האוניברסיטה הפתוחה
המחלקה למתמטיקה ולמדעי המחשב
החטיבה למדעי המחשב

פרויקט מתקדם במדעי המחשב

מימוש שיטות לזיהוי נוזקות באמצעות אלגוריתמי למידת מכונה על בסיס מאפיינים סטטיים

מנחה: פרופ׳ אהוד גודס

סמסטר א׳ 2022

מגיש:
רועי טלבי, ת.ז. 316411214

גרסא 1.0

הצעה לפרויקט זה הוגשה כחלק מהדרישות לקבלת תואר "מוסמך (M.Sc.) במדעי המחשב"
באוניברסיטה הפתוחה
יוני 2022

תוכן העניינים

1. <u>מבוא</u>

1.1 רקע- עולם אבטחת המידע

כיום, אנו חיים בעולם דיגיטלי, רוב רובו של המידע של ארגונים ואנשים פרטיים מאוחסן באמצעים דיגיטליים בצורה מקומית או רשתית (בענן).
תחום אבטחת המידע צבר תאוצה במרוצת השנים. מידי יום אנחנו נחשפים לתקיפות חדשות שנעשות כלפי ארגונים שונים לרבות חברות הייטק, אוניברסיטאות, ארגונים צבאיים וכמו כן כלפי אנשים פרטיים.

מטרת התוקף היא להשיג דריסת רגל בתוך הרשת ע״י ניצול של חולשות או פגיעויות שונות שנמצאו ברשת. או ע״י שיטות הנדסה חברתית ובכך להצליח להגיע לייעדו.
תוקף יכול לפעול ממניעים שונים, בניהם:

- <u>מניע כלכלי</u> - מידע שווה כסף. ונגישות למידע רגיש יהווה ערך רב עבור ארגונים שונים. כמו כן, יכולים לאיים בהסבת נזק למידע ולדרוש בעבורו תשלום כופר.
- <u>מניע אידיאולוגי-</u> תוקפים שפועלים בשם האידיאולוגיה על מנת לגרום ולהסב נזק למדינה/משטר אליו הם מתנגדים.

<u>שלושת היעדים העיקריים של עולם אבטחת המידע הם:</u>

- **שלמות** *(integrity)*
  הגנה מפני שינוי זדוני של המידע או השמדתו, כולל הבטחת אי התכחשות ואימות זהויות בעלי המידע.
- **סודיות** *(confidentiality)*
  הגבלת גישה או חשיפה, כולל הגנה על פרטיות וזכויות קנייניות.
- **זמינות** *(availability)*
  שמירה על זמינות ויעילות הגישה אל המידע בכל זמן נתון.

מערכות מידע עומדות בפני סיכונים יומיומיים המאיימים על שלמותן וביטחונן.
ההגנה עליהן כוללת אבטחה של מערכות החומרה והתוכנה ,אבטחת רכיבי התקשורת ואבטחת המידע הנאגר בהן.
הסיכון למידע גדל עבור מאגרי מידע רבים שמאוחסנים על גבי מחשבים בעלי גישה לאינטרנט .

1.2 זיהוי נוזקות ואתגרים

במשך שנים רבות מערכות לזיהוי נוזקות התבססו על 2 שיטות מסורתיות לגילוי חדירות (*Intrusion Detection*) וזיהוי נוזקות (*Malware Detection*) :

- <u>אימות מבוסס חתימה :</u>

טכניקה המבוססת על הגדרות מראש של סט חתימות עבור מתקפות ידועות. ע״י חיפוש דפוסים ספציפיים מערכת מבוססת חתימה תשווה רצפי חבילות שהתקבלו, רצפי פקודות, קבצים מוכרים לאלו של המתקפות המוכרות.

היתרון המרכזי בזיהוי מבוסס חתימה הוא ששיטה זו מאד אמינה, בעלת שיעור *false positive* נמוך מאד אך החיסרון המרכזי הוא שנדרש להחזיק חתימה עבור כל סוג מתקפה/נוזקה שיכולה להופיע ברשת. וכן נדרש לבצע עדכון חתימות בפרקי זמן יחסית תכופים וכן אין יכולת להתמודד עם מתקפות חדשות ופגיעויות מסוג *'zero day'* אשר טרם נחשפו ואופיינו ע״י מערכות *IDS* שונות בעולם.

- <u>אימות מבוסס אנומליה</u>

זיהוי נוזקות על בסיס אנומליה דורש הגדרת פרופיל בסיס שמייצג התנהגות שגרתית ונורמטיבית של המערכת, הפרופיל יכלול למשל נתוני רשת, נתוני אפליקציות וכו׳. לאחר מכן, כל פעילות שתחרוג ותסטה מהמוגדר בפרופיל תהווה מתקפה אופציונאלית.

היתרון המרכזי של אימות באמצעות אנומליה הוא היכולת לזהות מתקפות חדשות, כמו כן פגיעויות מסוג *'zero day'* ואילו מערכות זיהוי חדירות העושות שימוש בשיטה זו מאופיינות לרוב באחוז גבוה של *false positives*. היות והשיטה לא מבוססת על חתימה ידועה אלא על אירועים שחורגים מהנורמה ואירועים כאלו מתרחשים לעיתים גם במצבי שגרה ולא בזמן מתקפה.

תופעת לוואי שנובעת מכך, היא שאננות שנוצרת בשל אחוז גבוה של *false alarms* ולכן קיימת האפשרות שמתקפה אמיתית תעבור מבלי שיבחינו בה.

כפי שצוין לעיל, לשיטות המסורתיות חסרונות רבים ועל כן בשנים האחרונות התקיים מחקר משמעותי בתחום זיהוי הנוזקות, במטרה לייצר יכולת לסווג קובץ שטרם זוהה ואופיין בעבר כ-נוזקה או קובץ תמים באחוזי דיוק מספיק גבוהים.

1.3 מטרת הפרויקט

מטרת פרויקט זה לסקור ולממש שלוש שיטות שונות לזיהוי נוזקות באמצעות אלגוריתמי למידת מכונה בהתבססות על מאפיינים סטטיים של הקובץ הבינארי, כלומר מבלי להריץ את הקובץ בסביבת סימולציה מתאימה.

שתי השיטות הראשונות מגדירות אופן חילוץ שונה של מאפיינים סטטיים מקבצי הרצה (קבצי PE) המיועדים למחשבים אישיים (PC) המריצים מערכת הפעלה מסוג Windows.

גם בימים אלו מערכת ההפעלה Windows היא הנפוצה ביותר בשימוש עבור מחשבים אישיים, ועל כן היוותה ומהווה יעד למתקפות ע״י גורמים זדוניים ברחבי העולם.
ולכן בשנים האחרונות התגלו ואופיינו אלפים רבים של קבצי נוזקה עבור מערכת הפעלה זו וישנם Data Sets רבים של קבצי הרצה הידועים כזדוניים וקבצי הרצה הידועים כתמימים.
בפרויקט זה ייעשה שימוש ב-2 Datasets כאלו. וב-Dataset של קבצי הרצה תמימים.

השליטה השלישית מתמקדת בזיהוי נוזקות למערכת ההפעלה Android.
בשנים האחרונות המכשירים הניידים הפכו לחלק בלתי נפרד מחיינו ועל כן חלה עלייה משמעותית במספר המתקפות והנוזקות אשר נועדו למכשירים ניידים המריצים את מערכת ההפעלה Android.
בפרויקט זה ייעשה שימוש ב2 Data Sets שונים של נוזקות כאלו וב-Dataset של אפליקציות הידועות כתמימות ולא מזיקות.

במסגרת הפרויקט נבנה Dataset טבלאי מתויג לכל אחת מן השיטות שייסקרו בקצרה בסיכום הפרויקט ובהרחבה בעבודה המסכמת. כמו כן, נעשה שימוש במסווגים (Classifiers) מוכרים ונפוצים מתחום למידת המכונה עבור המאפיינים שחולצו.
בוצעה הערכת איכות לDatasets הסופיים שייבנו וכן נבנה ממשק משתמש אשר יאפשר למשתמש להכניס קובץ כקלט ועבורו יבוצע ניתוח וסיווג.

2. <u>תיאור הפרויקט</u>

2.1 תיאור הפרויקט

במסגרת הפרויקט נסקרו שתי שיטות לזיהוי וסיווג קבצי הרצה מסוג PE המיועדים למערכת ההפעלה Windows ושיטה אחת לזיהוי וסיווג קבצי הרצה מסוג APK המיועדים למערכת ההפעלה Android.

<u>השיטות שייסקרו בפרויקט הן:</u>

1. חילוץ מאפיינים מקבצי PE בשיטת NGRAM המתבססת על תדירות רצפי אופקודים בקובץ הבינארי.
2. חילוץ מאפיינים מקבצי PE בהתבסס על אזור הPE HEADER אשר מכיל את שמות ספריות המערכת המיובאות (Imported Libraries) ואת שמות פונקציות המערכת (System API Functions) שבהם קובץ ההרצה עושה שימוש.
3. חילוץ מאפיינים מקבצי APK בהתבסס על קובץ ה-Android Manifest אשר מכיל מידע אודות ההרשאות שאותן דורשת האפליקציה.

<u>המסווגים (Classifiers) אשר בהם ייעשה שימוש הם:</u>

1. KNN
2. SVM
3. Random Forest
4. Logistic Regression

בפרויקט נעשה שימוש בData Sets המכילים <u>קבצים הרצה גולמיים</u>.
במסגרת הפרויקט עבור כל Data Set גולמי כזה ועבור כל שיטה מן השיטות שפורטו לעיל, התוכנה תבצע את חילוץ המאפיינים ותבנה קובץ Data Set <u>טבלאי</u> מתאים מסוג CSV אשר יכיל את אוסף וקטורי המאפיינים (feature vectors) עם תיוג (מזיק/תמים) .

כמו כן, בפרויקט אבצע הערכת איכות (Evaluation) של כל אחד מן הData Sets שנבנו ובכל מסווג מן המסווגים שצוינו לעיל. הערכת האיכות תתבצע במדדי F1, Accuracy Rate בשיטת K-Fold cross validation.

בנוסף, נבנה ממשק משתמש מתאים, אשר יביא לידי ביטוי את היכולות אשר נבנו בפרויקט. המשתמש יוכל להכניס קובץ קלט ולבחור שיטה (מודל) על פיו הקובץ ינותח. התוכנה תחלץ מאפיינים לפי השיטה שנבחרה ותבצע חיזוי (prediction) לקובץ על פי כל אחד מן המסווגים ותחזיר את החלטת הרוב, התוצאה תוצג למשתמש בממשק הגראפי.

2.2 תכולות העבודה

א. חיפוש ואיתור של *Data Sets* המכילים קבצי הרצה גולמיים מסוג *PE* ו*APK*
ב. ניתוח של קובץ *PE* בשימוש במאפיינים מסוג *NGRAM*
ג. ניתוח של קובץ *PE* בשימוש במאפיינים מאזור ה*PE HEADER*
ד. ניתוח של קובץ *APK* בשימוש במאפיינים מה-*Android Manifest*
ה. בנייה של *Data Sets* מתאימים לכל אחת מן השיטות
ו. בנייה של Predictor העושה שימוש במסווגים (Classifiers) שנבחרו
ז. בנייה של *Evaluator* אשר מבצע הערכת איכות לכל אחד מן ה*Data Sets* שנוצרו
ח. בנייה של ממשק משתמש שיאפשר ניתוח של קובץ קלט במודל שנבחר.

2.3 תיאור סביבת העבודה

מכיוון שהפרויקט דורש עבודה וניתוח של קבצי הרצה זדוניים ומסוכנים, ולמרות שלא נדרש להריצם במחשב המארח שכן הניתוח מתבצע על סמך מאפיינים סטטיים בלבד, אך בשל החשש שהקבצים יורצו בטעות או יגרמו נזק למחשב נדרש לעבוד בסביבה מבודדת.

לשם כך ובכדי שיהיה ניתן לבטל הגנות אנטי וירוס שיפריעו ולא יאפשרו את תהליך הפיתוח והמחקר, נדרש לעבוד על מכונה וירטואלית, אשר מבודדת לחלוטין מהסביבה המארחת, ומספקת אבסטרקציה מלאה לתוכנות אשר ירוצו עליה באופן שלא ניתן יהיה לדעת שזו איננה מכונה פיזית אמיתית.

בחרתי לעבוד על מכונה וירטואלית מסוג *Windows Server 2018* אשר רצה באמצעות מנוע המאפשר הרצה של מכונות וירטואליות-*Microsoft Hyper-V* .

התוכנה בפרויקט פותחה בשפת *Python* בגרסא 3.9.

2.4 תיאור ה*Datas Sets* שנבחרו

בפרויקט נעשה שימוש ב*Datasets* מוכרים מספרות המחקרית בתחום.
עבור נוזקות מיועדות *PC* ונוזקות מיועדות אנדרואיד נבחרו 2 *Datasets* שונים שעבורם יבוצע תהליך אימון וחיזוי וכן בהמשך הסיכום תפורט השוואה בין תוצאות החיזוי במודלים השונים בשימוש בכל אחד מה-*Datasets*.

<u>עבור נוזקות מיועדות *Android* נעשה שימוש ב*Datasets* הבאים:</u>

- ***StormDroid_KuafuDet*** - מאגר של כ2100 נוזקות למערכת אנדרואיד
- ***MalDroid*** – מאגר של כ3000 נוזקות למערכת אנדרואיד
- ***Google Play Store***- תת מאגר ב-*Maldroid Dataset* אשר מאגד כ700 קבצי *apk* תמימים אשר מקורם בחנות הרשמית (*Google Play Store*).

<u>עבור נוזקות מיועדות *PC* נעשה שימוש ב*Datasets* הבאים:</u>

- ***Virusshare*** - מאגר של כ1800 נוזקות ל*PC* במערכת הפעלה *Windows*
- ***Malicia*** - מאגר של כ2500 נוזקות ל*PC* במערכת הפעלה *Windows*
- ***Benign Files***- תת מאגר ב-*Malicia Dataset* אשר מאגד כ1000 קבצי *EXE* אשר ידועים כתמימים ונפוצים בשימוש למערכת הפעלה *Windows* .

2.5 הערכת איכות

הערכת איכות של ה-*Datasets* הסופיים שנוצרו בכל אחת מן השיטות, תבוצע בשיטת:
K-Fold cross validation , כאשר המדדים יהיו: *Accuracy, F1* ו*K=5*.

בשיטה הנ"ל, מחלקים *Dataset* בגודל *n* ל*k* תתי קבוצות בגודל שווה. ועבור כל קבוצה מריצים את האלגוריתם כך $\frac{n-k}{n}$ מהאיברים משמשים עבור הלמידה ו $\frac{k}{n}$ מהם משמשים לבדיקה ואימות החיזוי (*prediction*).

- <u>מדד *Accuracy* מחושב בצורה הבאה:</u>

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- <u>מדד *F1* מחושב בצורה הבאה:</u>

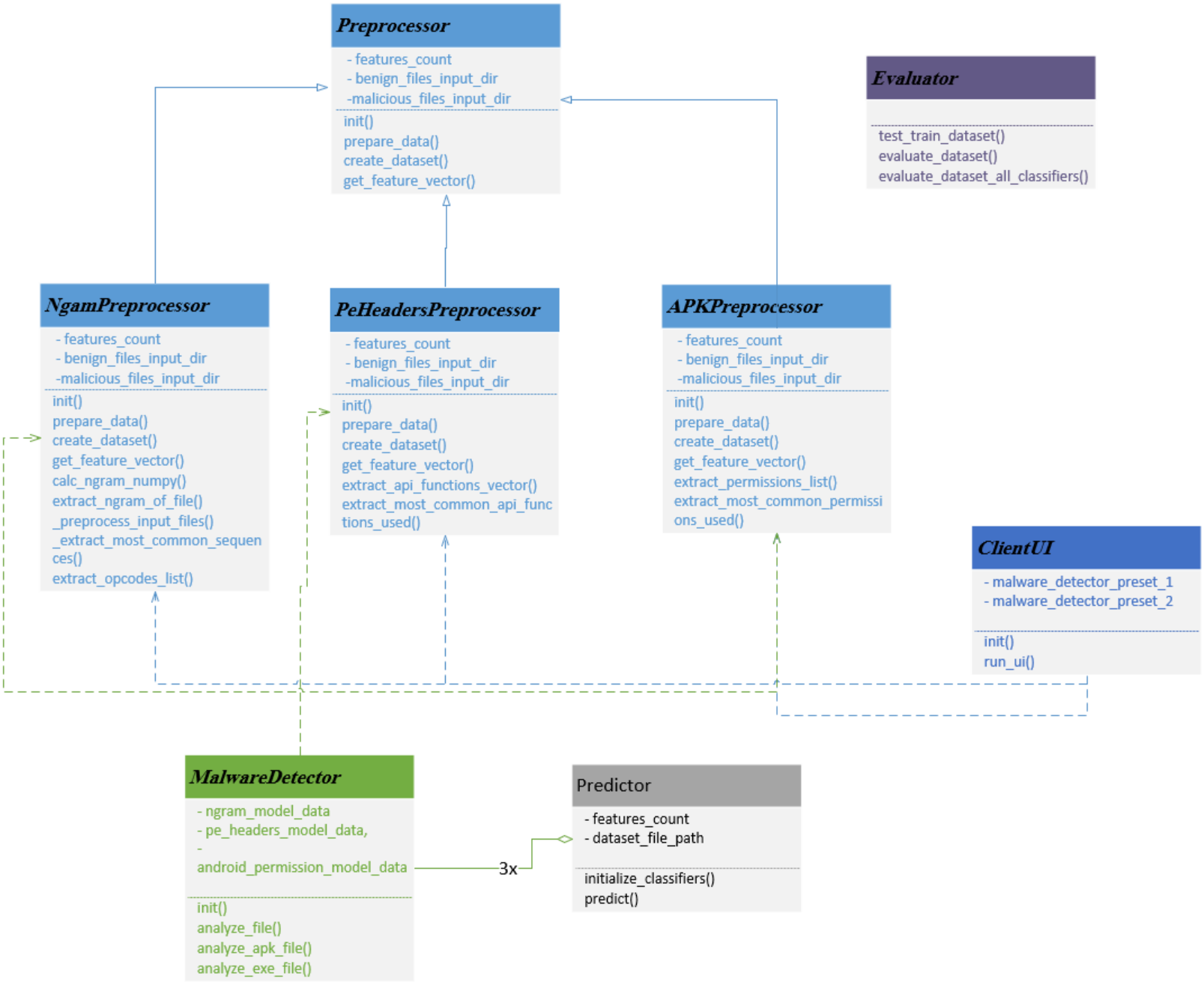
$$F1\text{-}measuer = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

כאשר:

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN}$$

2.6 ארכיטקטורה

התוכנה המרכזית מורכבת מהמודולים המתוארים בדיאגרמת ה*UML* הבאה:


Preprocessor
- features_count
- benign_files_input_dir
-malicious_files_input_dir
init()
prepare_data()
create_dataset()
get_feature_vector()
Evaluator
test_train_dataset()
evaluate_dataset()
evaluate_dataset_all_classifiers()
NgamPreprocessor
- features_count
- benign_files_input_dir
-malicious_files_input_dir
init()
prepare_data()
create_dataset()
get_feature_vector()
calc_ngram_numpy()
extract_ngram_of_file()
_preprocess_input_files()
_extract_most_common_sequences()
extract_opcodes_list()
PeHeadersPreprocessor
- features_count
- benign_files_input_dir
-malicious_files_input_dir
init()
prepare_data()
create_dataset()
get_feature_vector()
extract_api_functions_vector()
extract_most_common_api_functions_used()
APKPreprocessor
- features_count
- benign_files_input_dir
-malicious_files_input_dir
init()
prepare_data()
create_dataset()
get_feature_vector()
extract_permissions_list()
extract_most_common_permissions_used()
ClientUI
- malware_detector_preset_1
- malware_detector_preset_2
init()
run_ui()
MalwareDetector
- ngram_model_data
- pe_headers_model_data,
-
android_permission_model_data
init()
analyze_file()
analyze_apk_file()
analyze_exe_file()
3x
Predictor
- features_count
- dataset_file_path
initialize_classifiers()
predict()

- ***Preprocessor***
  - מודול ממשק אבסטרקטי .
  - מגדיר ממשק אחיד לכל המודולים שמבצעים עיבוד מקים על *Dataset* שמורכב מאוסף קבצים בינאריים במטרה לייצר *Dataset* טבלאי בהתאם למודל אשר מגדיר את אופן חילוץ המאפיינים.
  - מגדיר פונקציונאליות של חילוץ וקטור מאפיינים (*feature vector*)
  - מגדיר פונקציונאליות של בניית קובץ *Dataset*.

- ***NgramPreprocessor***
  - יורש מ-*Preprocessor* ומממש את פונקציות הממשק.
  - מממש לוגיקת חילוץ מאפיינים בצורה סטטית מקובץ בינארי ע"פי מודל *NGRAM*. בפרויקט נבחר להשתמש ב-*N=2*.
  - מממש לוגיקת חיפוש מאפיינים נפוצים ביותר בנוזקות, על מנת לבחור מאפיינים בשיטת *Information Gain*.
  - מייצר קובץ *Dataset* סופי בהינתן תיקיית קבצים זדוניים ותיקיית קבצים תמימים.

- ***PeHeadersPreprocessor***
  - יורש מ-*Preprocessor* ומממש את פונקציות הממשק.
  - מממש לוגיקת חילוץ מאפיינים בצורה סטטית מתוך *Header Section* בקובץ ה*PE*, שם יופיעו פונקציות ה*API* הנקראות מתוך קבצי ה*DLL* הנדרשים לריצת התוכנית.
  - מממש לוגיקת חיפוש מאפיינים נפוצים ביותר בנוזקות (חתימת הפונקציות הנפוצות ביותר בשימוש) על מנת לבחור מאפיינים בשיטת *Information Gain*.
  - מייצר קובץ *Dataset* סופי בהינתן תיקיית קבצים זדוניים ותיקיית קבצים תמימים.

- ***ApkPreprocessor***
  - יורש מ-*Preprocessor* ומממש את פונקציות הממשק.
  - מממש לוגיקת חילוץ מאפיינים בצורה סטטית מתוך קובץ ה*AndroidManifest.xml* אשר נכלל בקובץ ההרצה (קובץ ה*APK*) שם יופיעו הרשאות המערכת הנדרשות לאפליקציה.
  - מממש לוגיקת חיפוש מאפיינים נפוצים ביותר בנוזקות (הרשאות המערכת הנדרשות הנפוצות ביותר בשימוש) בכדי לבחור מאפיינים בשיטת *Information Gain*.
  - מייצר קובץ *Dataset* סופי בהינתן תיקיית קבצים זדוניים ותיקיית קבצים תמימים.

- ***Predictor***
  - מחלקה שמהווה ממשק נוח לביצוע סיווג לקובץ חדש על סמך *dataset* קיים.
  - מחזיק מסווג (*ML Classifier*) מכל סוג אשר נבחר לעשות בו שימוש בפרויקט זה, כמו כן מחזיק *Dataset* טבלאי מעובד.
  - מממש לוגיקת חיזוי *(prediction)* בהינתן וקטור מאפיינים (*feature vector*) של קובץ חדש, יבצע חיזוי על סמך כל אחד מהמסווגים ויחזיר את החלטת הרוב.

- ***MalwareDetector***
  - מודול התוכנה המרכזי.
  - עבור כל מודל שנתמך בפרויקט: יעשה שימוש בקובץ ה- *Dataset* המעובד שנוצר מבעוד מועד, וכן יחזיק מופע של מודול *Preprocessor* ומופע של מודול *Predictor* מתאים.
  - יממש לוגיקת סיווג לקובץ בהינתן נתיב לקובץ ומודל נבחר (מבין המודלים הנתמכים בפרויקט).

- ***ClientUI***
  - מחלקה שמטרתה לייצר ממשק משתמש גראפי פשוט ונוח שיאפשר למשתמש לבצע את הפעולות הקיימות במודול התוכנה המרכזי – *MalwareDetector*.
  - יאפשר למשתמש לבחור קובץ ממערכת הקבצים ולבחור מודל, ולשלוח את הקובץ לניתוח. את התשובה שחזרה יציג למשתמש בממשק הגראפי.

- ***Evaluator***
  - מודול עזר שנועד לבצע הערכת איכות *(evaluation)* לכל אחד מה*Datasets* המעובדים שנוצרו.
  - יחזיק מופעים של כל אחד מהמסווגים (*classifiers*) אשר נעשה בהם שימוש בפרויקט.
  - בהינתן קובץ *Dataset* מעובד יבצע אוולויואציה למדדי *Accuracy, F1* בשיטת *K-Fold Cross Validation* , את התוצאות ידפיס למסך.

3. זיהוי נוזקות PC בשימוש במאפיין *N-GRAM*

3.1 רקע

מאפייני *N-GRAM* מגדירים אפיון של קובץ בינארי על סמך רצפי הפקודות שמופיעות בו.
שיטה זו, מאפשרת חילוץ מאפיינים בצורה סטטית מקבצי הרצה ל*PC* (קבצים בינאריים בפורמט *PE*). בפרויקט זה נתמכים אך ורק קבצים המועדים לארכיטקטורת מעבדי *x86*.
בשיטה זו, לא מתייחסים לכלל מרכיבי קובץ ה-*PE*, אלא לאזור הקוד בלבד (ה-*Code Section*).
לקוד הבינארי נבצע תהליך דה-קומפילציה (*de-compilation*) על מנת לחלץ את רצפי האופקודים בהתאם לארכיטקטורה.
על סמך תדירות הופעת רצפי האופקודים בכל קובץ בינארי, נבנה וקטור מאפיינים מתאים כמתואר ב-תיאור השיטה.
NGRAM היא שיטה כללית עבור מספר טבעי-N. בפרויקט זה נעשה שימוש ב-$N=2$.
שיטה זו מתוארת בפירוט במאמר [1].

3.2 תיאור השיטה

נרצה לייצג קובץ בינארי (קובץ הרצה) באמצעות רצפי פקודות (*opcode-sequence*) ותדירות ההופעה שלהם בקובץ.

בהינתן תוכנה p המורכבת מסט פקודות: $\rho = (o_1, o_2, o_3, o_4, ..., o_{\ell-1}, o_\ell)$

כאשר l יהיה המספר הכולל של פקודות *code section* של התוכנה.

נגדיר רצף פקודות os: $os = (o1, o2, o3, ..., o_{m1}, o_m)$ להיות תת קבוצה של פקודות כך

ש: $os \subseteq p$. m יהיה אורך רצף הפקודות.

לדוגמא, בהינתן התוכנית הבאה:

```
def add_forty_two(n)

  pushq %rbp
  movq  %rsp, %rbp
  addl  $42, %edi
  movl  %edi, %eax
  popq  %rbp
  retq

end
```

נוכל לחלץ רצפי פקודות באורך 2:

S_1 = *(push, mov)*

S_2 = *(mov, add)*

S_3 = *(add, mov)*

S_4= *(mov, pop)*

....

לאחר מכן, נחשב את התדירות של כל רצף פקודות. נגדיר את המושג <u>תדירות רצף</u>:

(term frequency) או בקיצור ***tf*** להיות התדירות של הרצף באוסף כל הרצפים האפשריים הקיימים בתוכנית:

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

כאשר $n_{i,j}$ מוגדר להיות כמות הפעמים שהרצף $s_{i,j}$ הופיע בתוכנית.

ו $\sum_k n_{k,j}$ יהיה הסכום הכולל של כל רצפי הפקודות שיופיעו בקובץ הבינארי.

כאשר בוחרים במספר רב מידי של מאפיינים, תהליך הלמידה נהפך איטי ומורכב.

ולכן נבצע <u>בחירת מאפיינים ע"ס שינוי כמות האינפורמציה</u>- (*feature selection using Information Gain*)

נגדיר את המושג <u>משקל תדירות פקודה</u> *(Weighted Term Frequency)* או בקיצור ***'wtf'*** בכדי למדוד את מידת הרלוונטיות של כל פקודה כאשר מחשבים את תדירות הרצף.

באמצעות מאגר המידע הזה, נרצה לחשב את מדד השינוי באנטרופיה (*'information gain'*) עבור כל אופקוד שהיה בשימוש. כך נדע לסווג ולתת חשיבות גבוהה יותר לפקודות הנפוצות בנוזקות המוכרות:

$$I(X;Y) = \sum_{y \epsilon Y} \sum_{x \epsilon X} p(x,y) \log \left(\frac{p(x,y)}{p(x) \cdot p(y)} \right)$$

כאשר *Y={malware, benign}* ו *X={push, mov, add, xor ...}*

כלומר המחלקה Y תייצג את התוצאה (נוזקה/ לגיטימי) וX את אוסף כל האופקודים הקיימים.

כך נייצר מדד רלוונטיות עבור האופקודים הקיימים.

נגדיר את משקל תדירות הפקודה *(Weighted Term Frequency)* . ***wtf***$_{i,j}$ באופן הבא:

$$wtf_{i,j} = tf_{i,j} \cdot \prod_{o_z \epsilon S} \frac{weight(o_z)}{100}$$

כאשר $tf_{i,j}$ הוא תדירות הרצף עבור הרצף $s_{i,j}$ [באופן שהוגדר לעיל].

ו *weight*(o_z) הוא המשקל בחישוב ע"פי שינוי האנטרופיה עבור כל אופקוד o_z שברצף $s_{i,j}$.

לבסוף, יתקבל וקטור מאפיינים הנבנה המורכב ממשקל תדירויות הרצפים:

$$v = ((os_1, wtf_1), ..., (os_n, wtf_n))$$

בפרויקט בחרתי להשתמש ב-**230** המאפיינים המשמעותיים ביותר.

3.3 דוגמאות מהפרויקט

- להלן דוגמא למאפייני *NGRAM* הנפוצים ביותר בנוזקות מ-*Virushare -Dataset* ותדירות הופעתם בקבצים:

# OPCODE1	OPCODE2	COMBINED FREQUENCIES
mov	mov	5557665
push	push	3522152
int3	int3	3430057
add	add	2567554
push	call	2113188
push	mov	1994911
mov	push	1678171
call	mov	1324399
mov	cmp	1152273
mov	call	1135414
cmp	jne	990636
lea	push	976607
push	lea	976558
pop	pop	945002
cmp	je	821795
call	add	802300
…	…	….

- דוגמא למאפייני *NGRAM* הכי פחות נפוצים בנוזקות מאותו ה-*Dataset*:

xor	vphadduwd	1
xorps	jo	1
xorps	test	1
xrelease xchg	jno	1
xrstors	cld	1
xsave	wait	1
xsavec	rcl	1
cli	setge	1
lock and	jb	1
pcmpgtw	jnp	1
repe cmpsb	.byte	1
cmc	punpckhwd	1
fldenv	fucomip	1
jae	vpminsw	1
jb	rep insb	1
rep insb	insb	1
vpminsw	mov	1
fcmovnbe	fmulp	1
bnd call	hlt	1
jb	vpor	1
vpor	and	1

[הקבצים המלאים בפורמט *CSV* יופיעו בתיקיית *out-files* בתיקיית תוצרי הפרויקט]

3.4 תוצאות המחקר

לאחר בניית ה-*Dataset* על בסיס המודל שתואר לעיל, בוצעה הערכת איכות (*evaluation*) ל-*Dataset* שנוצר בשיטת *K-Fold cross validation* במדדי *F1, Accuracy*. כמתואר בתת פרק- הערכת איכות.
האוולויואציה בוצעה עבור שני *Datasets* שנבחרו לשימוש בפרויקט זה.

<u>להלן התוצאות שהתקבלו:</u>

<u>עבור מדד Accuracy:</u>

Model / Dataset	Virusshare Dataset	Malicia Dataset
KNN	0.906	0.943
Random Forest	0.955	0.965
SVM	0.895	0.912
Logistic Regression	0.879	0.876

<u>עבור מדד F1:</u>

Model / Dataset	Virusshare Dataset	Malicia Dataset
KNN	0.898	0.925
Random Forest	0.955	0.945
SVM	0.881	0.877
Logistic Regression	0.857	0.819

כפי שניתן לראות, מודל NGAM הניב אחוזי דיוק גבוהים מאד, ב-2 הDatasets שנבחרו,
כאשר המסווג בעל אחוזי הדיוק גבוהים ביותר היה Random Forest.

3.5 סיכום ומסקנות

התוצאות שהושגו בניסוי מאמתות את ההנחה שניתן לבנות מערכת לזיהוי נוזקות חדשות שלא נצפו בעבר על סמך מאפיינים סטטיים שחולצו בשיטת *N-GRAM* וכן להגיע לאחוזי דיוק מרשימים.

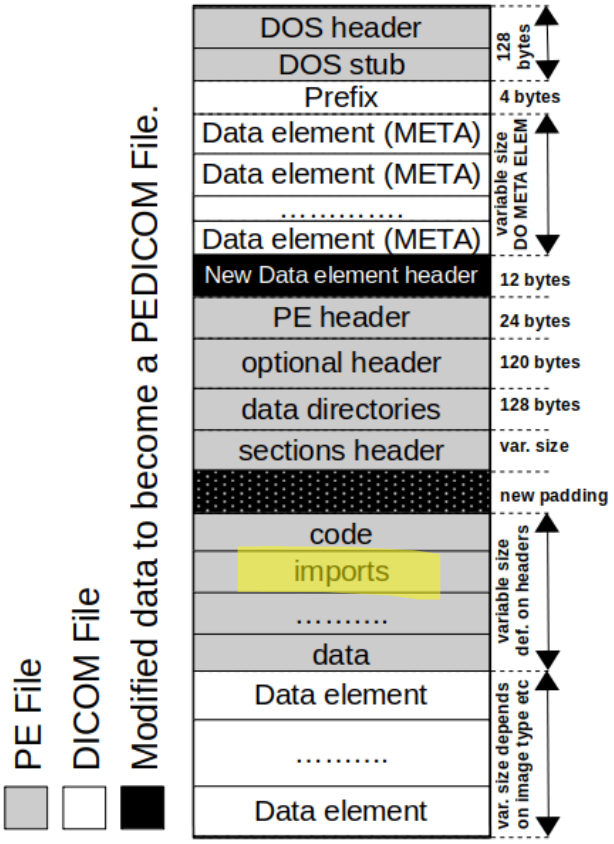
<u>עם זאת, יש עדיין כמה חסרונות בשימוש בשיטה הזו:</u>

1. בשיטה זו לא ניתן להתמודד עם נוזקות דחוסות/מוצפנות (*packed*) כלומר, תוכנות ששומרות את מירב הקוד בצורה מוצפנת ובמהלך הריצה מפענחות אותו, מעתיקות אותו לזיכרון ומריצות אותו.
2. השיטה מסוגלת לפעול אך ורק על קבצי הרצה בפורמט תקין ועבור ארכיטקטורת מעבד ידועה מראש. לעיתים רבות יש צורך בזיהוי נוזקות/חדירות והתראה על סמך תעבורת רשת/ קבצי קונפיגורציה ואחרים, ועבור המקרים הללו השיטה הזו לא רלוונטית.

4. <u>זיהוי נוזקות PC בשימוש במאפייני *PE Header*</u>

4.1 רקע

<u>פורמט קובץ הרצה במערכת ההפעלה *Windows* , נקרא גם פורמט *PE* ובנוי בצורה הבאה :</u>

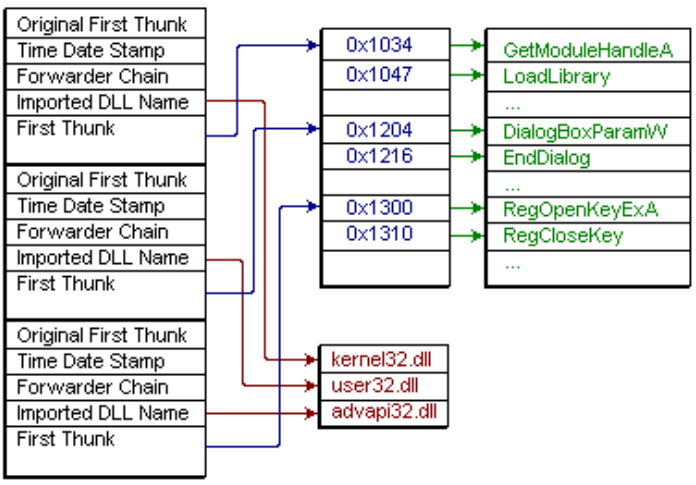


בניגוד לשיטה הקודמת, בה חילוץ המאפיינים התבצע על סמך הקוד שבאזור ה*Code Section* , כעת חילוץ המאפיינים יתבצע על בסיס ה-*Header* של קובץ ה*PE* ובאופן יותר מפורט – אזור ה-*Import table*.

ה*Import Table* מכיל שדה עבור כל ספריה דינאמיות *(DLL)* בה תלוי קובץ ההרצה – נקרא *DLL Entry*.

בכל אחד מה-*Entries* נמצאת כתובת/שם של פונקציה אותה מייצא אותו *DLL* ובה משתמשים בתוכנית.

<u>לדוגמא :</u>



4.2 תיאור השיטה

בהינתן קובץ בינארי בפורמט *PE*, נדרש לבצע את התהליך הבא:

1. פרסור (*Parsing*) של הקובץ באמצעות ספרייה/כלי ייעודי לניתוח קבצי *PE*
2. חילוץ ה-*DLL Entry Table*
3. עבור כל *Entry Table*:
   a. חילוץ של כל-*API Function* בשימוש מתוך ה*DLL* וצירופם לרשימה הסופית.

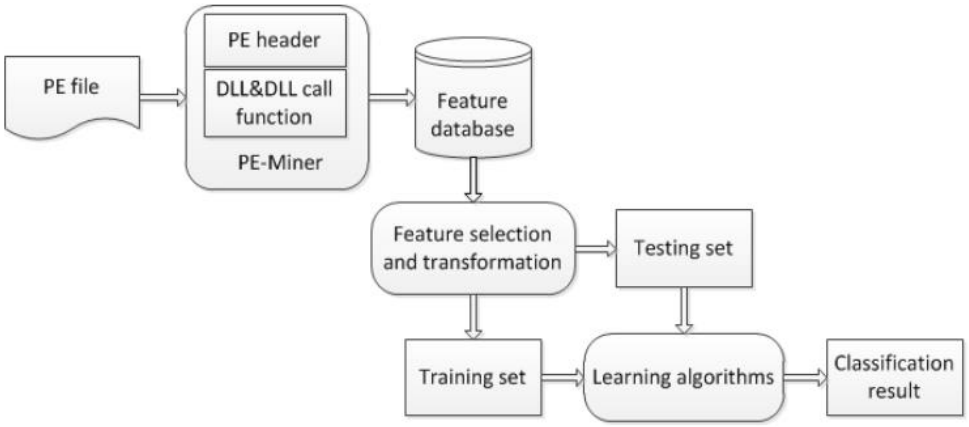
מכיוון שכמות גדולה מידי של מאפיינים תהפוך את תהליך הלמידה והחיזוי לאיטי ומסורבל, גם בשיטה זו בחירת המאפיינים המשמעותיים ביותר תתבצע ע"פי Information Gain.

על כן, לפני חילוץ המאפיינים מכל קבצי הDataset הגולמי אבצע סריקה מקדימה של קבצי הנוזקות הגולמיים בלבד, ומהם אחלץ את פונקציות המערכת (API Functions) כמתואר לעיל.
כלומר, אוסף המאפיינים (features) יתבסס על שמות פונקציות המערכת בהן נעשה שימוש.
נסמן את אוסף פונקציות המערכת הנפוצות ביותר בשימוש ב- **S**.

עבור כל קובץ נגדיר וקטור מאפיינים בצורה הבאה: $F = \{F_1, F_2, F_3, \ldots F_n\}$

כאשר $F_i=1$ אם $F_i \in S$, אחרת $F_i=0$. ו-n הוא מספר המאפיינים.
בפרויקט זה בחרתי לעבור עם n=**110** המאפיינים המשמעותיים ביותר.

סכמה כללית של תיאור תהליך החיזוי בשיטה זו:



[מתוך מאמר 2]

4.3 דוגמאות מהפרויקט

- דוגמא לפונקציות המערכת הנפוצות ביותר בשימוש בנוזקות מ-Virusshare – Dataset ותדירות הופעתן בקבצים:

# FUNCTION	COUNT
RtlUnwind	1432
GetModuleHandleA	1319
GetProcAddress	1148
GetLastError	1118
GetTickCount	1099
MultiByteToWideChar	1080
LoadLibraryA	1071
GetCurrentProcess	1061
GetStringTypeA	1059
RegCloseKey	1056
CloseHandle	1055
WaitForSingleObject	1051
GetACP	1049
InitializeCriticalSection	1047
ExitProcess	1046
WriteFile	1029
GetModuleFileNameA	1027
EnterCriticalSection	1025
LeaveCriticalSection	1023
Sleep	1021
VirtualFree	1010

- ודוגמא לפונקציות המערכת הכי פחות נפוצות בקבצים:

PolyDraw	1
SetPolyFillMode	1
CreateDCA	1
StrRChrIA	1
FindCloseChangeNotification	1
SetPriorityClass	1
UuidCreate	1
StrStrIA	1
StrCatW	1
WSCInstallProvider	1
WSCWriteProviderOrder	1
WSCEnumProtocols	1
CreateAcceleratorTableA	1
EnumWindowStationsW	1
SelectClipPath	1
CommandLineToArgvW	1
GetConsoleWindow	1
GetAncestor	1
SetLayout	1

[הקבצים המלאים בפורמט *CSV* יופיעו בתיקיית *out-files* בתיקיית תוצרי הפרויקט]

4.4 תוצאות המחקר

עבור מדד Accuracy:

Model / Dataset	Virusshare Dataset	Malicia Dataset
KNN	0.933	0.855
Random Forest	0.971	0.97
SVM	0.965	0.983
Logistic Regression	0.963	0.961

עבור מדד F1:

Model / Dataset	Virusshare Dataset	Malicia Dataset
KNN	0.928	0.836
Random Forest	0.972	0.962
SVM	0.961	0.968
Logistic Regression	0.96	0.958

גם שיטה זו הניבה אחוזי דיוק גבוהים, ב-2 הDatasets שנבחרו, כאשר המסווגים: SVM, Random Forest השיגו את אחוזי הדיוק הגבוהים ביותר.

4.5 סיכום ומסקנות

בשיטה זו אחוזי הדיוק היו גבוהים גם כן, ואף מעט יותר גבוהים מאשר בשיטת *NGRAM*.
כמו כן, לשיטה זו מתגברת על אחד היתרונות המרכזיים בשיטת *NGRAM* שכן גם נוזקות דחוסות/מוצפנות *(packer)* לא יכולות להימנע מלהצהיר בקובץ ההרצאה (ה-*PE*) על הספריות (*DLL*) ופונקציות המערכת בהן הן עושות שימוש.

עם זאת, יש עדיין כמה חסרונות בשיטה הזו:

1. נוזקות מתוחכמות שירצו להתחמק מזיהוי בשיטה זו יאופיינו ב'חתימה' רזה, כלומר קובץ בינארי מינימאלי אשר מטרתו תהיה לבצע תקשורת לשרת מרוחק ומשם יוריד את תוכן הנוזקה המקורי.
2. גם שיטה מסוגלת לפעול אך ורק על קבצי הרצה בפורמט *PE* תקין. לעיתים רבות יש צורך בזיהוי נוזקות/חדירות והתראה על סמך תעבורת רשת/ קבצי קונפיגורציה ואחרים, ועבור המקרים הללו השיטה הזו לא רלוונטית.

5. זיהוי נוזקות *Android* בשימוש במאפייני *Android Manifest*

5.1 רקע

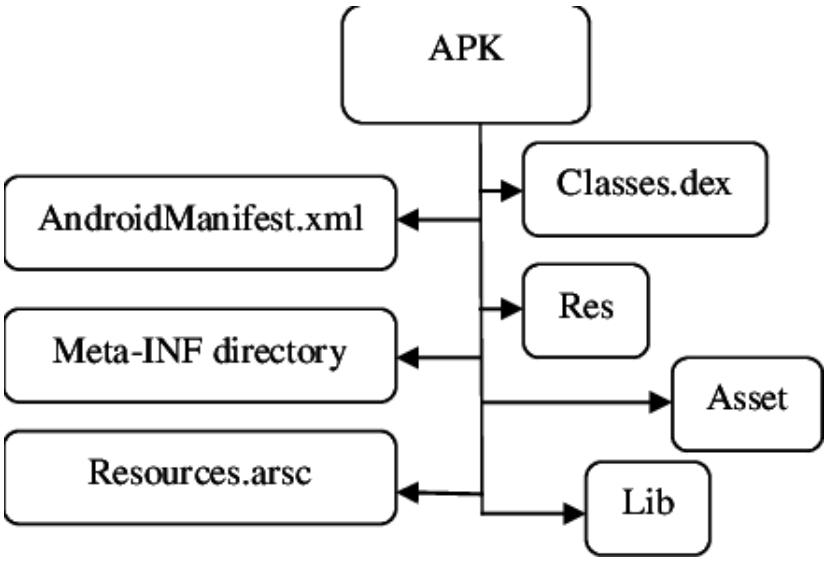
בשנים האחרונות חלה עלייה משמעותית בכמות הנוזקות המיועדות למכשירים ניידים, ובפרט למכשירים בעלי מערכת ההפעלה אנדרואיד.

זיהוי וסווג נוזקות המיועדות למכשירי אנדרואיד הינה מטרה מתבקשת, שכן המודעות לחדירות למכשירים הניידים אינה גבוה דיה וכן אמצעי ההגנה לוקים בחסר.

השיטה בה נעשה שימוש בפרויקט הינה חילוץ מאפיינים בצורה סטטית מתוך הקובץ הבינארי של האפליקציה (קובץ ה-*APK*).

קובץ *APK* הינו קובץ דחוס בפורמט המוכר- *ZIP* ומכיל מספר קבצים עיקריים:

- קבצי *java* מקומפלים – בפורמט *dex*.
- קובץ *AndroidManifest.xml* עליו ארחיב בהמשך הפרק.
- קבצי *inf* המכילים חתימות דיגיטאליות של ה*Manifest*.
- קבצי *lib* המכילים ספריות מקומפלות בהתאם לארכיטקטורת המעבד.
- תיקיית *res* המכילה קבצים לא מקומפלים
- תיקיית *assets* המכילה קבצי תוכן רלוונטים עבור האפליקציה כגון קבצי מוזיקה, פונטים וכדומה..



[מבנה פורמט קובץ *APK*]

בפרויקט, חילוץ המאפיינים מתבצע ממקור אחד והוא קובץ ה-AndroidManifest.xml
הקובץ הוא קובץ טקסטואלי בפורמט XML ומכיל מידע הצהרתי על האפליקציה, כגון:
שם האפליקציה, מספר גרסא, גרסת אנדרואיד נתמכת, שמות של activities וכן אוסף ההרשאות הנדרשות ע"י האפליקציה כאשר לכל הרשאה יש תווית (label) ייחודית.

להלן דוגמא לאפליקציה בשם **app_name** בגרסא **1.0** הדורשת את ההרשאות:
READ_FRIENDS, WRITE_FRIENDS, FRIEND_SERVICE,
RECEIVE_BOOT_COMPLETED, READ_CONTACTS, ACCESS_FINE_LOCATION.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="org.siislab.tutorial.friendtracker"
      android:versionCode="1"
      android:versionName="1.0.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".FriendTrackerControl"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <provider android:authorities="friends"
              android:name="FriendProvider"
              android:writePermission="org.siislab.tutorial.permission.WRITE_FRIENDS"
              android:readPermission="org.siislab.tutorial.permission.READ_FRIENDS">
        </provider>
        <service android:name="FriendTracker" android:process=":remote"
              android:permission="org.siislab.tutorial.permission.FRIEND_SERVICE">
        </service>
        <receiver android:name="BootReceiver">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED"></action>
            </intent-filter>
        </receiver>
    </application>

    <!--  Define Permissions -->
    <permission android:name="org.siislab.tutorial.permission.READ_FRIENDS"></permission>
    <permission android:name="org.siislab.tutorial.permission.WRITE_FRIENDS"></permission>
    <permission android:name="org.siislab.tutorial.permission.FRIEND_SERVICE"></permission>

    <!--  Uses Permissions -->
    <uses-permission android:name="org.siislab.tutorial.permission.READ_FRIENDS"></uses-permission>
    <uses-permission android:name="org.siislab.tutorial.permission.WRITE_FRIENDS"></uses-permission>
    <uses-permission android:name="org.siislab.tutorial.permission.FRIEND_SERVICE"></uses-permission>

    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"></uses-permission>
    <uses-permission android:name="android.permission.READ_CONTACTS"></uses-permission>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"></uses-permission>
</manifest>
```

השיטה בה נעשה שימוש בפרויקט זה מתמקדת בחילוץ ההרשאות המוצהרות מתוך הקובץ הנ"ל ובניית וקטור מאפיינים בהתאם.

5.2 תיאור השיטה

בהינתן קובץ בינארי בפורמט *APK*, נדרש לבצע את התהליך הבא:

1. חילוץ ופריסה של הקובץ הדחוס, באמצעות ספריית תוכנה ייעודית ייעודי לניתוח קבצי *APK*
2. חילוץ קובץ ה-*AndroidManifest.xml*
3. פרסור (*Parsing*) של קובץ ה-*AndroidManifest* לשם חילוץ ההרשאות הנדרשות.

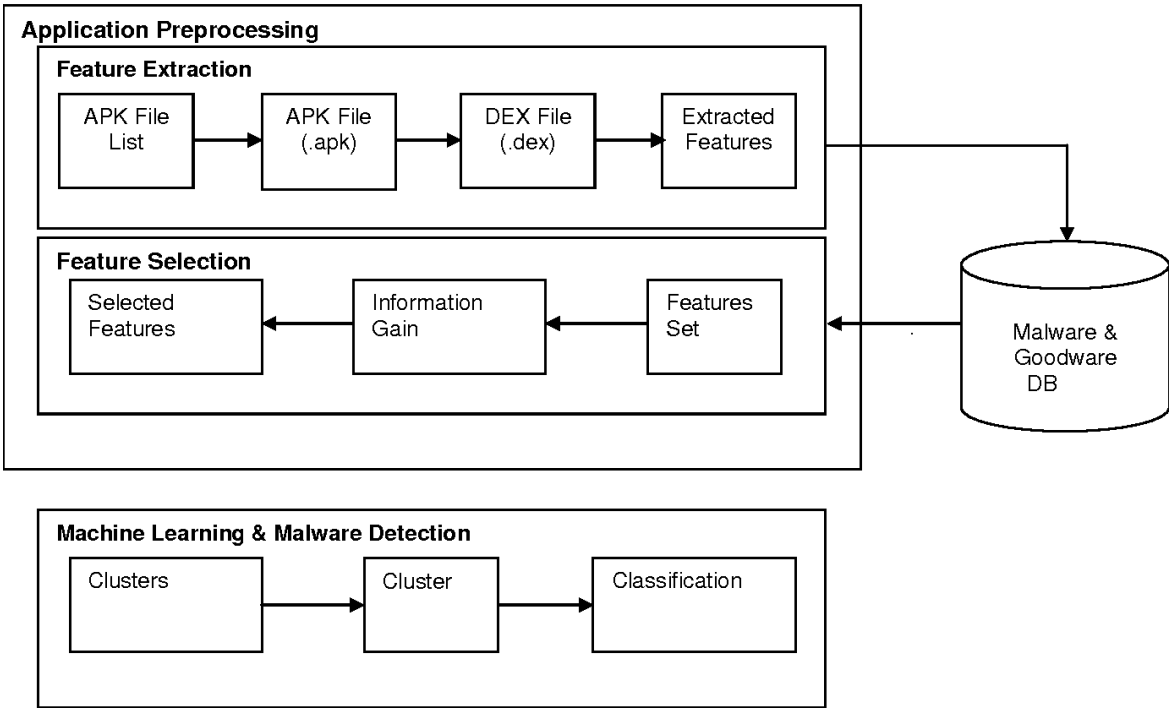
מכיוון שכמות גדולה מידי של מאפיינים תהפוך את תהליך הלמידה והחיזוי לאיטי ולא יעיל, גם בשיטה זו המאפיינים נבחרו ע"פי שיטת Information Gain.

על כן, לפני חילוץ המאפיינים מכל קבצי ה-Dataset הגולמי, אבצע סריקה מקדימה של קבצי הנוזקות הגולמיים, ומהם אחלץ את תוויות (labels) ההרשאות הנפוצות ביותר בשימוש בנוזקות.
כלומר, אוסף המאפיינים (features) יהיה שמות הרשאות המערכת הנדרשות.
נסמן את אוסף פונקציות המערכת הנפוצות ביותר בשימוש ב- **S**.

גם בשיטה זו , עבור כל קובץ נגדיר וקטור מאפיינים בצורה הבאה: $F = \{F_1, F_2, F_3, \ldots F_n\}$

כאשר $F_i=1$ אם $F_i \in S$, אחרת $F_i=0$. ו-n הוא מספר המאפיינים.
בפרויקט זה בחרתי לעבור עם **n=150** המאפיינים המשמעותיים ביותר.

<u>סכמה כללית של תיאור תהליך החיזוי בשיטה זו:</u>



5.3 דוגמאות מהפרויקט

1. דוגמא להרשאות הנדרשות הנפוצות ביותר בשימוש באפליקציות שתויגו כנוזקות מ-MalDroid – Dataset ותדירות הופעתן בקבצים:

# PERMISSION	COUNT
android.permission.INTERNET	3480
android.permission.READ_PHONE_STATE	3417
android.permission.WRITE_EXTERNAL_STORAGE	3158
android.permission.ACCESS_NETWORK_STATE	3105
android.permission.ACCESS_WIFI_STATE	2395
android.permission.GET_TASKS	2219
android.permission.RECEIVE_BOOT_COMPLETED	2064
android.permission.WAKE_LOCK	1782
android.permission.RECEIVE_SMS	1749
android.permission.SEND_SMS	1700
android.permission.READ_SMS	1593
android.permission.READ_CONTACTS	1487
android.permission.SYSTEM_ALERT_WINDOW	1447
android.permission.VIBRATE	1425
android.permission.CHANGE_WIFI_STATE	1387
android.permission.CALL_PHONE	1274
android.permission.WRITE_SMS	1108
android.permission.ACCESS_COARSE_LOCATION	941
com.android.launcher.permission.INSTALL_SHORTCUT	927
android.permission.ACCESS_FINE_LOCATION	878
android.permission.READ_LOGS	821
android.permission.WRITE_SETTINGS	812
android.permission.MOUNT_UNMOUNT_FILESYSTEMS	752
android.permission.RESTART_PACKAGES	688
android.permission.KILL_BACKGROUND_PROCESSES	671
android.permission.CHANGE_NETWORK_STATE	644
android.permission.WRITE_CONTACTS	592
android.permission.READ_EXTERNAL_STORAGE	525
android.permission.DISABLE_KEYGUARD	448
android.permission.MODIFY_PHONE_STATE	408
android.permission.PROCESS_OUTGOING_CALLS	399
com.android.launcher.permission.UNINSTALL_SHORTCUT	370
android.permission.MODIFY_AUDIO_SETTINGS	342
android.permission.CAMERA	320

2. ודוגמא לפונקציות המערכת הכי פחות נפוצות בקבצים:

android.permisison.ACCESS_SUPERUSER	1
com.d8hdp.mingge.ACCESS_WIFI_STATE	1
com.baidu.permission.SHARE	1
com.example.battery.permission.JPUSH_MESSAGE	1
android.permission.DUMP	1
android.permission.WRITE_GSERVICES	1
com.sec.android.provider.logsprovider.permission.READ_LOGS	1
com.wefeelsecure.feelsecure.permission.C2D_MESSAGE	1
com.sec.android.provider.logsprovider.permission.WRITE_LOGS	1
android.permission.BIND_INPUT_METHOD	1

5.4 תוצאות המחקר

עבור מדד Accuracy:

Model / Dataset	MalDroid Dataset	StormDroid_KuafuDet Dataset
KNN	0.914	0.852
Random Forest	0.959	0.904
SVM	0.945	0.872
Logistic Regression	0.944	0.881

עבור מדד F1:

Model / Dataset	MalDroid Dataset	StormDroid_KuafuDet Dataset
KNN	0.819	0.808
Random Forest	0.922	0.866
SVM	0.89	0.829
Logistic Regression	0.885	0.84

ניתן להבחין כי המסווג Random-Forest הניב את אחוזי הדיוק הטובים ביותר.
כמו כן, ניתן לראות ששימוש בשיטה זו לסיווג נוזקות אנדרואיד הניבה אחוזי דיוק גבוהים מאד ואכן ההרשאות המוצהרות ע״י האפליקציה יכולים להעיד במידה רבה מאד על כוונותיה.

5.5 סיכום ומסקנות

התוצאות שהושגו במחקר מאמתות את ההנחה שניתן לבנות מערכת לזיהוי וסיווג נוזקות אנדרואיד (שלא נצפו בעבר) על סמך מאפיינים סטטיים בלבד כגון ההרשאות המוצהרות ע״י האפליקציה. ואף להגיע לאחוזי דיוק גבוהים.

אחד היתרונות שקיימים במערכת ההפעלה *Android*, לעומת מערכת ההפעלה *Windows* - היא מנגנון ההרשאות המורכב הדורש ממפתחי האפליקציה הצהרה מראש על מגוון רחב של פעולות אותו האפליקציה נדרשת לבצע. דבר זה מאפשר אפיון טוב של מטרות וכוונות האפליקציה, גם מבלי להריץ אותה בסביבת סימולציה דינאמית.

לשיטה עדיין קיים חסרון משמעותי:
כיום ישנן נוזקות מתוחכמות אשר ע״י טכניקות הזרקה או טכניקות מתקדמות אחרות, מסוגלות לרוץ במסגרת מרחב (*Context*) של אפליקציות אחרות שרצות על המכשיר ולהן כבר יש הרשאות לפעולות להן נדרשת הנוזקה, או שמסוגלות לרוץ כחלק מתהליך של מערכת ההפעלה.
ואילו עדיין מדובר בסוג נדיר ומתוחכם מאד של נוזקות ובמקרים רבים שיטה זו תהיה אפקטיבית.

6. אופן שימוש בתוכנה

6.1 סביבת הרצה

סביבת ההרצה הנדרשת לתוכנה היא תוכנת *Python* בגרסא 3.7.11 ומעלה.
כמו כן, נדרש להתקין את רשימת הספריות המפורטות בקובץ *requirements.txt* בתיקיית הפרויקט.
מכיוון שפרויקט זה מערב עבודה עם נוזקות מסוכנות ובכדי למנוע נזק למחשב המארח, יש לעבוד עם מכונה וירטואלית המריצה מערכת הפעלה *Windows* בגרסא 10 ומעלה.
בפרויקט אני בחרתי לעבוד עם הסביבה החינמית *Microsoft Hypervisor* המגיעה עם מערכת ההפעלה *Windows 10 pro* .

6.2 דוגמאות הרצה

לתוכנה המרכזית 3 תרחישי ריצה אפשריים:

1. הרצה של תוכנית שתיצור מחדש את קבצי ה*Dataset* המעובדים.
   ניתן לבצע זאת ע"י שורת ההרצה הבאה:

```
python main.py -p CREATE_DATASETS
```

2. הרצה של תוכנית אשר תבצע אוולויואציה לכל אחד מה-*Datasets* שנוצרו באמצעות מודול ה*Evaluator* אשר תואר בפרק ארכיטקטורה.
   ניתן לבצע זאת ע"י שורת ההרצה הבאה:

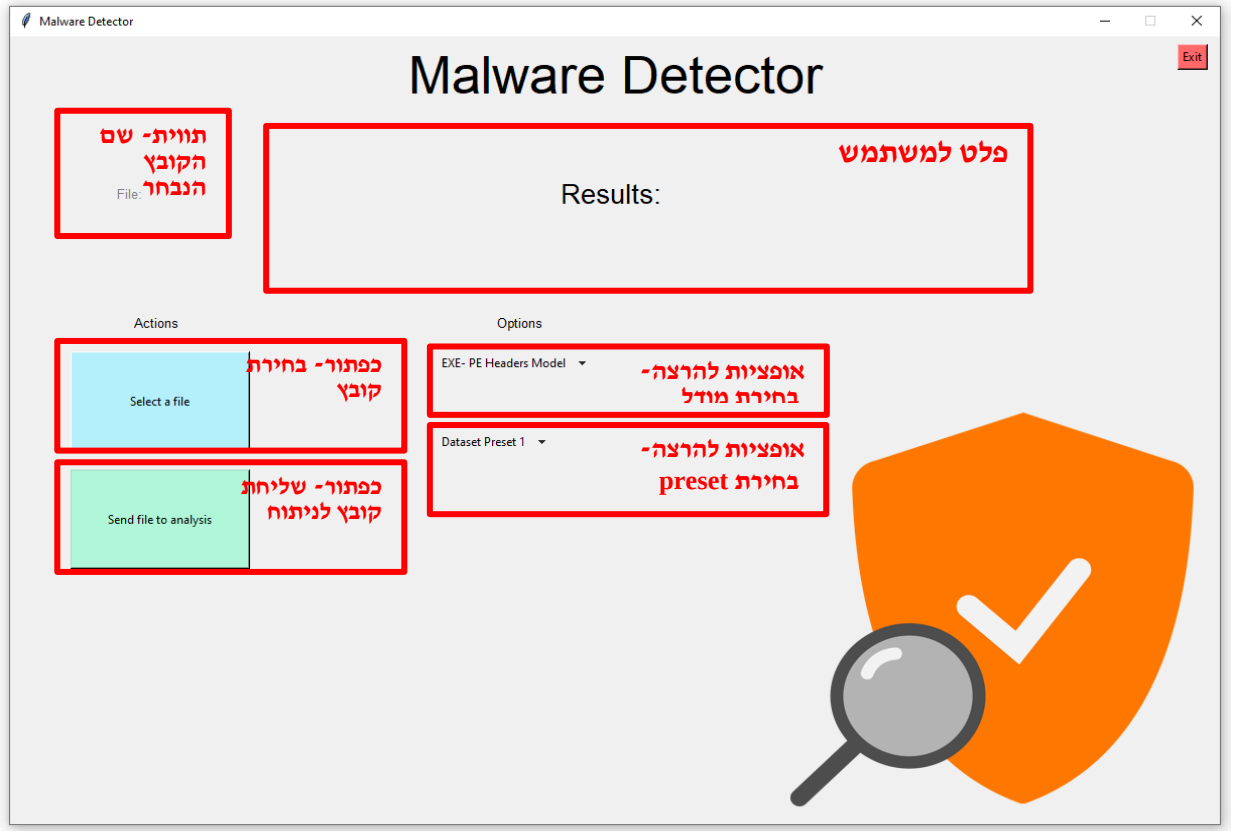
```
python main.py -p EVALUATOR
```

3. הרצה של הממשק הגראפי. הממשק מתועד בפרק מסכים – ממשק גרפי.
   זו גם אופציית ברירת המחדל.
   ניתן לבצע זאת ע"י שורת ההרצה הבאה:

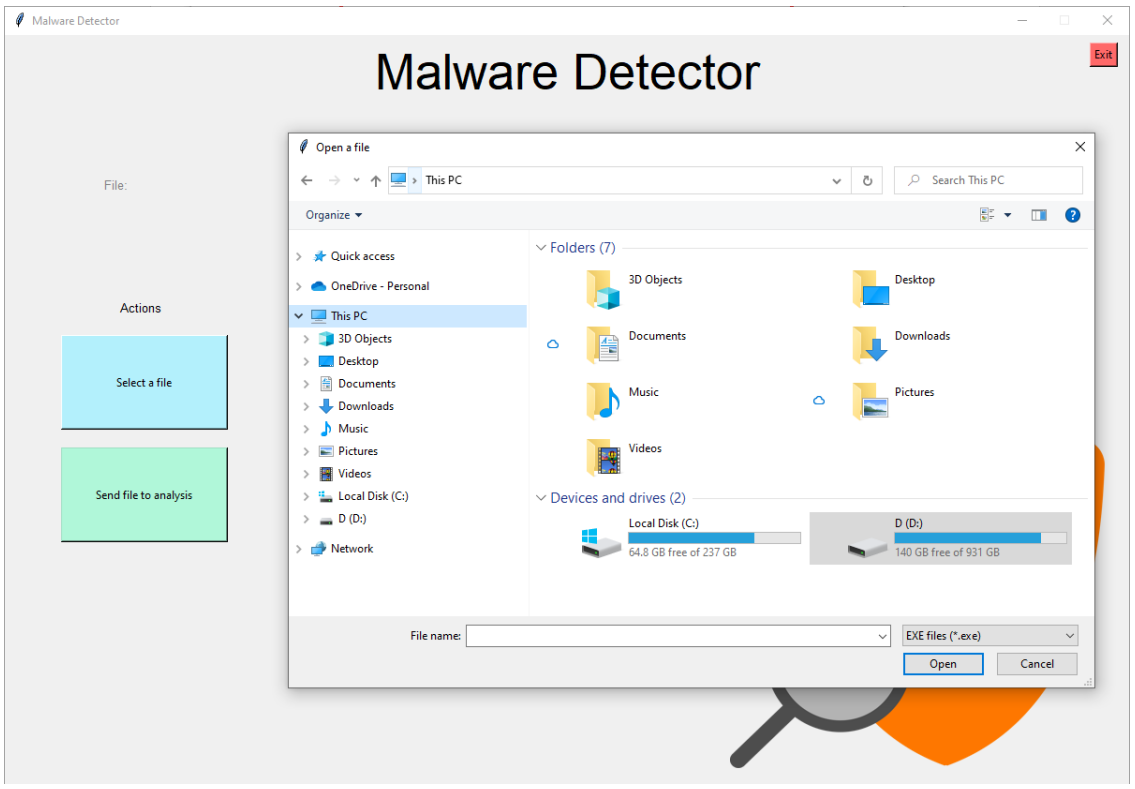
```
python main.py -p GUI
```

6.3 מסכים – ממשק גרפי

1. <u>מסך ראשי:</u>



2. <u>מסך בחירת קובץ:</u>
יש לבחור בקובץ מתאים באמצעות סייר הקבצים.



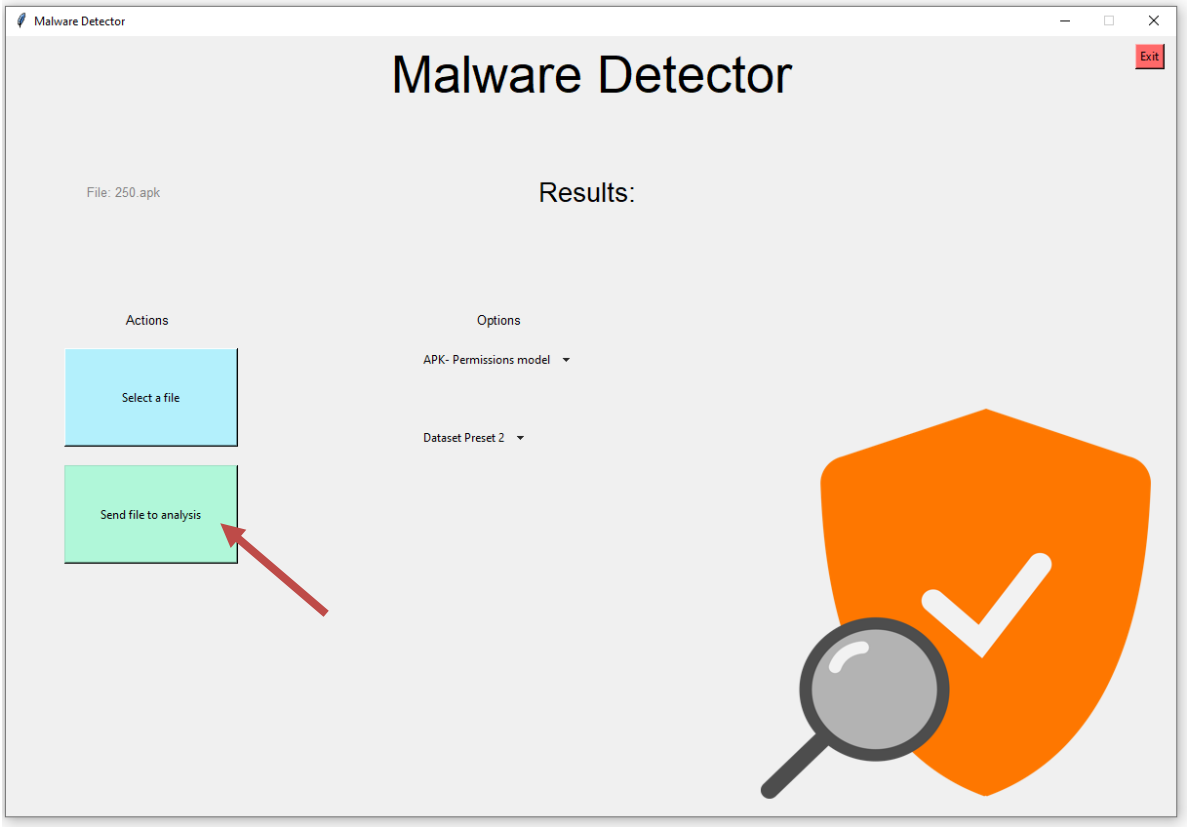
יש לבחור בסיומת הקובץ הרצוי (*EXE/APK*) ולאחר מכן לבחור קובץ.
לאחר בחירת קובץ, שם הקובץ יופיע בתווית המתאימה.

3. כעת יש לבחור מודל מתאים מבין המודלים הנתמכים בפרויקט:
   - אם מדובר בקובץ *EXE* אז במודל: *N-GRAM* או במודל *PE Headers*.
   - אם מדובר בקובץ *APK* יש לבחור במודל *APK-Permissions*

4. כעת יש לבחור ב *Preset* הרצוי:
   - *Preset 1* כולל את ה*Datasets*: *StormDroid_KuafuDet*, *Virushare*,
   - *Preset 2* כולל את ה*Datasets*: *Maldroid*, *Malica*,

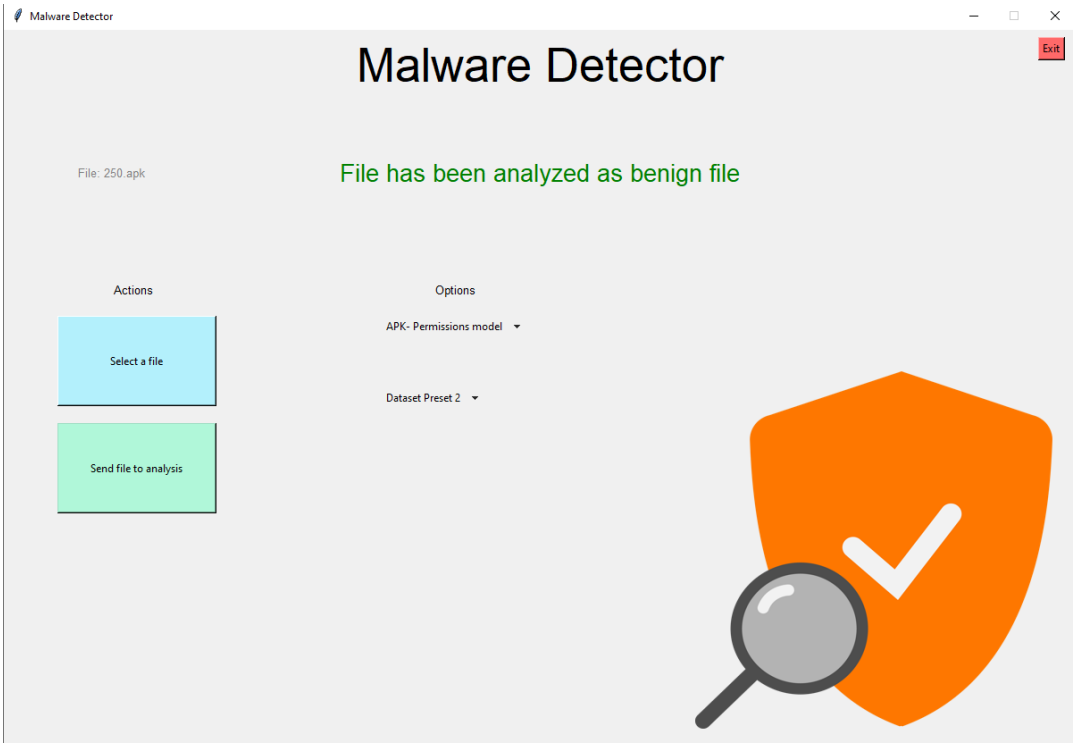
התוכנה תבצע חיזוי לפי ה-*Dataset* המתאים ב-*Preset* הנבחר.

5. כעת יש ללחוץ על כפתור *Send File to Analysis*

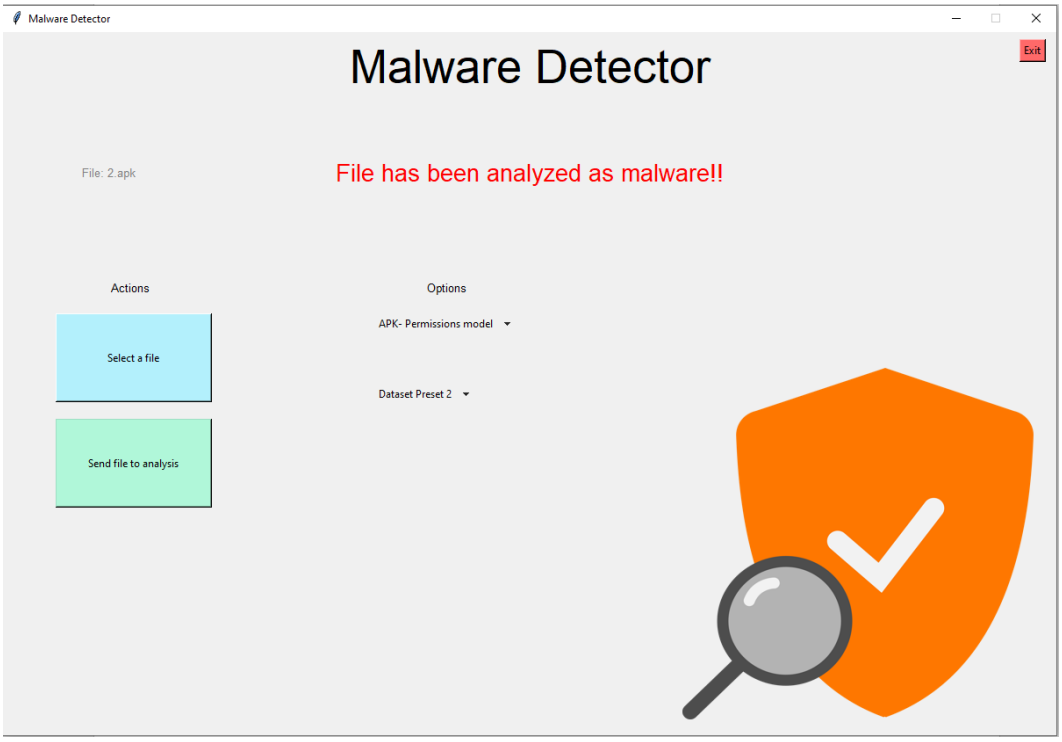


6. <u>תוצאת החיזוי תופיע בתווית הפלט:</u>

- **במקרה של קובץ שזוהה כ-תמים:**



- **במקרה של קובץ שזוהה כ-נוזקה:**



7. סיכום ומסקנות

7.1 סיכום ומסקנות הפרויקט

בפרויקט זה הוצג בקצרה רקע כללי בתחום המערכות לגילוי חדירות וזיהוי נוזקות.
נסקרו הגישות הנפוצות והמסורתיות למימוש מערכות שכאלו. ואת היתרונות והחסרונות של כל אחת מהשיטות הללו ואת הצורך במציאת פתרון אמין ויעיל יותר.
לאחר מכן, נסקרו שלוש שיטות חדשניות למימוש מנגנון גילוי חדירות וזיהוי נוזקות באמצעות אלגוריתמי למידת מכונה על בסיס מאפיינים סטטיים.
שתי השיטות הראשונות נועדו לזהות נוזקות למחשבים אישיים והשלישית ונועדה לזהות נוזקות למכשירים ניידים. הוצג הצורך המתגבר בהגנה על מכשירים ניידים ואת האילוצים השונים בזיהוי נוזקות המיועדות לפלטפורמות שונות.

בפרויקט מומשו שלוש שיטות שונות לזיהוי נוזקות, בוצעה השוואה בין יעילות מסווגי למידת מכונה שונים אל מול תוצאות מ2 מאגרי מידע (*Datasets*) שונים.
פרויקט זה הראה שניתן לבנות מערכת לזיהוי נוזקות חדשות שלא נותחו בעבר, על סמך מאפיינים סטטיים בלבד ובשימוש אלגוריתמי למידת מכונה וכן להגיע לאחוזי דיוק גבוהים מאד.

ברור לכל, כי מתקפות סייבר ימשיכו והתוקפים יהפכו מתוחכמים יותר ועל כן, מערכות הניטור וההגנה יאלצו לאמץ גישות חדשניות ולאתגר את הנחות היסוד, על מנת להתאים עצמן למציאות המשתנה.

7.2 אתגרים והמשך מחקר בתחום

בפרויקט זה ובתחום הזה ישנם מספר אתגרים שעמם נדרש לדעת להתמודד ושם נדרש להתמקד המשך המחקר בתחום.

האתגר הראשון והעיקרי נובע מהצורך לתחזק מאגרי מידע (*Datasets*) מעודכנים עם נוזקות חדשות שמתגלות יום יום. בפרויקט היה קושי רב להגיע למאגרי מידע גולמיים (המכילים קבצים בינאריים) אשר היו גדולים ורחבים מספיק והכילו קבצי נוזקות עדכניות.
הצורך הבסיסי ביותר בסיווג על סמך מסווגי למידת מכונה הוא הצורך במידע (*Data*) רב ועדכני ועל כן נדרש לעדכן את מאגרי המידע בצורה רציפה על מנת להבטיח אחוזי דיוק מספיק גבוהים.

אתגר נוסף הוא הצורך להתמודד עם נוזקות מסוג *Packers*, כלומר נוזקות מוצפנות או דחוסות אשר עיקר הקוד שלהן מפוענח ורץ אך ורק בזמן ריצת התוכנית. גישה המתבססת על מאפיינים סטטיים בלבד לוקה בחסר בכל הנוגע לנוזקות מסוג זה, שכן מסתמכת אך ורק על קובץ ההרצה.
על כן, ישנה חשיבות רבה לפיתוח מערכות לגילוי וזיהוי נוזקות המסתמכות על מאפיינים דינאמיים של קובץ ההרצה, בה הקובץ מורץ בסביבת סימולציה והתנהגותו מנוטרת בזמן אמת.

כמו כן, אתגר נוסף ואחת המגבלות של התוכנה אשר פותחה בפרויקט הוא הצורך לזהות נוזקות על סמך מידע/קובץ שהוא אינו קובץ ההרצאה. כלומר על סמך קבצי קונפיגורציה/ תעבורת תקשורת/ קבצים זמניים וכו'. זהו צורך משמעותי בתחום המחקר של זיהוי נוזקות שכן במקרים רבים נדרש לדעת לזהות תקיפה מבלי שיש קובץ הרצה 'חשוד' וזמין.

עבודה זו מהווה בסיס מעשי למערכות זיהוי נוזקות על בסיס מאפיינים סטטיים. ברור לכל כי במרוצת השנים הבאות הנוזקות יהפכו מתוחכמות יותר וימצאו דרכים חדשנות להתגבר על מנגנוני הזיהוי.
ועל כן יעד עתידי יהיה ליצור מנגנונים אשר ייתנו מענה זמן אמת, בצורה אפקטיבית ויעילה יותר עבור נוזקות מודרניות.

8. ביבליוגרפיה

8.1 מאמרים אקדמיים

[1] Hanqi Zhang, Xi Xiao, Francesco Mercaldo, Shiguang Ni, Fabio Martinelli, Arun Kumar Sangaiah:
Classification of ransomware families with machine learning based on N-gram of opcodes. Future Gener. Comput. Syst. 90: 211-221 (2019)

[2] Usukhbayar Baldangombo, Nyamjav Jambaljav, Shi-Jinn Horng:
A Static Malware Detection System Using Data Mining Methods. CoRR abs/1308.2831 (2013)

[3] Abdullah Talha Kabakus:
What Static Analysis Can Utmost Offer for Android Malware Detection. Inf. Technol. Control. 48(2): 235-240 (2019)

[4] Mehadi Hassen, Marco M. Carvalho, Philip K. Chan:
Malware classification using static analysis based features. SSCI 2017: 1-7

[5] Yousra Aafer, Wenliang Du, Heng Yin:
DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android. SecureComm 2013: 86-103

8.2 מקורות מידע נוספים

מקורות ה-*Datasets* הגולמיים:

https://sen-chen.github.io/kuafuDet/kuafuDet.html

http://205.174.165.80/CICDataset/MalDroid-2020/Dataset/APKs/

https://github.com/iosifache/DikeDataset