

שם בית הספר: תיכון הרצוג

שם העבודה: IP Grabber

שם התלמיד: עידן רטיק

תעודת זהות: 216239939

שם המנחה: אופיר שביט

שם החלופה: הגנת סייבר

תאריך ההגשה: 24/05/25

תוכן עניינים

[מבוא 4](#_Toc199021297)

[ייזום 4](#_Toc199021298)

[**תיאור ראשוני של המערכת** 4](#_Toc199021299)

[**הגדרת הלקוח** 4](#_Toc199021300)

[**הגדרת יעדים ומטרות** 4](#_Toc199021301)

[**סקירת טכנולוגיות בפרויקט** 4](#_Toc199021302)

[**סייגים והגבלות** 5](#_Toc199021303)

[**תיחום הפרוייקט** 5](#_Toc199021304)

[סקירת פתרונות קיימים 5](#_Toc199021305)

[**Grabify** 5](#_Toc199021306)

[**IPLogger** 7](#_Toc199021307)

[**IP Tracker online** 8](#_Toc199021308)

[אפיון המערכת 9](#_Toc199021309)

[**יכולות ההורה** 9](#_Toc199021310)

[**פירוט בדיקות** 10](#_Toc199021311)

[**לו״ז** 12](#_Toc199021312)

[פירוט יכולות 12](#_Toc199021313)

[יכולות בצד הלקוח 12](#_Toc199021314)

[יכולות בצד השרת 14](#_Toc199021315)

[רקע תיאורטי 15](#_Toc199021316)

[מונחים מתקדמים ברשתות 15](#_Toc199021317)

[**ARP spoofing** 15](#_Toc199021318)

[**איש באמצע – MITM** 15](#_Toc199021319)

[**הרעלת DNS** 16](#_Toc199021320)

[הטכנולוגיה הרלוונטית 16](#_Toc199021321)

[**שפות תכנות** 16](#_Toc199021322)

[**מערכות הפעלה** 16](#_Toc199021323)

[**תקשורת** 16](#_Toc199021324)

[מבנה וארכיטקטורה של הפרוייקט 17](#_Toc199021325)

[תיאור הארכיטקטורה של המערכת 17](#_Toc199021326)

[**תיאור החומרה** 17](#_Toc199021327)

[**החומרה בשרטוט** 17](#_Toc199021328)

[זרימת מידע 18](#_Toc199021329)

[**כללי** 18](#_Toc199021330)

[**תת תהליך MITM + הרעלת DNS.** 20](#_Toc199021331)

[אלגוריתמים מרכזיים בפרויקט 22](#_Toc199021332)

[**תקשורת בין תתי תהליכים** 22](#_Toc199021333)

[סביבת הפיתוח 24](#_Toc199021334)

[**כלי הפיתוח הדרושים** 24](#_Toc199021335)

[תחום 24](#_Toc199021336)

[כלים / טכנולוגיה שתעשה בהם שימוש 24](#_Toc199021337)

[**סביבה וכלים הדרושים לבדיקות** 25](#_Toc199021338)

[תיאור הפרוטוקול 25](#_Toc199021339)

[**הסבר מילולי** 25](#_Toc199021340)

[**פירוט** 25](#_Toc199021341)

[**מימוש** 26](#_Toc199021342)

[תיאור מסכי המערכת 27](#_Toc199021343)

[**מסך התחברות** 27](#_Toc199021344)

[**מסך הרשמה** 28](#_Toc199021345)

[**תפריט ראשי** 29](#_Toc199021346)

[**הוספת קישור** 30](#_Toc199021347)

[**מחיקת קישור** 31](#_Toc199021348)

[**מציאת כתובת אמיתית** 32](#_Toc199021349)

[**בקשת מידע** 33](#_Toc199021350)

[**תרשים זרימה מסכים** 36](#_Toc199021351)

[תיאור מסדי הנתונים 36](#_Toc199021352)

[**משתמשים** 36](#_Toc199021353)

[**קישורים** 37](#_Toc199021354)

[**כניסות** 37](#_Toc199021355)

[סקירת חולשות ואיומים 37](#_Toc199021356)

[מימוש הפרויקט 37](#_Toc199021357)

[סקירת מודולים 37](#_Toc199021358)

[**מודולים מיובאים** 37](#_Toc199021359)

[**מודולים בפיתוח עצמי** 38](#_Toc199021360)

[קטעי קוד ראויים לציון 47](#_Toc199021361)

[**תקשורת בין תהליכים** 47](#_Toc199021362)

[**שימוש בwrapper לסוקט** 48](#_Toc199021363)

[מסמך בדיקות מלא 49](#_Toc199021364)

[מדריך למשתמש 50](#_Toc199021365)

[עץ קבצים 50](#_Toc199021366)

[התקנת המערכת 51](#_Toc199021367)

[**סביבה וכלים נדרשים** 51](#_Toc199021368)

[**אתחול נתונים** 52](#_Toc199021369)

[אופן הפעלת המערכת 52](#_Toc199021370)

[**שרת** 52](#_Toc199021371)

[**לקוח** 52](#_Toc199021372)

[רפלקציה 53](#_Toc199021373)

[ביבליוגרפיה 54](#_Toc199021374)

[נספחים 54](#_Toc199021375)

# **מבוא**

## **ייזום**

### **תיאור ראשוני של המערכת**

אני מפתח כלי IP Grabber שמאפשר לאסוף ולאחסן כתובות IP של משתמשים שמבקרים בדף אינטרנט מסוים. הכלי יאפשר גם איסוף מידע נוסף כמו פרטי מערכת הפעלה, דפדפן, ומיקום גיאוגרפי משוער. המוצר המוגמר יהיה פלטפורמה שמאפשרת יצירה ושימוש ב-IP Grabber באמצעות ממשק פשוט ונוח למשתמש.

בחרתי בפרויקט זה כדי להעמיק את הידע שלי באבטחת תקשורת, בטכניקות מתקדמות ברשתות, ופיתוח. הפרויקט משלב היבטים של רשתות, מערכות הפעלה, פיתוח צד-שרת וצד-לקוח, וניתוח נתונים.

רוב הקשיים שאני צופה בהכנה של המערכת יהיו קשורים, לטכניקות מתקדמות ברשתות כגון ARP spoofing, וDNS spoofing, בתקשורת בין פרוססים

### **הגדרת הלקוח**

מיועדת להורים, שמעוניינים להשגיח על גישת ילדיהם לאתרים מסוימים. בעזרת שימוש במוצר, יוכלו ההורים לבדוק

### **הגדרת יעדים ומטרות**

* יצירת כלי פשוט וידידותי למשתמש לאיסוף כתובות IP, ללא פעולה אקטיבית של המתחברים לאתר.
* אפשרות לאחסון והצגת הכתובות שנאספו בממשק מאורגן
* יצירת גרפים וניתוחים סטטיסטיים של הנתונים שנאספו
* אפשרות לייצא את הנתונים בפורמטים שונים

חסכונות:

חיסכון בזמן באיתור והתמודדות עם גישות חשודות

ייעול תהליכי ניטור אבטחה

הפחתת הצורך בכלים יקרים יותר לניטור תעבורה

### **סקירת טכנולוגיות בפרויקט**

צד לקוח: HTML, FLASK, CSS, SOCKET/TCP

צד שרת: HTTP,SCAPY, DNS, ARP, Multiprocessing ,Signal ,socket/TCP

נתונים: Json

### **סייגים והגבלות**

על המשתמש להכניס בצורה מפורשת את הURL שדרכו מתבצע המעקב.

חלק מהטכנולוגיות לא יעבדו כראוי אם יש רשת עם הרבה הגנות, או אם משתמשים בדפדפן זהיר.

### **תיחום הפרוייקט**

הפרויקט עוסק במספר תחומים הקשורים לרשתות ומערכות הפעלה -

רשתות:

תקשורת בין לקוח לשרת בעזרת SOCKET

שימוש בפרוטוקול ARP כדי לנתב את בקשות הDNS אל השרת.

שימוש בפרוטוקול DNS כדי לגרום לURL שלא קיימים לעבוד.

שימוש בScapy כדי לשלוח פקטות.

שימוש בFLASK למימוש שרת http בשביל gui בסיסי

שימוש בפרוטוקול http כדי לבצע redirect.

מערכות הפעלה:

שימוש בmultiprocessing בצד השרת.

שימוש בPIPE כדי לדבר בין הפרוססים השונים (IPC)  
שימוש במערכת קבצים לשמירת מידע.

שימוש בsignal כדי לצאת מprocess בצורה נקייה.

הפרויקט יתמקד באיסוף מידע באופן פסיבי בלבד (כלומר, מידע שנשלח כחלק מהבקשות הרגילות) ולא יכלול איסוף אקטיבי או חדירה למערכות.

בנוסף, הפרוייקט יעבוד אך ורק בין מכשירים שנמצאים באותה הרשת. לא קיימת בפרוייקט התעסקות או התייחסות לNAT, port forwarding, וכדו׳.

## **סקירת פתרונות קיימים**

### **Grabify**

**סקירה כללית:**

Grabify הוא כלי מבוסס-ווב המאפשר יצירת קישורים מקוצרים לצורכי מעקב אחר מידע טכני של גולשים, בדגש על איתור כתובת ה-IP, מידע על הדפדפן, מערכת ההפעלה, המדינה, ספק האינטרנט ועוד. המשתמש יוצר קישור דרך הפלטפורמה, משתף אותו עם אחרים, וכל גישה לקישור מתועדת.

לאחר שמזינים את הURL שאותו רוצים להסוות, מקבלים קוד שבעזרתו ניגשים אל הנתונים לאחר מכן.

**יתרונות:**

פשטות שימוש - ממשק אינטואיטיבי וידידותי למשתמש, גם ללא ידע טכני.

גישה מיידית למידע - ניתן לצפות במידע שנאסף מיידית לאחר לחיצה על הקישור.

תוספים נלווים - כולל אפשרויות להצגת עמודי הפניה מזויפים (fake redirect) שגורמים למשתמש לחשוב שמדובר באתר רגיל.

מערכת קיצור כתובות - המערכת מקצרת כתובות באופן אוטומטי ומסתירה את מטרת הקישור האמיתית.

**חסרונות:**

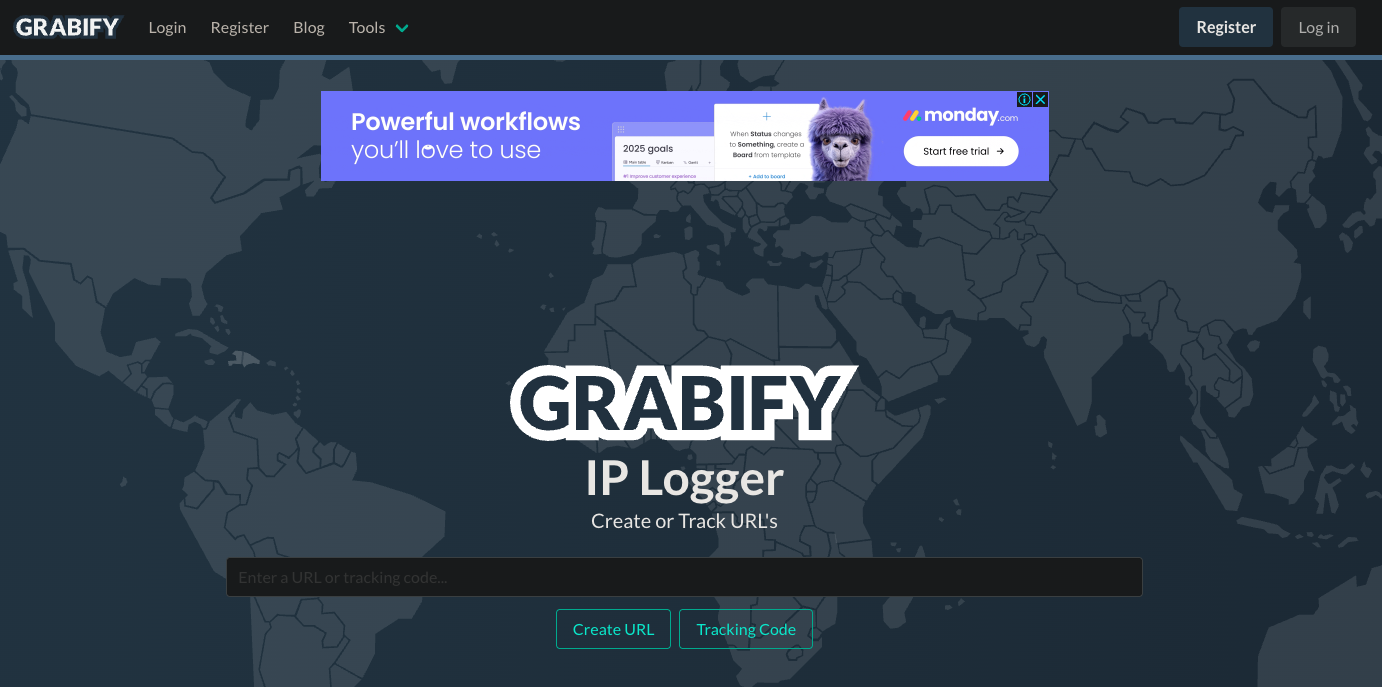
פונקציונליות מוגבלת - הכלי אוסף רק נתונים בסיסיים ואינו מספק כלי ניתוח מתקדמים או תיעוד היסטורי מתוחכם.

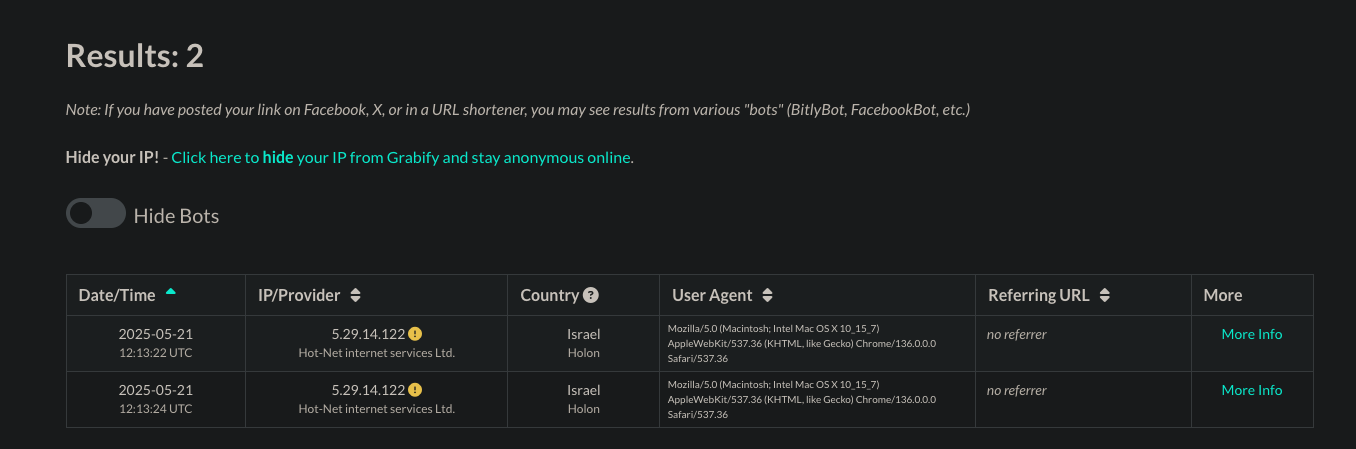
קל לזיהוי - מנגנוני אבטחה (כגון אנטי וירוס או דפדפנים) לעיתים מזהים את הקישורים כחשודים.

חוסר בהתאמה אישית -אין אפשרות ליצור דף נחיתה מותאם אישית או ממשק API מתקדם.

**אופן הפעולה**: כאשר נוצר קישור, Grabify עוקב אחרי כל מי שלוחץ עליו, שומר את כתובת ה-IP ומידע נוסף. ניתן להוסיף הפניות לדפים אחרים כך שהמשתמש לא ישים לב שהוא במעקב.

**ייחודיות:** מיועד בעיקר לשימוש אישי ולא לארגונים. הדגש הוא על קלות ופשטות ולא על עושר בפונקציות.

. 

****

### **IPLogger**

**סקירה כללית:**  
IPLogger הוא שירות ותיק יותר מ-Grabify, והוא מציע קישור למעקב אחרי גולשים עם פיצ'רים מתקדמים יותר. הכלי מציע אפשרות ליצירת כתובות מסוגים שונים, כגון תמונה מוסתרת (invisible image), פיקסל בדוא"ל, או דף הפניה מותאם.

**יתרונות:**

מגוון אפשרויות מעקב - ניתן לבחור בין קישור רגיל, תמונה שקופה, כפתור, או קובץ להורדה.

שירותי API - מאפשר אינטגרציה עם מערכות קיימות או כתיבת סקריפטים מותאמים אישית.

נתוני גיאולוקציה מדויקים - מזהה מדינה, עיר, דפדפן, רזולוציית מסך, ונתוני HTTP נוספים.

מעקב בזמן אמת - תיעוד כל גישה כולל מיקום, שעה, ומידע על המכשיר.

**חסרונות:**

ממשק מיושן - הממשק נראה ישן ומעט עמוס.

יותר מדי פרסומות - השירות החינמי כולל כמות גדולה של פרסומות, מה שפוגע בחוויית השימוש.

ללא מנגנוני הסוואה מתקדמים - קישור שנוצר לא תמיד מוסתר בצורה שמונעת חשד.

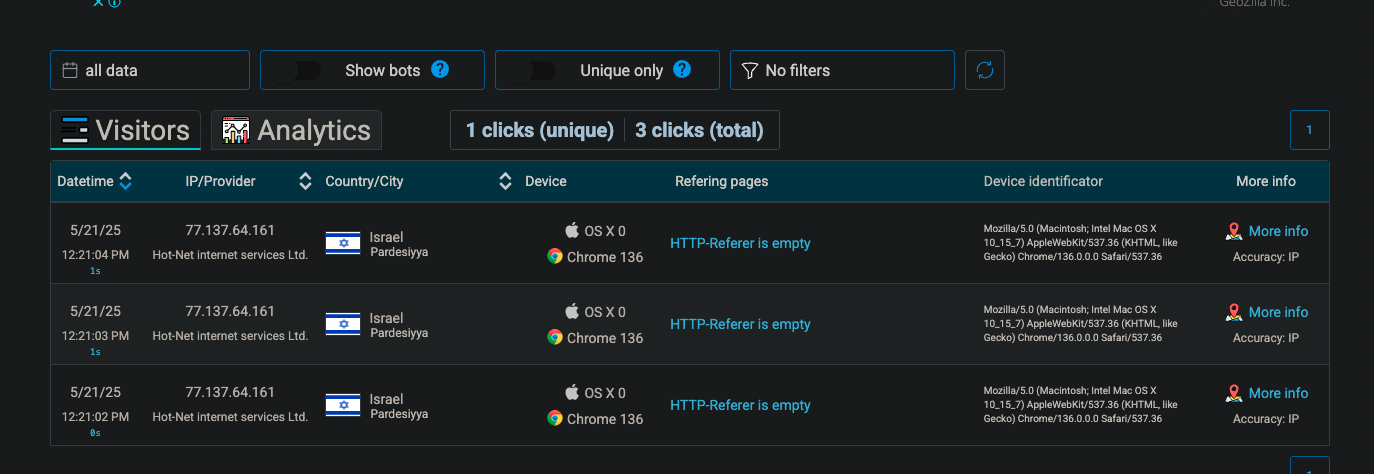
**אופן הפעולה**:

IPLogger מספק כמה דרכים "להחביא" את כלי המעקב – למשל, על ידי החדרת תמונה בלתי נראית באימיילים או דפי אינטרנט. כל כניסה או טעינה של הקובץ גורמת לתיעוד מלא של המשתמש.

**ייחודיות:**

פונה גם למשתמשים מתקדמים – כולל API, פיצ’רים מותאמים אישית, ויכולות מעקב מורחבות. מתאים לשילוב בפרויקטים גדולים יותר.

****

****

### **IP Tracker online**

**סקירה כללית:**  
IP Tracker Online הוא כלי פשוט אך זמין, המיועד לאנשים שרוצים לבדוק כתובת IP ספציפית או לבצע מעקב אחרי קישור אחד, לרוב ללא התחברות לחשבון. הוא אינו כלי קיצור כתובות אלא יותר שירות לבדיקת כתובות IP קיימות..

**יתרונות:**

כלי חינמי לגמרי - ללא צורך ברישום או תשלום – כל הכלים פתוחים.

בדיקת IP קיימת - בניגוד לשני הכלים האחרים, ניתן להזין כתובת IP ידנית ולבדוק עליה פרטים.

**חסרונות:**

**פונקציונליות בסיסית בלבד:** - אין אפשרות ליצור קישורים למעקב – אלא רק לבדוק IP ידוע מראש.

אין API – לא נועד לשימוש אוטומטי, או מקצועי.

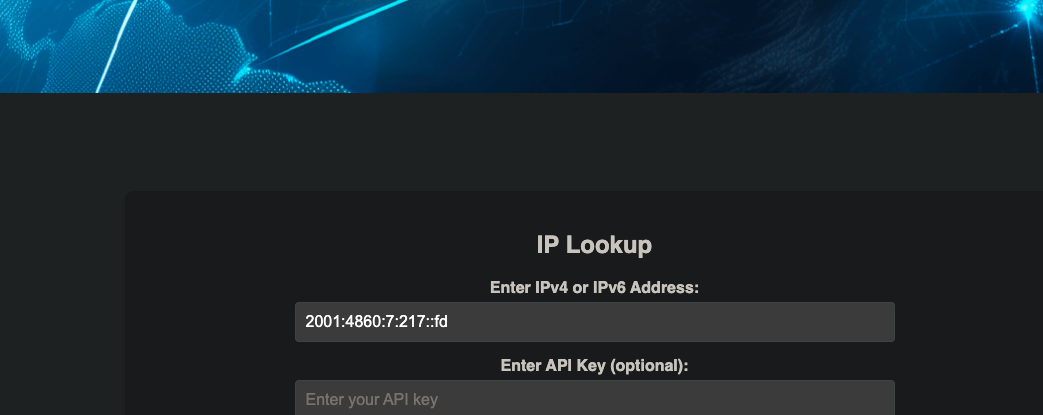
אין תיעוד – כל בדיקה היא חד פעמית, ולא נשמרת.

**אופן הפעולה**:

המשתמש מזין כתובת IP או שם דומיין, והמערכת מפיקה מידע הכולל מיקום גאוגרפי, ספק אינטרנט, ולעתים גם מעלה חשד אם מדובר בproxy או בVPN.

**ייחודיות:**

פועל כמזהה IP ולא ככלי מעקב.



## **אפיון המערכת**

המערכת משמשת בתור כלי של ההורה, כדי לנטר ולבדוק מתי כל ילד התחבר לאתרים מסוימים.

השרת יחזיק אצלו בצורה לא מוצפנת מאגר של קישורים. המאגר יקשר כל קישור מומצא לקישור אמיתי.

ההורה יתחבר אל המערכת ושם יוצגו לו האופציות, אם להתחבר בתור משתמש רשום או להירשם.

לאחר שיתחבר בתור משתמש, יוכל לנהל בצורה דינמית את מאגר הקישורים שמוחזק בצד הלקוח.

כאשר מישהו ישתמש בקישור המומצא, המערכת תתעד את זמן הכניסה, כתובת ה-IP ממנה בוצעה הגישה, ופרטים נוספים. בנוסף, המערכת תנתב את הילד לקישור האמיתי מבלי שירגיש שהוא מנוטר.

כאשר יתחבר ההורה אל המערכת, תהיה לו אופציה לעבור לעמוד שבו יוכל לראות את כל הכניסות שבוצעו, למיין אותם לפי שעה, לפי מחשב, ועוד. ההורה יוכל לראות גרפים וסטטיסטיקות על תדירות השימוש, שעות הגלישה המועדפות, והתפלגות הגלישה בין המשתמשים השונים, ההורה יכול לייצא את הנתונים הללו לאקסל.

מערכת זו לא תדרוש שום דבר מהילד, חוץ מלהכניס את הקישור המזויף שניתן לו.

### **יכולות ההורה**

* ליצור חשבון.
* להתחבר לחשבון.
* לבקש את הקישור האמיתי, של קישור מסוים, מהמאגר.
* להכניס אל המאגר קישור חדש.
* להוציא מהמאגר קישור קיים.
* לבקש את כל הכניסות לאתר מסוים.
* לעבור בין התפריטים.

### **פירוט בדיקות**

|  |  |  |  |
| --- | --- | --- | --- |
| מס' | יכולת לבדיקה | תיאור הבדיקה | איך בודקים? |
| 1 | יצירת משתמש (הורה) | בדיקת תהליך הרשמה של משתמש חדש למערכת | מריצים את אפליקציית הממשק, נכנסים לדף ההרשמה וממלאים את הטופס עם פרטים תקינים. מוודאים שהמשתמש נוצר במסד הנתונים ושאפשר להתחבר עם הפרטים החדשים. יש לבדוק גם מקרי קצה: הכנסת פרטים לא תקינים, שם משתמש קיים, וסיסמה חלשה. |
| 2 | התחברות משתמש | בדיקת התחברות משתמש קיים | יוצרים משתמש מראש, מתנתקים מהמערכת ומנסים להתחבר מחדש עם שם המשתמש והסיסמה. בודקים שהמערכת מזהה נכון את פרטי ההתחברות ומציגה את ממשק הניהול. יש לבדוק גם התחברות עם פרטים שגויים. |
| 3 | יצירת קישור מזויף | בדיקת יצירת קישור מזויף וקישורו לאתר אמיתי | נכנסים לממשק ניהול הקישורים, יוצרים קישור חדש עם שם ייחודי והאתר האמיתי (למשל google.com). בודקים שהקישור נוצר במסד הנתונים ומוצג ברשימת הקישורים בממשק. |
| 4 | ARP Spoofing | בדיקת יכולת התחזות לנתב | מפעילים את מנגנון ה-ARP Spoofing כאשר Wireshark פועל על מחשב אחר ברשת. בודקים ב-Wireshark שחבילות ה-ARP המזויפות נשלחות לרשת, ושהמחשבים ברשת מעדכנים את טבלאות ה-ARP שלהם להצביע על ה-MAC של השרת שלנו במקום זה של הנתב. |
| 5 | יירוט בקשות DNS | בדיקת זיהוי ויירוט בקשות DNS לדומיינים מזויפים | מפעילים את מנגנון יירוט ה-DNS ואת Wireshark. מהמחשב של משתמש הקצה, מנסים לגשת לדומיין מזויף שיצרנו. בודקים ב-Wireshark שהשרת שלנו מספק תשובת DNS מזויפת המפנה ל-127.0.0.1 או לכתובת IP של השרת שלנו. |
| 6 | ניהול HTTP Server | בדיקת שרת ה-HTTP שמקבל את הפניות | מפעילים את שרת ה-HTTP ומוודאים שהוא עולה ללא שגיאות. מבצעים בקשות HTTP מפוזרות לשרת וצופים בלוגים כדי לוודא שהשרת מקבל אותן כראוי. בודקים את זמני התגובה תחת עומס על ידי יצירת מספר בקשות במקביל. |
| 7 | הפניה לאתר אמיתי | בדיקת הפניית משתמש מהקישור המזויף לאמיתי | לאחר הפעלת כל המנגנונים, ניגשים לקישור מזויף מדפדפן משתמש הקצה. בודקים שהדפדפן מופנה מיד לאתר האמיתי, וצופים ב-Developer Tools של הדפדפן כדי לוודא שמתקבלת הפניית 302 תקינה. |
| 8 | תיעוד כניסות | בדיקת תיעוד מידע על כניסות לקישורים מזויפים | מבצעים גישה לקישור מזויף ממספר מכשירים, דפדפנים ורשתות שונות. לאחר מכן, בודקים במסד הנתונים שכל הכניסות תועדו עם הפרטים המתאימים: כתובת IP, זמן הכניסה, סוג הדפדפן והמכשיר. |
| 9 | הצגת נתונים בממשק הורה | בדיקת הצגת סטטיסטיקות ונתוני כניסה | לאחר צבירת מספר כניסות במערכת, נכנסים לממשק ההורה ובודקים שהנתונים מוצגים נכון בגרפים ובטבלאות. בודקים את העדכון בזמן אמת על ידי פתיחת הממשק במקביל לביצוע כניסות חדשות. |
| 10 | תמיכה בריבוי משתמשים | בדיקת תמיכה במספר הורים ומספר ילדים | יוצרים מספר חשבונות משתמשים (הורים) ולכל אחד מהם מספר פרופילים של ילדים. בודקים שכל משתמש רואה רק את הקישורים והנתונים השייכים אליו, ולא את אלו של משתמשים אחרים. |
| 11 | ביצועי MITM | בדיקת יכולת Man-in-the-Middle | מפעילים את מנגנון ה-MITM ובמקביל מבצעים פעולות גלישה שונות ברשת. בודקים בעזרת Wireshark שהתעבורה עוברת דרך השרת שלנו, ושאין עיכובים משמעותיים או שגיאות תקשורת. |
| 12 | זיהוי DNS המזויף | בדיקת יכולת זיהוי דומיין מזויף | מפעילים את מנגנון יירוט ה-DNS ושולחים בקשות DNS הן לדומיינים מזויפים והן לדומיינים לגיטימיים. בודקים שהמערכת מזהה ומטפלת רק בדומיינים המזויפים ולא מפריעה לתעבורה הרגילה. |
| 14 | מולטי-פרוססינג | בדיקת פעולת המערכת במספר תהליכים במקביל | מפעילים את כל רכיבי המערכת במקביל (ARP Spoofing, יירוט DNS, שרת HTTP, ממשק ניהול). בודקים את צריכת המשאבים של המערכת ומוודאים שכל התהליכים מתקשרים ביניהם כראוי. מבצעים פעולות במקביל ובודקים שאין התנגשויות או חסימות. |
| 15 | עמידות בתקלות | בדיקת התנהגות המערכת במקרה של תקלות | מדמים מצבי תקלה שונים: ניתוק רשת פתאומי, כיבוי אחד התהליכים, ניסיון גישה לדומיין לא קיים. בודקים שהמערכת מתאוששת או מטפלת בתקלות בצורה הולמת ולא קורסת. |

### **לו״ז**

|  |  |
| --- | --- |
| **תאריך** | **משימה** |
| 31.11 | אפיון בסיסי של הפרוייקט |
| 1.1 | הוכחות יכולות בסיסיות (ARP spoofing, dns poisoning) |
| 3.1 | מימוש תקשורת שרת לקוח עם ממשק טקטסואלי |
| 31.3 | חיבור התקשורת עם כל היכולות |
| 10.4 | הכנת ממשק גרפי |
| 31.4 | סיום הכנת קוד כולל בדיקת באגים |

# 

# **פירוט יכולות**

## **יכולות בצד הלקוח**

|  |  |  |  |
| --- | --- | --- | --- |
| **שם** | **מהות** | **יכולות הדרושות למימוש היכולת** | **אובייקטים נחוצים** |
| יצירת משתמש | שליחת לשרת הודעה מוצפנת של יצירת משתמש עם הפרטים הנדרשים. | שליחת הודעה לשרת, בניית הודעה לפי פרוטוקול, קבלת קלט משתמש | תקשורת מוצפנת עם השרת, סביבת משתמש גרפית, מאגר משתמשים, טיפול בשגיאות שהשרת מחזיר. |
| התחברות למשתמש | שליחת הודעה מוצפנת לשרת של התחברות למשתמש. | שליחת הודעה לשרת, בניית הודעה לפי פרוטוקול, קבלת קלט משתמש | תקשורת מוצפנת עם השרת, סביבת משתמש גרפית, מאגר משתמשים, טיפול בשגיאות שהשרת מחזיר. |
| בקשת מידע מהשרת | שליחת לשרת הודעה מוצפנת של לפי הפרוטוקול | שליחת הודעה לשרת, בניית הודעה לפי פרוטוקול, קבלת קלט משתמש | תקשורת מוצפנת עם השרת, סביבת משתמש גרפית, מאגר משתמשים, טיפול בשגיאות שהשרת מחזיר, מאגר קישורים. |
| הוספה של קישור למאגר | שליחת לשרת הודעה מוצפנת של לפי הפרוטוקול | שליחת הודעה לשרת, בניית הודעה לפי פרוטוקול, קבלת קלט משתמש | תקשורת מוצפנת עם השרת, סביבת משתמש גרפית, מאגר משתמשים, טיפול בשגיאות שהשרת מחזיר, מאגר קישורים. |
| מחיקת קישור מהמאגר | שליחת לשרת הודעה מוצפנת של לפי הפרוטוקול | שליחת הודעה לשרת, בניית הודעה לפי פרוטוקול, קבלת קלט משתמש | תקשורת מוצפנת עם השרת, סביבת משתמש גרפית, מאגר משתמשים, טיפול בשגיאות שהשרת מחזיר, מאגר קישורים. |
| בקשה של פירוט התחברויות לאתר מסוים | שליחת לשרת הודעה מוצפנת של לפי הפרוטוקול | שליחת הודעה לשרת, בניית הודעה לפי פרוטוקול, קבלת קלט משתמש | תקשורת מוצפנת עם השרת, סביבת משתמש גרפית, מאגר משתמשים, טיפול בשגיאות שהשרת מחזיר, מאגר קישורים. |
| בקשת הקישור האמיתי התואם לקישור מזויף מסוים. | שליחת לשרת הודעה מוצפנת של לפי הפרוטוקול | שליחת הודעה לשרת, בניית הודעה לפי פרוטוקול, קבלת קלט משתמש | תקשורת מוצפנת עם השרת, סביבת משתמש גרפית, מאגר משתמשים, טיפול בשגיאות שהשרת מחזיר, מאגר קישורים. |
| החלפת מפתחות | החלפת מפתחות AES בעזרת RSA | תקשורת בין שרת ללקוח | עוטף לסוקט, מפתח RSA |
| הצפנה ופענוח מידע | הצפנה ופענוח מידע שנשלח או מגיע מהשרת בעזרת AES | תקשורת בין שרת ללקוח, החלפת מפתחות | עוטף לסוקט, מפתח AES |
| שליחת הודעה מוצפנת לשרת | שילוב בין יכולת הצפנת המידע לבין יכולת שליחת הודעה לשרת. | שליחת הודעה לשרת, הצפנת מידע | עוטף לסוקט, מפתח AES |
| בניית הודעה לפי פרוטוקול | בניית הודעה לפי הפרוטקול, בהתאם לקלט המשתמש. |  | עוטף לסוקט. |
| שליחת הודעה לשרת |  | יצירת תקשורת עם השרת | עוטף לסוקט. |
| יצירת תקשורת עם השרת. | Connect |  | עוטף לסוקט. |
| קבלת קלט משתמש | שימוש בממשק גרפי כדי לקבל ולנתח קלט משתמש. | הצגה ושימוש בממשק גרפי | ממשק גרפי. |
| הצגה ושימוש בממשק גרפי. | הצגת כל התפריטים, מעבר בינהם, והזנת נתונים. |  | Flask |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## **יכולות בצד השרת**

|  |  |  |  |
| --- | --- | --- | --- |
| **שם** | **מהות** | **יכולות הדרושות למימוש היכולת** | **אובייקטים נחוצים** |
| יצירת משתמש חדש | רישום משתמש חדש לצורך שימוש בפלטפורמה | ממשק משתמש, אימות קלט, כתיבה למסד נתונים | טופס רישום, מסד נתונים |
| בדיקת התחברות | אימות זהות משתמש קיים | אימות מול מסד נתונים, השוואת סיסמה, | טופס התחברות, מסד נתונים |
| הוספת קישור למאגר קישורים | יצירת URL שמוביל לסקריפט לכידת IP | יצירת קישור ייחודי, אחסון במסד נתונים | מסד נתונים, ממשק לניהול קישורים |
| מחיקת קישור מהמאגר | הסרת URL שכבר לא בשימוש | שליפה ומחיקה מהמסד | ממשק ניהול, מסד נתונים |
| מציאת קישור תואם | חיפוש קישור לפי מזהה או תבנית | חיפוש טקסטואלי או לפי מזהה ייחודי | מסד נתונים |
| שליפת מידע על ההתחברויות | הצגת היסטוריית כניסות לכל קישור | שליפה מהמסד, סינון לפי מזהה קישור | מסד התחברויות |
| שליחת מידע מוצפן | תקשורת באמצעות AES | אלגוריתם הצפנה (כגון AES), ניהול מפתחות | ספריית קריפטוגרפיה, מפתח הצפנה |
| החלפת מפתחות | החלפת מפתחות AES באמצעות RSA | RSA, שליחת מידע | מודול הצפנה |
| שליחת מידע | שליחת מידע ללקוח. |  | עוטף לסוקט. |
| ניהול שרת HTTP | הפעלת שרת שמאזין לבקשות מהקורבן | פתיחת תת תהליך, סגירת תת תהליך |  |
| ביצוע ARP spoofing | התחזות ל-Gateway כדי ליירט חבילות (מתקפה מקומית בלבד) | פתיחת תת תהליך, סגירת תת תהליך | סקאפי |
| מימוש איש באמצע (MITM) | יירוט תקשורת בין לקוח לשרת | פתיחת תת תהליך, סגירת תת תהליך, הסנפה ושליחה של פקטות. | עוטף לסוקט, סקאפי |
| החזרת תשובת DNS פיקטיבית | שליחת תשובת DNS מזויפת כדי להפנות לשרת HTTP פנימי. | MITM | עוטף לסוקט, סקאפי |
| פתיחת תת-תהליך | הפעלת כלי עזר במקביל (למשל spoofing או sniffer) | ניהול תהליכים במערכת (multiprocessing / subprocess) | גישת מערכת |
| סגירת תת-תהליך | עצירת רכיב פועל במערכת | מעקב אחרי PID, שליחת סיגנל סיום | מזהי תהליך (PID), בקר תהליכים |

# 

# **רקע תיאורטי**

## **מונחים מתקדמים ברשתות**

***הבהרה: משתמשים בטכנולוגיות אלו גם (ובעיקר) בהקשרים זדוניים. בפרויקט אני משתמש בהם במטרה שאינה מטרה זדונית, והמימוש המדויק שקיים בפרויקט הוא לא מימוש שניתן להשתמש בו בהקשרים זדוניים.***

### **ARP spoofing**

שיטה לשנות בעבור מחשב מסוים את שיוך בין IP לבין MAC, ולמעשה מעניקה היכולת ׳להתחזות׳ למחשב אחר.

באופן טכני, כדי לבצע את השינוי, צריך לשלוח באופן נשנה תשובת ARP שעונה על השאלה ״מי זה המחשב עם הIP הזה״ עם התשובה ״לי״.

בפרויקט אנחנו משתמשים בשיטה זו, כדי שהמחשב של הילד יחשוב שהשרת הוא הrouter.

אנחנו שולחים מספר אקראי של פעמים בשנייה תשובת ARP כמתואר.

### **איש באמצע – MITM**

ופעה שבה צד שלישי מצליח להיכנס לאמצע של תקשורת בין שני צדדים – לדוגמה, בין מחשב של משתמש לבין אתר אינטרנט – מבלי שתידרש פעולה אקטיבית מצידם.

במצב כזה, "האיש שבאמצע" יכול לצפות במידע שעובר, ולשנות אותו אם יבחר.

בפרויקט אני משתמש ב ״איש באמצע״ בשילוב עם ARP spoofing בכדי להסתכל בבקשות בין הילד לבין הראוטר, ובמידת הצורך (בקשת DNS עם דומיין או דומיינים מסוימים) לשלוח תשובה במקום הראוטר.

המשמעות הטכנית של הדבר (והמימוש הספציפי שבו אני משתמש) היא שהמחשב של הילד שולח את התעבורה אל הIP של הראוטר, אך בגלל הARP spoofing, זה מגיע אל השרת. במקרה והבקשה לא רלוונטית השרת מעביר אותה **ללא שינוי** אל הראוטר. **הראוטר יחזיר את הבקשה ישירות אל הילד**, בניגוד לשימושים הזדוניים של איש באמצע.

במידה והפקטה היא בקשת DNS רלוונטית, השרת יחזיר אל המחשב של הילד תשובת DNS, שלכאורה הגיע מהשרת DNS. כתובת הIP שתהיה בתשובה, היא הIP של השרת.

### **הרעלת DNS**

תופעה שבה, בצורה מכוונת איש באמצע מחזיר תשובת DNS לא נכונה.

בפרויקט אני משתמש בזה, כדי לנווט לשרת HTTP שרץ על מכונת השרת במידה והילד הכניס לדפדפן URL שצריך לבצע לו redirect.

## **הטכנולוגיה הרלוונטית**

### **שפות תכנות**

בקאנד באמצעות python.

פרונטאנד באמצעות html וcss.

רנדרינג באמצעות flask בpyhton.

### **מערכות הפעלה**

שימוש בקבצי json לשמירת הנתונים הבאים בצד שרת –

נתוני כניסה לאתרים

משתמשים קיימים

מאגר קישורים

שימוש במודול multiprocessing בכדי להזניק ולכבות תתי תהליכים, בעת הצורך.

תקשורת בין תתי התהליכים (IPC) באמצעות PIPE.

שימוש במודול signal כדי לכבות תת תהליכים בצורה נקייה ולנקות משאבים.

שימוש בLOCK בגישה לקבצים.

### **תקשורת**

תקשורת בסיסית בין שרת ללקוח, באמצעות אובייקט SOCKET מעל TCP בפייתון.

תקשורת עם שרת http כדי לבצע redirect.

שליחת פקטות ARP (שׁכבה 2) בכדי לנתב את בקשות הDNS אל השרת. בעזרת scapy.

הסנפה בעזרת scapy של כל הפקטות שנשלחות, כדי לתפוס פקטות DNS שמבקשות domainים מסוימים.

שליחת פקטות DNS Response בעזרת scapy.

# **מבנה וארכיטקטורה של הפרוייקט**

## **תיאור הארכיטקטורה של המערכת**

### **תיאור החומרה**

**שרת** –

מריץ את השרת הראשי שאחראי על תקשורת עם הלקוחות, ובנוסף מריץ שלושה תתי תהליכים:

* תהליך שמבצע arp spoofing לילד
* תהליך שמממש איש באמצע (mitm), כדי לתפוס ולשנות בקשות dns מסוימות
* תהליך שמריץ שרת http, כדי לטפל בredirectים.

שלוש התהליכים מוזנקים בתחילת ריצת של התוכנית המרכזית, ויוצאים בצורה נקייה כשהתוכנית הראשית מסיימת.

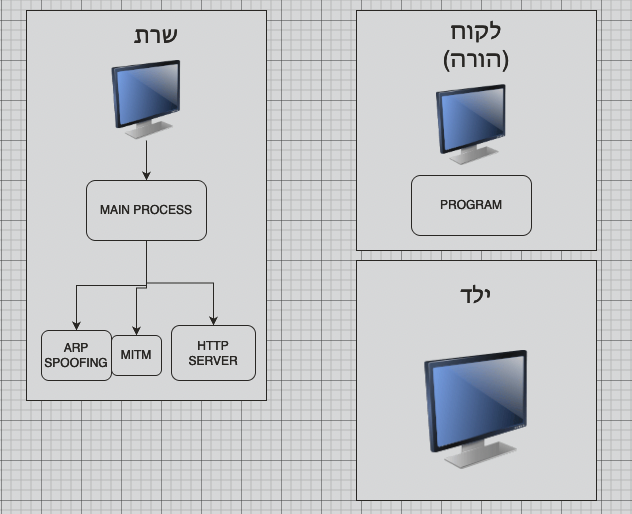
**לקוח** –

הלקוח מריץ את האפליקציה, ואז מתחבר מהדפדפן לhttp://localhost:5000. מאחורי הקלעים ממומש שרת http, שדואג לgui.

**ילד –**

גולש כרגיל בדפדפן. לא צריך לבצע שום פעולה אקטיבית.

### **החומרה בשרטוט**



## **זרימת מידע**

### **כללי**

ישנן שלוש ישויות. הלקוח (ההורה), השרת והילד. הלקוח והשרת מדברים בינהם בפרוטוקול סינכרוני.

מטרת פרוטקול זה (כפי שיורחב בהמשך) היא לתת להורה שליטה דינמית על מסד הנתונים של הקישורים.

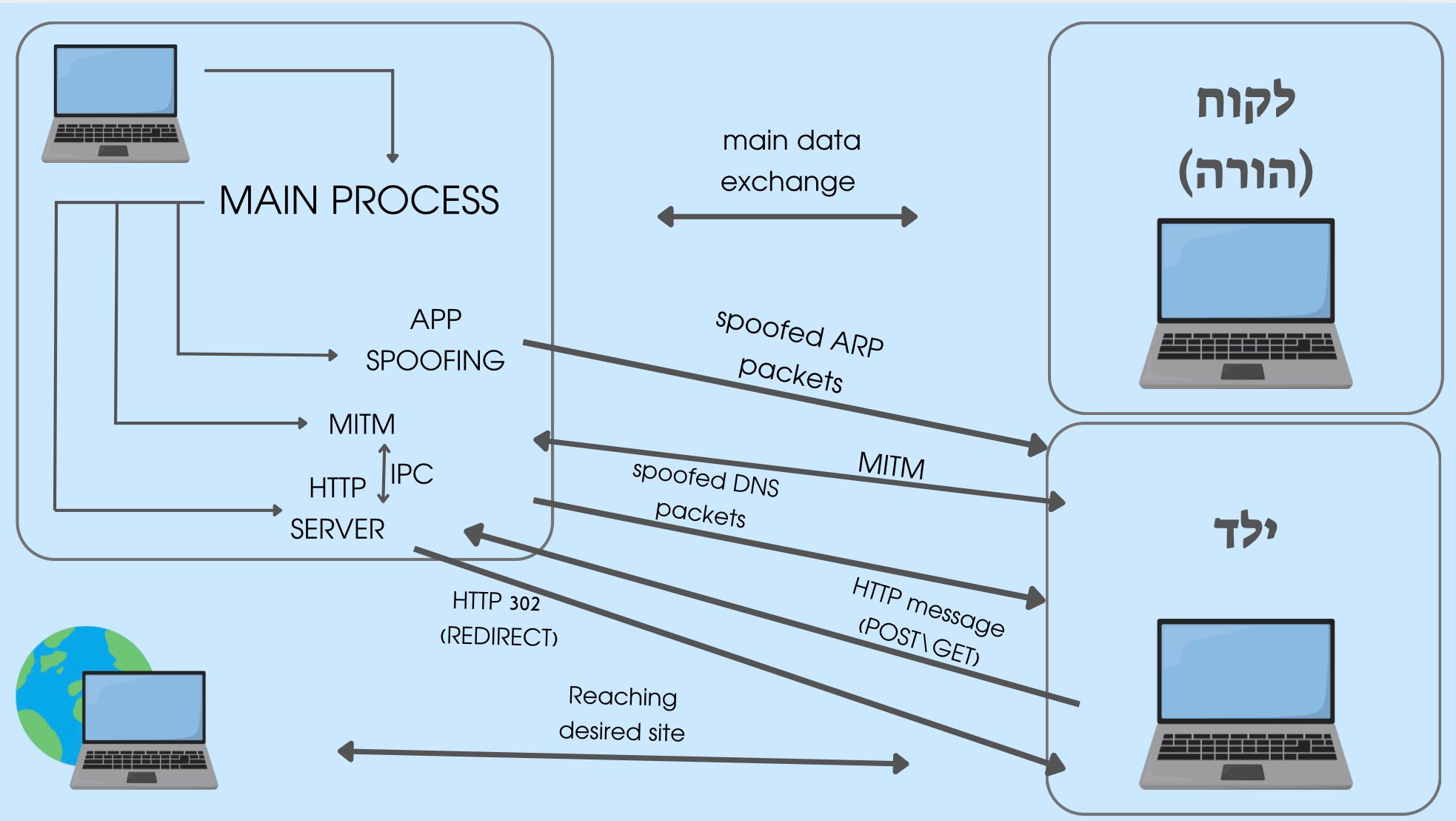
השרת מריץ ברקע תהליכים כפי שפירטתי, במטרה לשנות את התנהגות המחשב של הילד.

השרת גורם למחשב של הילד לחשוב שהשרת הוא הראוטר, ולכן כל התעבורה (ובכלל זה כלל בקשות הDNS), יעברו דרך השרת. בבקשות שאינן בקשות DNS, השרת יעביר כפי שהן אל הראוטר. בגלל המימוש הספציפי של איש באמצע שבו משתמשים (וכפי שהוסבר בחלק הרקע התיאורטי) אין צורך לטפל בהעברת התשובה של הראוטר.

כאשר הילד ירצה לגלוש אל אחד מן האתרים המנוטרים, ויזין בדפדפן את URL שלא קיים, תישלח בקשת DNS, על אותו הדומיין. הבקשה תישלח אל השרת, שישים לב שהדומיין קיים במסד הנתונים שההורה מנהל באופן דינמי, ולכן במקום להעביר לראוטר יחזיר תשובה מזויפת, עם הIP שלו. בנוסף, ינטר את הכניסה, ויכתוב אותה במסד נתונים הרלונטי, וגם יעדכן את תת התהליך של שרת הHTTP, שבוצע ניסיון התחברות לכתובת מסוימת, ולאיזו כתובת לעשות redirect.

הדפדפן ינסה לפנות אל השרת, ויגיע לשרת HTTP, שמורץ כתת תהליך מהתהליך הראשי של השרת. שרת הHTTP, ציפה לפנייה הזאת (כיוון שתת התהליך של הMITM עדכן אותו), וידע לאן המחשב של הילד ניסה להתחבר, וישלח תשובת HTTP עם קוד 302 (redirect) עם הכתובת התואמת.

תוצאת כל התהליך, היא שהילד הכניס לדפדפן איזשהו URL מזויף, והגיע לאתר אמיתי, בלי שירגיש, וכניסתו נותרה ותועדה. ההורה בוחר לאלו אתרים לייצר קישור מזויף, שלאחר מכן ייתן לילד כדי שתיחבר דרכם.

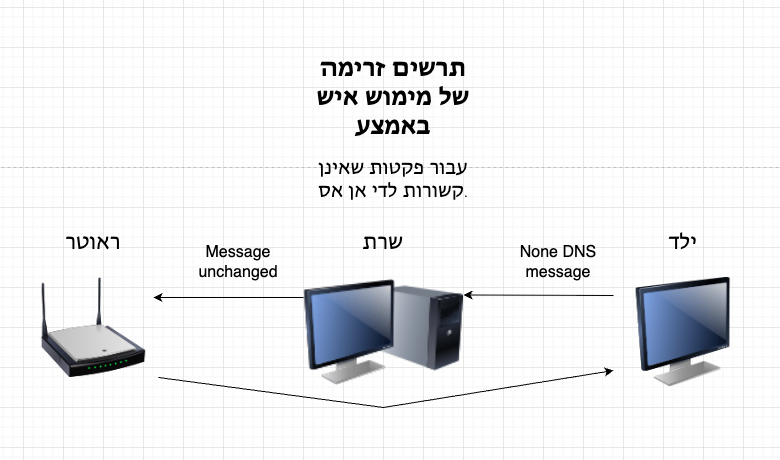


### **תת תהליך MITM + הרעלת DNS.**

פקטות רגילות (איש באמצע) –

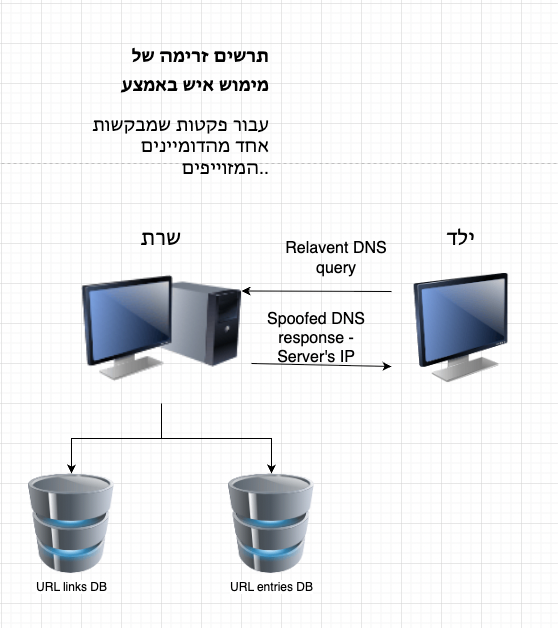
השרת יקבל את ההודעה של הילד, יבין שהיא לא רלוונטית, ויעביר אותה **ללא שינוי** אל הראוטר. הראוטר יחזיר את ההודעה ישירות אל המחשב של הילד.

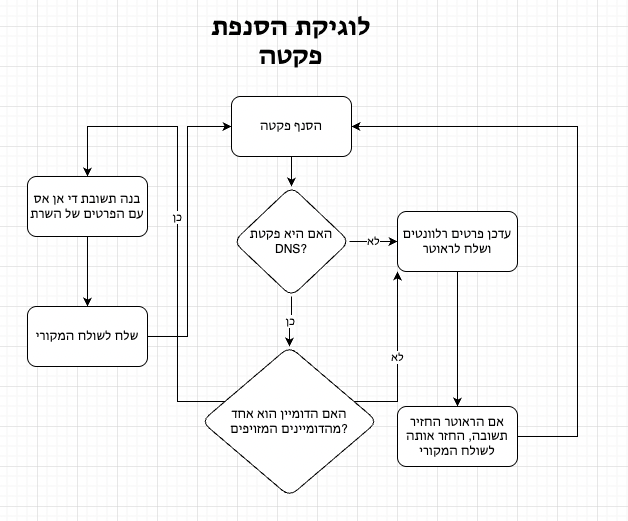
**הבהרה:** מימוש זה שונה מהמימוש הנפוץ של MITM, שבו בדר״כ גם התשובה עוברת דרך האיש באמצע. במקרה שלנו, אין צורך בלהתפרץ להודעות החוזרות מהראוטר.



פקטות DNS, שהדומיין שלהם נמצא במאגר בתור קישור מנוטר (הרעלת DNS) -

השרת יזהה שמדובר בדומיין שרשום בתור דומיין שצריך לנטר, ולכן יבנה ויחזיר תשובת DNS, עם הIP שלו, ישלח לשרת HTTP לאן לעשות redirect, ויתעד את הכניסה. במקרה כזה אין שימוש באיש באמצע.





## **אלגוריתמים מרכזיים בפרויקט**

### **תקשורת בין תתי תהליכים**

באופן כללי, תקשורת בין תתי תהליכים (IPC) מתבצעת כאשר יש צורך בשיתוף מידע בין שני תהליכים שרצים בנפרד.

כפי שתואר, יש צורך בתקשורת בין תת התהליך של http לבין תת התהליך של MITM, כדי לדעת לאיזה כתובת יש לבצע redirect לילד.

**פיתרון 1: PIPE**

איזשהו buffer בזיכרון שנוצר ע״י מערכת ההפעלה, אליו תהליך אחד כותב ותהליך אחר קורא.

|  |  |
| --- | --- |
| יתרונות | חסרונות |
| תקשורת פשוטה מבוססת קבצים. | חד כיווני. |
| נתמך בכל מערכות ההפעלה. | איטי יותר מאפשרויות אחרות |
| פשוט מאוד למימוש | דירוש תזמון בין הקריאה לכתיבה – עלול לגרום blocking |

**פיתרון 2 : Shared memory**

אזור בזיכרון שניתן לשתף בין שני תהליכים – שני התהליכים ניגשים לאותו מרחב זיכרון

|  |  |
| --- | --- |
| יתרונות | חסרונות |
| הביצועים הגבוהים ביותר | דורש סנכרון ידני. סכנה לrace condition |
| |  | | --- | | שימושי במיוחד במערכות real time | | מורכב למימוש |
|  | טעויות בסנכרון יגרמו לבאגים קשים |

**פתרון 3: תקשור באמצעות socket פנימי.**

socket מקומי בתוך מערכת הקבצים. התהליכים מדברים בפרוטוקול דמוי TCP, אך בלי חיבור ממשי לרשת.

|  |  |
| --- | --- |
| יתרונות | חסרונות |
| תומך בתקשורת דו כיוונית | תקורה יחסית גבוהה |
| מאפשר שליחת קבצים והנדלים | פחות יעיל מאפשרויות אחרות |
| מתאים במיוחד לתהליכים לא תלויים | דורש קונפיגורציה וקוד די מורכבים |

**סיכום אפשרויות:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| שיטה | דו כיוונית | קלות מימוש | מהירות | דורשת סנכרון |
| PIPE | לא | בינונית | בינונית | חלקי |
| Shared memory | כן | נמוכה | גבוהה מאוד | כן (קריטי) |
| Sockets | כן | בינונית-נמוכה | בינונית-נמוכה | תלוי במימוש |

**למה דווקא pipe?**

במימוש שאנחנו מחפשים, אנחנו זקוקים לתקשורת חד כיוונית, שעובדת לפי סדר FIFO, ולכן PIPE הוא אפשרות מעולה, שמספקת פתרונות לשתי הבעיות האלה, במימוש קל ביותר.

## **סביבת הפיתוח**

### **כלי הפיתוח הדרושים**

|  |  |
| --- | --- |
| תחום | כלים / טכנולוגיה שתעשה בהם שימוש |
| Frontend | * HTML * CSS 3 * Browser (firefox, chrome) |
| Backend | * Python 3 * Socket * FLASK (python) |
| DB | * SQLite * Json |
| Networking | * Scapy * ARP * DNS * HTTP |
| IDE | * Visual Studio Code |
| ניהול גרסאות | * Git |

### **סביבה וכלים הדרושים לבדיקות**

לבדוק gui – דפדפן.

לבדוק תקשורת בין שרת ללקוח – socket עובד, וterminal.

לבדוק את אפקטיביות תהליכי הרקע של השרת – מספר מחשבים באותה הרשת, וwireshark מותקן. זאת בנוסף לscapy וsocket מותקנים ומעודכנים.

## **תיאור הפרוטוקול**

### **הסבר מילולי**

הפרוטקול הוא פרוטוקול סינכרוני, שמתבצע בארכיטקטורת שרת לקוח באמצעות TCP.

תיאור הפרוטוקול מפרט את כלל ההודעות שנשלחות בין השרת ללקוח, וכולל את הודעות השגיאה.

הפרוטוקול מחולק לשדות, כאשר המפריד הוא ‘~’. השדה הראשון הוא אורך ההודעה, (לא כולל את אורך השדה), והוא לא בגודל קבוע. השדה השני הוא קוד ההודעה, וכל השדות הנוספים הם פרמטרים.

### **פירוט**

פורמט ההודעה - <size>~<code>~<parm1>~<parm2>~…

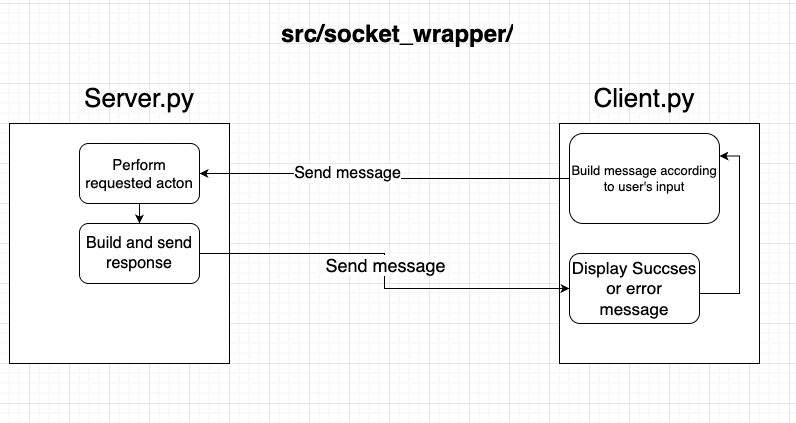
סוגי ההודעות –

|  |  |  |
| --- | --- | --- |
| **Code** | **Parmateres** | **Message Format** |
| SIGN\_UP | username: desired account name password: user's password  confirm\_password: retyped password for confirmation | SIGN\_UP~<username>~<password>~<confirm\_password> |
| SIGN\_IN | username: existing account name  password: account password | SIGN\_IN~<username>~<password> |
| ADD | url: add a URL to provide a fake URL for. | ADD~<url> |
| DEL | fake\_url: the fake URL to be removed | DEL~<fake\_url> |
| GET | fake\_url: the fake URL whose real URL is requested | GET~<fake\_url> |
| REQ | fake\_url: the fake URL to retrieve access statistics for | REQ~<fake\_url> |
| HELLO | N/A | HELLO |
| ERR | function number: every function has a corresponding number  message: error message | ERR~<function\_number>~<error\_message> |

### **מימוש**

ביצוע הפרוטוקול ממומש בתוך תיקיית socket\_wrapper. ישנם קבצי client.py וserver.py שמספקים API לשרת וללקוח בהתאמה.  
  
למעשה לכל הודעה מוקצית פעולה, שמקבלת פרמטרים ובונה לפיהם את ההודעה, ולאחר מכן מחזירה את ההודעה.

השרת מבצע את ניתוח ההודעה בתוך אותו קובץ, ובונה בצורה אוטומטית את התשובה הנכונה.



## **תיאור מסכי המערכת**

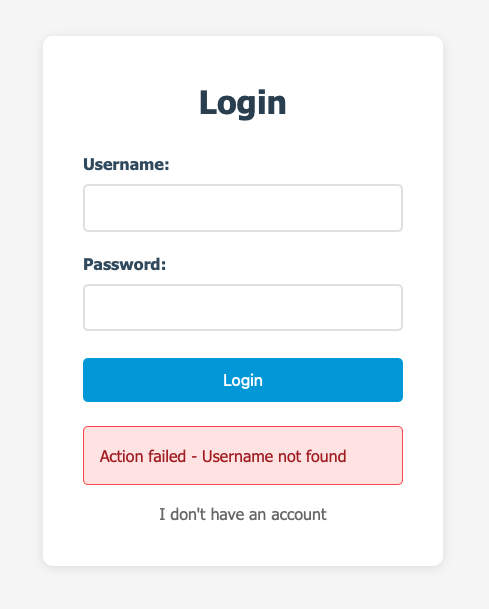
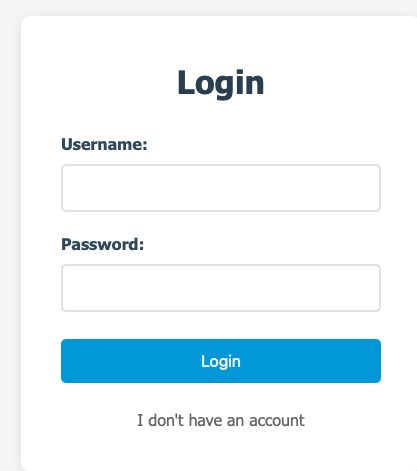
### **מסך התחברות**

מסך זה הוא מסך הפתיחה של האפליקציה.

הוא נותן למשתמשים שכבר נרשמו אל המערכת אופציה להתחבר אל החשבון הקיים שלהם.

במסך זה שני שדות, אחד לשם משתמש ואחד לסיסמא, וישנה אפשרות לעבור למסך יצירת משתמש במידה ולא קיים חשבון. בנוסף, במסך זו יוצגו הודעות שגיאה במידה והפרטים אינם נכונים. כאשר המשתמש יזין פרטים נכונים, הוא יעבור בצורה אוטומטית למסך התפריט הראשי.

כאשר מזינים את הסיסמא, המסך מציג נקודות כדי לשמור על פרטיות המשתמש.



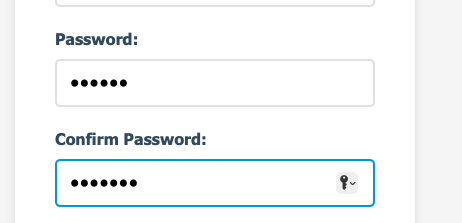


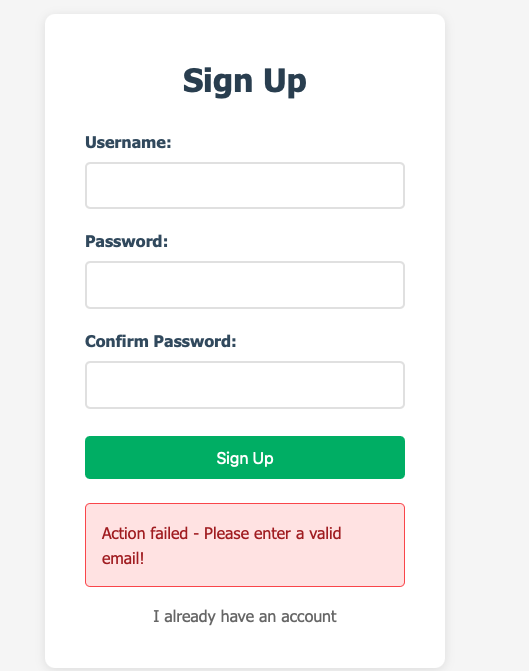
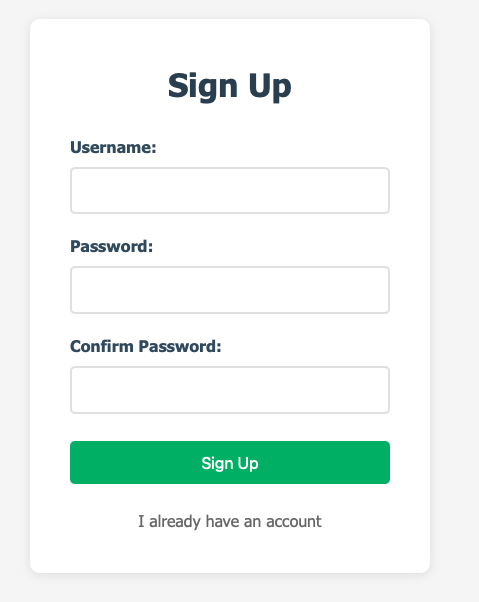
### **מסך הרשמה**

מסך זה נותן למשתמשים ליצור משתמש חדש.

במסך זה שלוש שדות, אחד לשם משתמש, אחד לסיסמא ואחד לאימות סיסמא. ניתן לעבור למסך התחברות במידה וקיים חשבון. בנוסף, במסך זו יוצגו הודעות שגיאה במידה והפרטים אינם נכונים. כאשר המשתמש יזין פרטים תקינים (מייל נכון וסיסמאות תואמות) הוא יעבור באופן אוטומטי אל מסך ההתחברות.

כאשר מזינים את הסיסמא, המסך מציג נקודות כדי לשמור על פרטיות המשתמש.



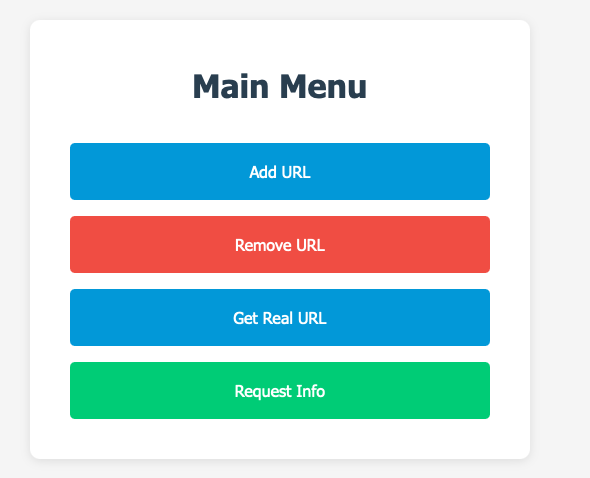


### **תפריט ראשי**

מסך זה מהווה מסך מקשר בין כלל הפונקציונליות השונות של ממשק המשתמש, והוא התפריט הראשי.

הוא מציג בפני המשתמש אפשרות לבחור לאן ירצה להמשיך. בלחיצה על האפשרות, הוא יגיע למסך התואם.

ממנו ניתן להגיע למסכים הבאים: הוספת קישור, מחיקת קישור, בירור קישור אמיתי, ובקשת מידע.



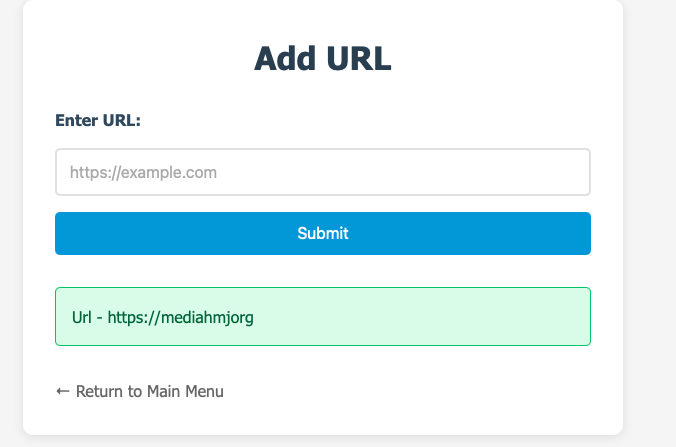
### **הוספת קישור**

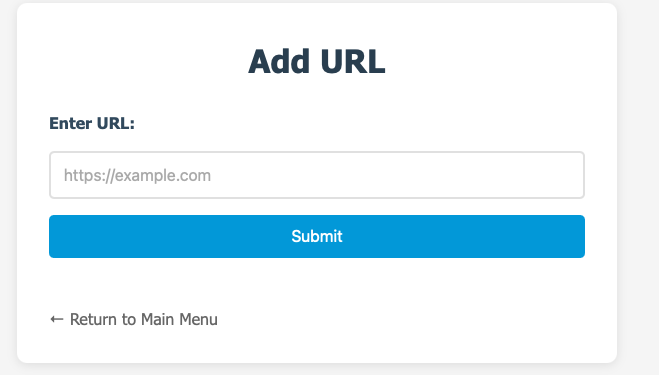
מסך זה נותן למשתמש להוסיף קישור חדש, אל מאגר הקישורים שמתנהל אחריהם פיקוח.

מסך זה מכיל שדה, שניתן להזין לתוכו את הקישור החדש, וכפתור כדי להוסיף אותו. (ניתן גם ללחוץ על אנטר). לאחר מכן, יוצג על המסך את הקישור המזוייף שמנתב אל האתר שהוספנו.

בנוסף ניתן לחזור אל התפריט הראשי.

במידה והקישור כבר קיים במאגר, תוצג הודעת שגיאה במסך זה.





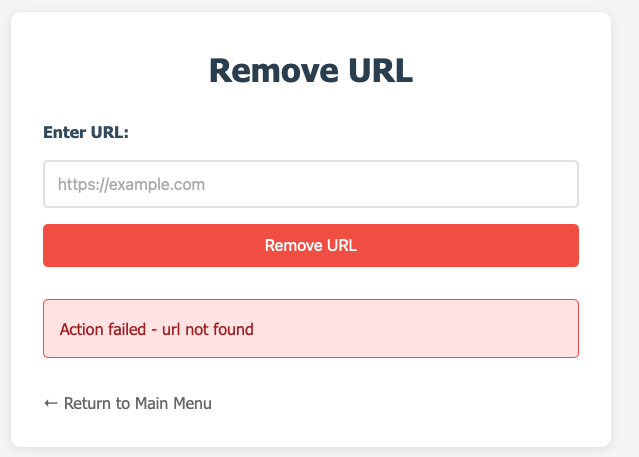
### **מחיקת קישור**

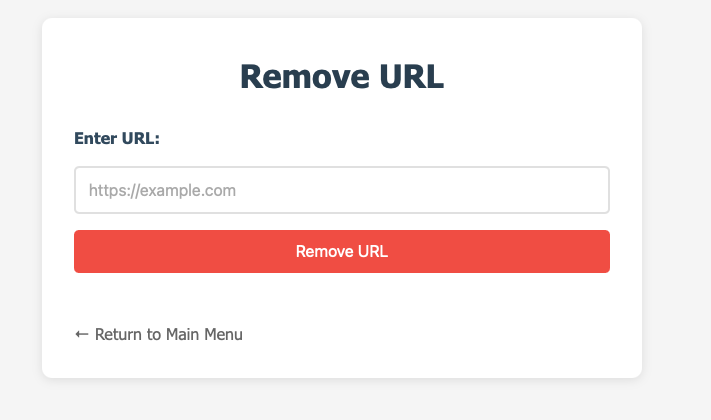
מסך זה מאפשר למשתמשים למחוק קישורים מהמאגר.

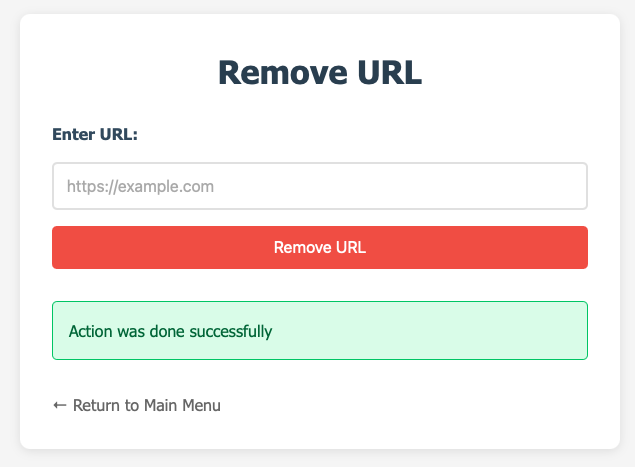
מסך זה מכיל שדה, שניתן להזין לתוכו את הקישור, וכפתור כדי למחוק אותו. (ניתן גם ללחוץ על אנטר). במידה והפעולה בוצע בהצלחה, תופיעי על המסך הודעת הצלחה.

במידה והקישור לא קיים במאגר, תוצג הודעת שגיאה במסך זה.

בנוסף ניתן לחזור אל התפריט הראשי.







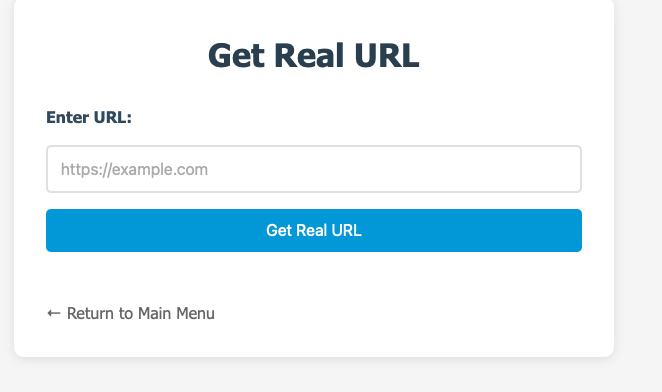
### **מציאת כתובת אמיתית**

מטרת המסך היא לאפשר למשתמש להזין קישור מזויף, ולקבל את הכתובת האמיתית שהוא מנתב אליה.

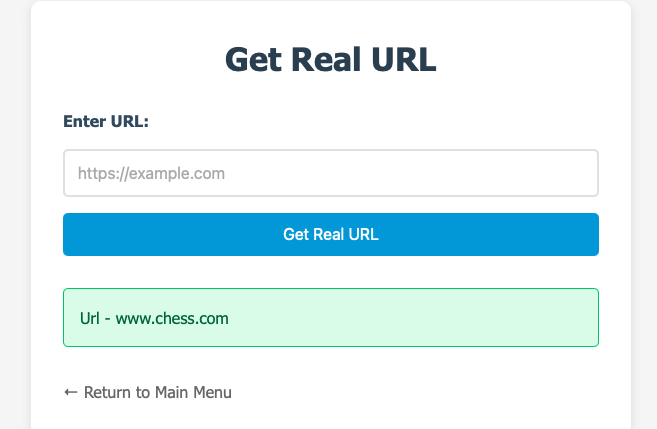
מסך זה מכיל שדה, שניתן להזין לתוכו את הקישור, וכפתור כדי להגיש אותו. (ניתן גם ללחוץ על אנטר). במידה והקישור קיים, תופיע הכתובת האמיתית על המסך.

במידה והקישור לא קיים במאגר, תוצג הודעת שגיאה במסך זה.

בנוסף ניתן לחזור אל התפריט הראשי.



לאחר הזנת קישור שמנתב אל הכתובת www.chess.com



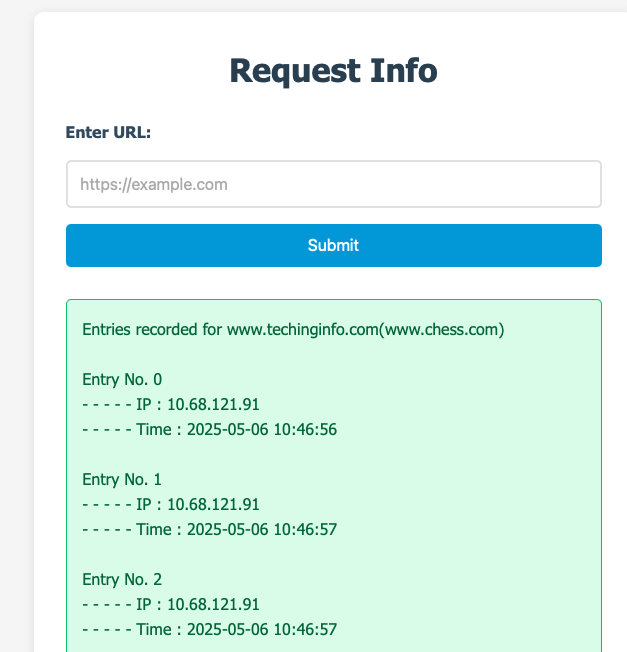
### **בקשת מידע**

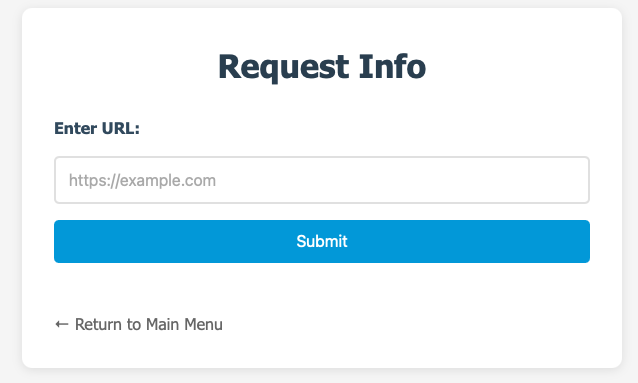
המסך מאפשר למשתמש לבקש את כל ההתחברויות שנעשו לכתובת מסוימת.

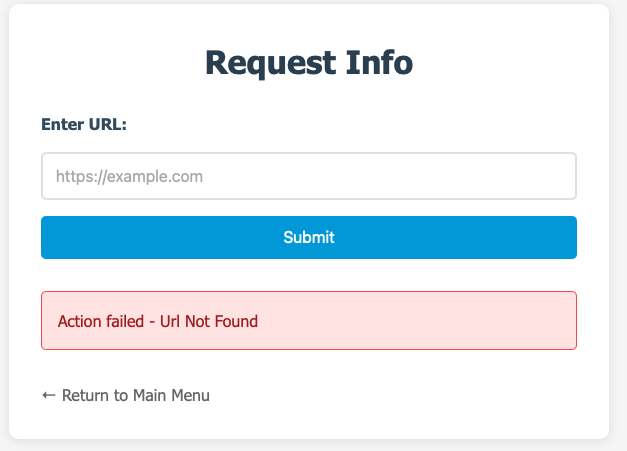
מסך זה מכיל שדה, שניתן להזין לתוכו את הקישור, וכפתור כדי להגיש אותו. (ניתן גם ללחוץ על אנטר). במידה והקישור קיים, יופיע כל המידע הלוונטי על המסך.

במידה והקישור לא קיים במאגר, תוצג הודעת שגיאה במסך זה.

בנוסף ניתן לחזור אל התפריט הראשי.



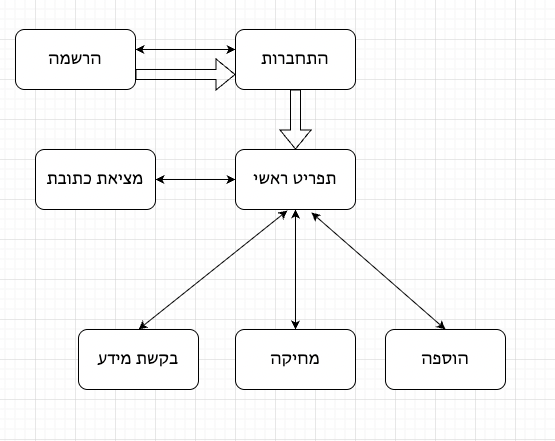




### **תרשים זרימה מסכים**

מקרא: חוץ עבה – מעבר אוטומטי ברגע שהפרטים תקינים.

חץ דק – ניתנת למשתמש אופציה לעבור בין המסכים.



## **תיאור מסדי הנתונים**

### **משתמשים**

שמירת המשתמשים מתבצעת בקובץ json בשם users.json שממוקם בroot.

הjson מכיל מילון:

מפתח (מחרוזת) - מייל

ערך (מערך של מחרוזות) – סיסמא (לאחר האש), סולט.

### **קישורים**

שמירת הקישורים מתבצעת בקובץ json בשם urls.json שממוקם בroot.

הjson מכיל מילון:

מפתח (מחרוזת) - קישור מזויף

ערך (מחרוזת )– קישור אמיתי.

### **כניסות**

תיעוד הכניסות מתבצע במסד נתונים בsql.

שם המסד: entries

טבלה לכל url.

שדות:

שם – IP, טיפוס – מחרוזת

שם – time, טיפוס מחרוזת

# 

# **מימוש הפרויקט**

## **סקירת מודולים**

### **מודולים מיובאים**

|  |  |  |
| --- | --- | --- |
| **מודול** | **שימוש בפרויקט** | **תיאור קצר וכללי** |
| Os | מחיקה ויצירה של קבצים, הרכבת pathים, הריגת תהליכים. | ממשק למערכת הפעלה |
| Json | שמירת מבני נתונים שונים בקבצים | סריליזציה פשוטה |
| Typing | יצירת typehint מתקדמים עם תמיכה בpython 3.8 | מספק טיפוסים מתקדמים לtyphint. |
| Pickle | שליחת מבני נתונים בתקשורת. | סריליזציה מתקדמת |
| Socket | תקשורת TCP בין שרת ללקוח | ממשק למימוש תקשורת רשת |
| Sys | הוספת pathים לקובץ. | ממשק לפעולות הקשורות למערכת |
| Random | יצירת string אקראי (סולט) | יצירת ערכים אקראיים |
| String | פעולות של סטרינגים | פעולות של סטרינגים |
| Pathlib | שימוש בPath מתוך המודול, בכדי להוסיף pathים לקובץ. | עבודה עם מיקומי קבצים. |
| fuctools | שימוש בwraps ליצירת דקורטרים. | פונקציות עזר לפונקציות. |
| Signal | יציאה נקייה של פרוססים. | טיפול באותות מערכת להפסקה מסודרת של תהליכים |
| Jinja2 | יצירת תגי html | מנוע תבניות html לflask |
| Flask | רנדרינג לgui | ממשק לבניית שרתי http, ולממשק משתמש גרפי. |
| Threading | שימוש בLOCk בכדי לסנכרן פרוססים שונים. | ניהול תהליכונים. |
| Time | רישום זמן. | מדידת זמן, והשהייה. |
| Scapy | שליחת פקטות arp וdns | ממשק לבנייה וניתוח של פקטות רשת. |
| http.server | יצירת שרת http | מודול ליצירת שרת http בפייתון |
| subprocess | יצירת תתי תהליכים | ממשק להזנה, עצירה וניתוח של תתי תהליכים. |
| Traceback | הצגת הודעות שגיאה בתפיסת שגיאות. | הדפסת מידע על שגיאות וstack trace |
| re | בדיקת מייל תקין. | עבודה עם ביטויים רגוליים regex |

### **מודולים בפיתוח עצמי**

אביא את כלל הדוקומנטציה בצורת טבלה, שלשם נוחות תהיה באנגלית.

מעל כל טבלה, מצוין המודול, והקלאס שבתוכו נמצאות הפעולות.

app.py - Client Manager

Properties

| **Property Name** | **Type** | **Purpose** | **Description** |
| --- | --- | --- | --- |
| \_sock | socket | Socket connection | Socket for server communication |
| \_app | Flask | Flask application | Web application instance |
| \_session | Dict[str, Any] | Session storage | Stores user session data |

Functions

| **Function Name** | **Input** | **Output** | **Purpose** | **Description** |
| --- | --- | --- | --- | --- |
| \_\_init\_\_ | self | None | Initialize Flask application and socket connection | Sets up Flask app with templates, creates socket connection, starts server |
| \_setup\_routes | self | None | Register Flask routes | Maps URL routes to handling methods and sets up Jinja filters |
| exit | self | tuple(str, int) | Handle application exit | Performs cleanup and terminates application process |
| cleanup | self | None | Clean up resources | Performs necessary cleanup operations for client |
| start\_menu | self | str | Display main menu interface | Renders menu template for URL management options |
| login | self | str or redirect | Handle user login | Processes login form, authenticates with server, manages session |
| signup | self | str or redirect | Handle user registration | Validates signup data, registers with server, manages response |
| main\_menu | self | str | Display main menu after login | Renders main menu template with user-specific options |
| add\_url | self | str or redirect | Handle URL addition | Processes URL submission and communicates with server |
| remove\_url | self | str or redirect | Handle URL removal | Processes URL removal and communicates with server |
| get\_real\_url | self | str or redirect | Retrieve real URL | Gets original URL corresponding to shortened URL |
| req\_info | self | str or redirect | Request URL information | Gets information about specific URL entry |
| \_nl2br\_filter | text (str) | str | Convert newlines to HTML | Replaces newline characters with HTML break tags |

**data\_helper.py - Data Management**

| **Function Name** | **Input** | **Output** | **Purpose** | **Description** |
| --- | --- | --- | --- | --- |
| get\_data | None | DataDict | Read JSON data file | Loads JSON data file into dictionary structure |
| save\_data | data (DataDict) | None | Save to JSON file | Writes dictionary data to JSON file |
| record\_entry | fake\_url (str), packet\_dict (PacketData) | None | Record URL access entry | Adds/updates URL access information in data store |
| fetch\_stats | fake\_url (str) | List[PacketData]/None | Retrieve URL statistics | Fetches recorded access entries for URL |

**socket\_wrapper/network\_wrapper.py - Network Base Class**

Properties

| **Property Name** | **Type** | **Purpose** | **Description** |
| --- | --- | --- | --- |
| \_serv\_sock | socket | Network socket | Base socket for network communication |

Functions

| **Function Name** | **Input** | **Output** | **Purpose** | **Description** |
| --- | --- | --- | --- | --- |
| \_\_init\_\_ | None | None | Initialize network socket | Creates and configures socket with address reuse |
| recv\_by\_size | sock (Optional[socket.socket]) | bytes | Receive sized message | Receives complete message using size field |
| send\_by\_size | to\_send (bytes), sock (Optional[socket.socket]) | None | Send sized message | Sends message with size prefix through socket |

**socket\_wrapper/client.py - Client Implementation**

| **Function Name** | **Input** | **Output** | **Purpose** | **Description** |
| --- | --- | --- | --- | --- |
| \_\_init\_\_ | ip (str), port (int) | None | Initialize client socket | Creates socket connection to server |
| recv\_by\_size | None | bytes | Receive server data | Receives data using parent class method |
| send\_by\_size | to\_send (bytes) | None | Send data to server | Sends data using parent class method |
| parse | from\_server (bytes) | tuple(str, str) | Parse server messages | Decodes server responses into user-friendly format |
| client\_hello | None | bytes | Create hello message | Returns client hello message for handshake |
| cleanup | None | None | Clean up resources | Closes server connection |
| format\_data | data (list), fake (str), real (str) | str | Format URL statistics | Creates readable string from URL stats |
| sign\_up | username, password, cpassword, err (str) | bytes | Create signup request | Formats signup request for server |
| login | username, password, err (str) | bytes | Create login request | Formats login request for server |
| add\_url | url (str), err (str) | bytes | Create URL add request | Formats URL addition request |
| remove\_url | fake\_url (str), err (str) | bytes | Create URL remove request | Formats URL removal request |
| get\_real\_url | fake\_url (str), err (str) | bytes | Create URL lookup request | Formats request for original URL |
| req\_info | fake\_url (str), err (str) | bytes | Create info request | Formats request for URL statistics |

**socket\_wrapper/server.py - Server Implementation**

Properties

| **Property Name** | **Type** | **Purpose** | **Description** |
| --- | --- | --- | --- |
| \_\_DEBUG | bool | Debug mode flag | Indicates if server is in debug mode |
| \_\_port | int | Server port | Port number server listens on |
| \_\_ip | str | Server IP | IP address server binds to |
| \_sock | socket | Client socket | Socket for client communication |
| urls\_path | str | URLs file path | Path to JSON file storing URL mappings |

Functions

| **Function Name** | **Input** | **Output** | **Purpose** | **Description** |
| --- | --- | --- | --- | --- |
| \_\_init\_\_ | port (int) | None | Initialize server socket | Sets up socket, binds to port, waits for client |
| recv\_by\_size | None | bytes | Receive client data | Receives sized message from client |
| send\_by\_size | to\_send (bytes) | None | Send data to client | Sends sized message to client |
| parse | data (bytes) | bytes | Parse client requests | Interprets and routes client commands |
| server\_hello | None | bytes | Handle client greeting | Returns acknowledgment message |
| show\_stats | fake\_url (bytes) | bytes | Get URL statistics | Fetches and formats URL access stats |
| cleanup | None | None | Clean up resources | Closes client connection |
| retrieve\_url | urls (UrlDict), fake\_url (bytes) | str | Get real URL | Retrieves original URL from mapping |
| add\_url | urls (UrlDict), real\_url (bytes) | bytes | Add URL mapping | Creates fake URL and maps to real URL |
| remove\_url | urls (UrlDict), fake\_url (bytes) | bytes | Remove URL mapping | Removes fake URL mapping if exists |
| get\_real\_url | urls (UrlDict), fake\_url (bytes) | bytes | Get real URL | Gets original URL for fake URL |
| generate\_fake\_url | None | str | Generate fake URL | Creates random fake URL using components |
| manage\_urls | func (Callable) | Callable | Manage URL operations | Handles URL file loading/saving |

**users.py - User Management**

| **Function Name** | **Input** | **Output** | **Purpose** | **Description** |
| --- | --- | --- | --- | --- |
| manage\_users | func (Callable) | Callable | Manage user data | Handles user data file access with thread safety |
| does\_user\_exists | users (UserDict), user (str) | bool | Check user existence | Verifies username in database |
| check\_sign\_in | users (UserDict), username (str), password (str) | bytes | Verify login credentials | Checks username and password validity |
| get\_salt | users (UserDict), username (str) | str | Get user salt | Retrieves password hashing salt |
| sign\_up | users (UserDict), username, password, salt | bytes | Register new user | Creates new user account after validation |
| create\_salt | None | str | Generate salt | Creates random hex salt for passwords |
| \_hash | to\_hash (str) | str | Create password hash | Generates SHA-256 hash of input |
| clear | None | None | Clear user data | Removes users data file |
| load\_users | None | UserDict | Load user database | Loads user data from JSON file |
| is\_valid | email (str) | bool | Validate email format | Checks email against regex pattern |

**server.py - Main Server**

| **Function Name** | **Input** | **Output** | **Purpose** | **Description** |
| --- | --- | --- | --- | --- |
| start\_processes | host\_ip, target\_ip, router\_ip (str) | None | Start background processes | Launches HTTP server and DNS poisoning processes |
| kill\_processes | None | None | Stop background processes | Terminates running background processes |
| main | None | None | Main server function | Initializes and runs server with cleanup |

**networking.py - Network Packet Spoofing**

Properties

| **Property Name** | **Type** | **Purpose** | **Description** |
| --- | --- | --- | --- |
| \_\_ip | str | Host IP | IP address of host machine |
| \_\_target\_ip | str | Target IP | IP of target machine |
| \_\_router\_ip | str | Router IP | IP address of router |
| \_\_target\_mac | str | Target MAC | MAC address of target |
| urls | Dict[str, str] | URL mappings | Dictionary of URL mappings |

Functions

| **Function Name** | **Input** | **Output** | **Purpose** | **Description** |
| --- | --- | --- | --- | --- |
| \_\_init\_\_ | host\_ip, target\_ip, router\_ip (str) | None | Initialize spoofer | Sets up spoofer with network addresses and target MAC |
| send\_spoofed\_packet | None | None | Send spoofed ARP | Creates and sends ARP packet pretending to be router |
| restore\_defaults | dest, source (str) | None | Restore ARP mappings | Sends ARP packets to restore original network config |
| get\_mac | ip (str) | str | Get MAC address | Uses ARP to discover MAC address of given IP |
| process\_packet | packet (Any) | None | Process DNS queries | Handles DNS queries, provides spoofed responses |
| forward\_to\_router | None | None | MITM attack | Sniffs and processes network packets during attack |
| build\_dict\_from\_packet | packet (Any) | Dict[str,str] | Extract packet info | Creates dictionary with timestamp and source IP |
| get\_urls | None | Dict[str,str] | Load URL mappings | Loads URL mappings from urls.json file |
| nslookup | domain (str) | Packet | DNS lookup | Sends DNS query and returns response packet |

**http\_helper.py - HTTP Request Handling**

Properties

| **Property Name** | **Type** | **Purpose** | **Description** |
| --- | --- | --- | --- |
| MAPPER | ClientMapper | Client mapper | Maps client IPs to domains |

Functions

| **Function Name** | **Input** | **Output** | **Purpose** | **Description** |
| --- | --- | --- | --- | --- |
| do\_GET | None | None | Handle GET requests | Retrieves target domain for client IP, sends redirect |
| do\_POST | None | None | Handle POST requests | Delegates POST handling to GET handler |
| run\_http\_server | port (int) | None | Start HTTP server | Creates and runs HTTP server with redirect handler |

**dns\_poison.py - DNS Spoofing**

Properties

| **Property Name** | **Type** | **Purpose** | **Description** |
| --- | --- | --- | --- |
| URLS | Dict[str,str] | URL mappings | Dictionary of URLs to spoof |
| SPOOF\_IP | str | Spoof IP | IP to redirect to (localhost) |
| INTERFACE | str | Network interface | Interface for packet sniffing |
| MAPPER | ClientMapper | Client mapper | Maps clients to domains |

Functions

| **Function Name** | **Input** | **Output** | **Purpose** | **Description** |
| --- | --- | --- | --- | --- |
| load\_urls | None | Dict[str,str] | Load URL mappings | Loads mappings from urls.json or uses defaults |
| dns\_spoof | pkt (Any) | None | Handle DNS spoofing | Analyzes DNS queries, sends spoofed responses |
| build\_dict\_from\_packet | pkt (Any) | Dict[str,str] | Extract packet info | Creates dict with source IP and timestamp |

**cli\_mapper.py - Client IP Mapping**

Properties

| **Property Name** | **Type** | **Purpose** | **Description** |
| --- | --- | --- | --- |
| \_\_map | Dict[str,str] | Client mappings | Maps client IPs to domains |
| \_\_lock | Lock | Thread lock | Synchronization for thread safety |

Functions

| **Function Name** | **Input** | **Output** | **Purpose** | **Description** |
| --- | --- | --- | --- | --- |
| \_\_init\_\_ | None | None | Initialize mapper | Creates empty map and thread lock |
| add\_client | ip, domain (str) | None | Add client mapping | Maps client IP to requested domain |
| get\_domain | ip (str) | str | Get client domain | Retrieves and removes domain mapping for IP |
| get\_map | None | None | Load client mappings | Loads IP-domain mappings from file |
| save\_map | None | None | Save client mappings | Saves current mappings to file |

## **קטעי קוד ראויים לציון**

### **תקשורת בין תהליכים**

שימוש בpipe בכדי לתקשר בין התהליכים, כפי שפורט בחלק על [בעיות אלגוריתמיות](#_תקשורת_בין_תתי)

ב server.py (הזנקת התהליכים)



בתוך http\_helper.py



בתוך dns\_poison.py



### **שימוש בwrapper לסוקט**

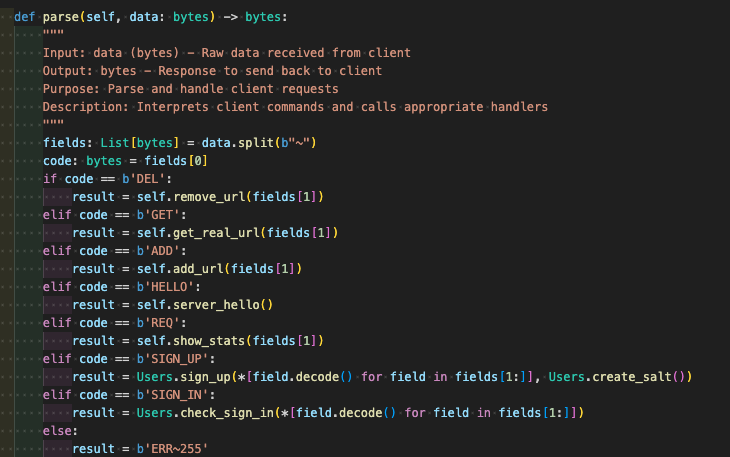
כלל הטיפולים בשליחה וקבלה של מידע בsocket, מטופל ע״י חבילה שכתבתי. החבילה מכילה שלושה קבצים – network\_wrapper.py, client.py ו server.py.   
network\_wrapper הינה מחלקת בסיס לשתי המחלקות האחרות (Serverו Client). הוא מממש פעולות בסיסיות של שליחה וקבלה של הודעות.

שתי המחלקות האחרים יורשות אותו, וכל אחד מהם מממש את כלל הפעולות הקשורות לפרוטוקול.

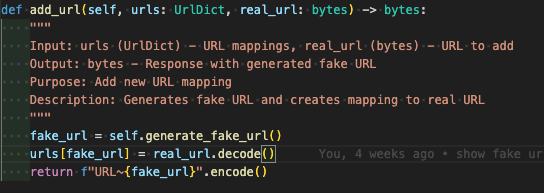
כלומר: במקום שהשרת יפרסר את ההודעה ויבנה תשובה בתוך הקובץ הבסיסי של השרת, הקובץ הבסיסי של השרת מכיל אוביקט מסוג Server, שעליו הוא קורא לפעולה Parse, שיודעת לקלוט את ההודעה, לפרק אותה לרכיבים, לבצע את הפעולות המבוקשות, לבנות תשובה, ולהחזיר אותה.

בשיטה דומה פועל אובייקט הClient.

בתוך socekt\_wapper:

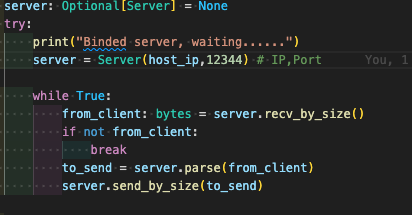


דוג׳ לפעולה – add\_url.



ניתן לראות שהפעולה מבצעת את הפעולה (מוסיפה קישור), ולאחר מכן בונה הודעה, ומחזירה אותה. הכל בתוך הwrapper!

ולעומת זאת, הקובץ הראשי server.py –



ניתן לראות שהקובץ הראשי, אינו מכיל שום התייחסות להודעות או לsocket, אלא נותן לwrapper לטפל בזה.

## **מסמך בדיקות מלא**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| מס' | יכולת לבדיקה | תיאור הבדיקה | ניסון | סטטוס |
| 1 | יצירת משתמש (הורה) | ניסיון יצירת משתמש עם סיסמה חלשה | 1 | כישלון |
| 1 | יצירת משתמש (הורה) | יצירת משתמש עם פרטים תקינים | 2 | הצלחה |
| 2 | התחברות משתמש | התחברות עם פרטי התחברות נכונים | 1 | הצלחה |
| 3 | יצירת קישור מזויף | יצירת קישור עם שם ייחודי | 1 | הצלחה |
| 4 | ARP Spoofing | הפעלת ARP Spoofing – נצפו חבילות ברשת | 1 | הצלחה |
| 5 | יירוט DNS | יירוט נכשל עקב קונפליקט פורטים | 1 | כישלון |
| 5 | יירוט DNS | ניסוי נוסף – עדיין נכשל עקב שגיאת firewall | 2 | כישלון |
| 5 | יירוט DNS | הגדרת חוקים חדשים ב-firewall – יירוט הצליח | 3 | הצלחה |
| 6 | ניהול שרת HTTP | השרת עלה וקיבל בקשות HTTP רגילות | 1 | הצלחה |
| 7 | הפניה לאתר אמיתי | הפניה גרמה ללולאת הפניות (302 חוזרת) | 1 | כישלון |
| 7 | הפניה לאתר אמיתי | תיקון ההפניה – התקבלה 302 תקינה | 2 | הצלחה |
| 8 | תיעוד כניסות | רק חלק מהכניסות נרשמו למסד הנתונים | 1 | כישלון |
| 8 | תיעוד כניסות | הוספה של Thread-safe logging – כל הנתונים נרשמו בהצלחה | 2 | הצלחה |
| 9 | הצגת נתונים בממשק הורה | גרפים לא נטענו | 1 | כישלון |
| 9 | הצגת נתונים בממשק הורה | תיקון צד לקוח – הגרפים מוצגים כראוי | 2 | הצלחה |
| 10 | תמיכה בריבוי משתמשים | הגדרת משתמשים ופרופילים – כל משתמש ראה רק את הנתונים שלו | 1 | הצלחה |
| 11 | ביצועי MITM | השהיות בתקשורת עקב עומס | 1 | כישלון |
| 11 | ביצועי MITM |  | 2 | הצלחה |
| 12 | זיהוי DNS מזויף | לא זיהה בקשות  dns | 1 | כישלון |
| 12 | זיהוי DNS מזויף | שינוי קונפגירציות scapy. | 2 | הצלחה |
| 13 | פעולה במקביל (מולטי-פרוססינג) | שימוש ב-Multiprocessing – כל המודולים עבדו במקביל | 1 | הצלחה |
| 14 | עמידות בתקלות | קריסה מלאה לאחר ניתוק רשת | 1 | כישלון |
| 14 | עמידות בתקלות | הוספת Auto-Retry – המערכת התאוששה | 2 | הצלחה |

# **מדריך למשתמש**

## **עץ קבצים**

.

├── doc/

│ ├── funciton\_doc.md

│ ├── network\_protocol.md

│ └── project main.drawio

├── src/

│ ├── data/

│ │ ├── data\_helper.py

│ │ └── data.db

│ ├── socket\_wrapper/

│ │ ├── \_\_init\_\_.py

│ │ ├── client.py

│ │ ├── network\_wrapper.py

│ │ └── server.py

│ ├── app.py

│ ├── server.py

│ ├── arp\_spoofer.py

│ ├── cli\_mapper.py

│ ├── dns\_poison.py

│ ├── http\_helper.py

│ ├── networking.py

│ └── users.py

├── ui/

│ ├── add\_url.html

│ ├── get\_real\_url.html

│ ├── login.html

│ ├── main\_menu.html

│ ├── signup.html

│ ├── remove\_url.html

│ └── req\_info.html

├── .gitignore

├── README.md

├── requirements.txt

├── urls.json

└── users.json

## **התקנת המערכת**

### **סביבה וכלים נדרשים**

* לפחות שני מחשבים.
* Python 3.8 והלאה.
* Netcap במידה ומדובר במחשב windows.
* חיבור לרשת.
* התקנת המודולים הבאים (פייתון) – flask, jinja2, scapy,

### **אתחול נתונים**

אין חובה באתחול נתונים.

במידה ורוצים להוסיף באופן ידני ולא מתוך המערכת משתמשים או קישורים יש לפעול באופן הבא:

**משתמשים** – ליצור תחת התיקייה הראשית (ברירת מחדל: project) קובץ users.json, ולכתוב בו באופן הבא.  
{“<valid@mail.name>”:[”<sha256(mail + salt)>”,”<salt>”]}

עובר כל משתמש נוסף, להוסיף פסיק לאחר ה], ורק בסוף כל המשתמשים לסגור {}.

לשים לב שהפורמט תואם לפורמט json, מבני הנתונים הרלוונטי הוא Dict[str,List[str]].

**קישורים -** ליצור תחת התיקייה הראשית (ברירת מחדל: project) קובץ urls.json, ולכתוב בו באופן הבא.

{“<fake\_url>”:”<real\_url>”}

עובר כל קישור נוסף, להוסיף פסיק, ורק בסוף לסגור {}.

לשים לב שהפורמט תואם לפורמט json, מבני הנתונים הרלוונטי הוא Dict[str,str].

הערה: אין סיבה להוסיף את הנתונים האלה ידנית, קל להוסיף אותם דרך ממשק המשתמש, ויש סיכוי גבוה שהפורמט לא יהיה נכון, ובמקרה כזה בכל מקרה הכל מאותחל.

## **אופן הפעלת המערכת**

הערה: השרת והלקוח יכולים לרוץ על שני מחשבים שונים באותה הרשת, או על אותו המחשב.

### **שרת**

יש לעדכן את הפרטים הבאים:

בקובץ src/server.py, בשורה 51, יש לעדכן את הפרטים הלוונטים.

במידה ומעוניינים לשנות את הport, עושים זאת באותו קובץ בשורה 57. (אם משנים, לוודא שהשרת על אותו הport)

בכדי להריץ:

* נווט אל התיקייה הראשית
* הקלד לterminal: pip install -r requirments.txt
* הקלד לterminal: python src/server.py

### **לקוח**

יש לעדכן את הפרטים הבאים:

בקובץ src/app.py, בשורה 21, יש לעדכן את הפרטים הלוונטים. לוודא שהport תואם לשרת.

* נווט אל התיקייה הראשית
* הקלד לterminal: pip install -r requirments.txt
* הקלד לterminal: python src/app.py
* הקלד בדפדפן https://<server\_ip>/

הלקוח יגיע אל המסך [login](#_מסך_התחברות)

# **רפלקציה**

במהלך השנה עבדתי על פרויקט שממש מערכת ניטור לרשת מקומית, שמטרתה לאפשר להורים לעקוב אחרי פעילות האינטרנט של ילדיהם. המערכת מזהה ניסיונות גישה לדומיינים שנבחרו מראש, מנתבת את הגולש דרך שרת ביניים, מתעדת את הבקשה ומחזירה אותו לאתר המקורי. המשתמש, כלומר ההורה, מקבל ממשק גרפי פשוט מבוסס Flask שמאפשר שליטה מלאה במערכת וצפייה בנתונים שנאספים.

בתהליך העבודה עברתי שינוי משמעותי באופן שבו אני ניגש לפיתוח תוכנה. בתחילת הדרך עבדתי בצורה ליניארית – פתרון אחר פתרון, קובץ אחר קובץ – מבלי להתייחס למורכבות של מערכת שמכילה מספר תהליכים שפועלים במקביל ודורשים סנכרון. מהר מאוד הבנתי שגישה כזו אינה מספיקה, ונאלצתי ללמוד איך לחלק את המערכת לרכיבים עצמאיים שנשארים מסונכרנים לאורך זמן. המעבר הזה דרש ממני להבין לעומק נושאים כמו תקשורת בין תהליכים, ניהול זיכרון משותף, ותכנון ארכיטקטורת תוכנה ברמת מערכת שלמה – דברים שלא היו מוכרים לי כלל בתחילת הפרויקט.

אחת מנקודות המפנה בפרויקט הייתה המעבר ממערכת שכוללת רק תיעוד טכני (למשל הדפסות לקונסול), למערכת עם ממשק משתמש גרפי המיועדת למשתמש לא טכני. המעבר הזה חייב שינוי גישה – הבנתי שעליי לחשוב לא רק על איך לגרום לדברים לעבוד, אלא גם על איך להציג אותם בצורה פשוטה וברורה. כתוצאה מכך למדתי לעבוד עם Flask ברמה מתקדמת, לבנות API פנימיים, ולטפל במצבים שבהם הנתונים משתנים בזמן אמת – כמו תיעוד בקשות תוך כדי שהמערכת פועלת.

לאורך הדרך היו קשיים משמעותיים – בעיקר בניסיון לגרום לכל הרכיבים לעבוד יחד בלי להתנגש. פעמים רבות המערכת עבדה חלקית: למשל, הרכיב שמיירט את ה־DNS עבד, אבל שרת ה־HTTP לא קיבל את המידע בזמן. בכל פעם כזו למדתי טכניקות חדשות לפתרון בעיות – שימוש בתהליכים נפרדים, שימוש במחלקות ניהול, ועבודה עם מבני נתונים משותפים בין תהליכים. כל בעיה הובילה להבנה עמוקה יותר של המערכת ושל היכולת שלי להתמודד איתה.

במהלך העבודה רכשתי הרבה כלים שימושיים שאני מתכוון לקחת איתי לפרויקטים הבאים. למדתי איך לפרק בעיה מורכבת לתת־בעיות ברורות, איך לבדוק את המערכת בצורה מתודית, ואיך לבנות קוד שיכול להתפתח ולא להיתקע בשלב מסוים. כמו כן, הבנתי את החשיבות של בניית מערכת עם מבנה גמיש – כזה שאפשר להרחיב אותו בקלות בלי לפרק את מה שכבר עובד.

במבט לאחור, יש הרבה שהייתי עושה אחרת – בראש ובראשונה, הייתי מתחיל בתכנון של כל המערכת על הנייר: אילו רכיבים קיימים, איך הם מדברים זה עם זה, ואיך נראית זרימת הנתונים. בנוסף, הייתי משקיע מוקדם יותר ביצירת ממשק גרפי – כי ברגע שהיה ממשק, היה לי הרבה יותר קל להבין איך המערכת מתנהגת בפועל. אם היו לי עוד משאבים, הייתי מוציא את המערכת מהרשת הפנימית, ושכבות מתקדמות יותר של ניתוח ואבטחה. בנוסף, הייתי מוסיף על הטכנולוגיה הקיימת עוד פיצ׳רים לא טכנלוגיים, כמו ניהול משתמשים בצורה גרפית.

העבודה על הפרויקט הזה חידדה לי את החשיבות של למידה עצמאית, של תכנון מערכת בקנה מידה רחב, ושל עבודה עם כלים שמשלבים בין צד שרת לצד משתמש. זו הייתה הזדמנות אמיתית להבין איך לבנות מערכת שלמה – כזו שלא רק עובדת, אלא גם ניתנת לשימוש, לתחזוקה ולהרחבה.

# **ביבליוגרפיה**

**ספר רשתות מחשבים:** רוזנביו ע׳, גונן ב׳ והוד ש׳. **(**2020)*. המרכז הישראלי לסייבר, והמרכז לחינוך סייבר*

<https://data.cyber.org.il/networks/networks.pdf>

**ARP Spoofing:** Whalen, S. (2001). An introduction to ARP spoofing. SANS Institute InfoSec Reading Room, 1-15. <https://www.sans.org/reading-room/whitepapers/threats/introduction-arp-spoofing-853>

**Network Security Architecture:** Stallings, W., & Brown, L. (2018). Computer security: Principles and practice (4th ed.). Pearson Education.

**DNS Poisoning:** Kaminsky, D. (2008). It's the end of the cache as we know it. Black Hat USA 2008. <https://www.blackhat.com/presentations/bh-jp-08/bh-jp-08-Kaminsky/BlackHat-Japan-08-Kaminsky-DNS08-BlackOps.pdf>

**HTTP Security:** Rescorla, E. (2001). SSL and TLS: Designing and building secure systems. Addison-Wesley Professional.

**Alternative DNS Security Source:** Arends, R., Austein, R., Larson, M., Massey, D., & Rose, S. (2005). DNS security introduction and requirements (RFC 4033). Internet Engineering Task Force. <https://tools.ietf.org/html/rfc4033>

# **נספחים**