנו, בפועל היא יוצרת Thread בתהליך האפליקציה הרלוונטית, המיועד להרצת הסקריפט שיצרנו.

נתחיל בהדפסה פשוטה שתודיע לנו כי הסקריפט נטען בהצלחה, ונתחיל ביצירת משתנה שייצג את המחלקה הרלוונטית על ידי שימוש בפונקציה use, המאפשרת לנו "לגשת" ליישיות בעולם ה-Java, וכמו במקרה שלנו, יצירת Reference למחלקה כרצוננו.

כעת, האובייקט theClass הוא למעשה Reference למחלקה הרלוונטית, ונוכל להשתמש בו כדי לשנות מימוש של פונקציות השייכות ל-Class.

פעולה זו מיושמת על ידי "פנייה" לפונקציה הרלוונטית (נתייחס לאובייקט theClass כאל אובייקט רגיל לחלוטין, המכיל Attributes שמייצגים פונקציות ומשתנים) במקרה שלנו, b ו-g (גם אליהן נתייחס כאובייקטים) כך שנדרוס את ערך ה-Implementation שלהן (Attribute המייצג את המימוש של הפונקציה) ע"י מימוש פונקציית JavaScript אנונימית אשר תהווה את המימוש החדש.

נדפיס את ערך החזרה של המימוש המקורי של הפונקציה ע"י קריאה לפונקציה הרלוונטית מתוך האובייקט this, אשר כמו ב-Java, מהווה רפרנס לאובייקט הנוכחי, ולבסוף, נחזיר את ערך החזרה של המימוש המקורי של הפונקציה (כמובן שיוכלנו להכניס את ערך החזרה למשתנה ולוותר על 2 קריאות, אך המימוש הרלוונטי מאפשר לנו לגעת שוב במשמעות של האובייקט this ב-Context של Frida)

נשמור את הסקריפט ונריץ אותו (כפי שראינו במהלך המאמר), ונראה את ה-Output הבא:

```
zam.android.lite
┌───┐
│ ( ) │ Frida 16.0.7 - A world-class dynamic instrumentation toolkit
│ > │
│ /_/_/ │
│ . . . │
│ . . . │
│ . . . │
│ . . . │ More info at https://frida.re/docs/home/
│ . . . │
│ . . . │ Connected to SM G980F (id=RF8N21R4EBP)
Spawning `com.shazam.android.lite`. Resuming main thread!
[SM G980F::com.shazam.android.lite ]-> Script loaded
g ret - 425
b ret - il
```



מעולה! נראה שהפונקציות שמצאנו מחזירות בדיוק את הערכים שחיפשנו! כל מה שנשאר לנו לעשות הוא לשנות את המימוש של ה-Hook שכתבנו, כך שבמקום להדפיס את ערך החזרה של המימוש המקורי, נחזיר את הערכים שמצאנו.

נערוך את הסקריפט בהתאם:

```
Java.perform(function()  
{  
  console.log("Script loaded")  
  var theClass = Java.use("com.shazam.android.lite.d.b")  
  
  theClass.b.implementation = function()  
  {  
    console.log("Return fake value for b")  
    return "in"  
  }  
  
  theClass.g.implementation = function()  
  {  
  
    console.log("Return fake value for g")  
    return "404"  
  }  
})
```

צ'אקה! עקפנו את המנגנון וכעת נוכל להשתמש ב-Shazam Lite על אף ההגבלות ©

## סיכום

במאמר זה עקפנו מנגנון באפליקציית Android מוכרת תוך שימוש במספר כלים בעלי עוצמה רבה, התחלנו מלהתגבר על מנגנונים המקשים עלינו בביצוע מחקר עד לכדי מחקר סטאטי של קוד האפליקציה וזיהוי של הפונקציה הנקודתית אשר מהווה מקור שאלת המחקר שלנו.

ישנם נושאים רבים ורלוונטים בהם לא נגענו במאמר זה, אך הכלים שהכרנו מהווים Toolbox משמעותי ולגמרי מספק כדי להמשיך ולצלול אל עולם המחקר בסביבת Android.

מקווה שנהנתם!

## על המחבר

עידן שכטר, בן 25, אוהב בעלי חיים ואת הים, חוקר בקבוצת NSO.