

## חלק ראשון

### סעיף א

#### **א. דרך יצירת קובץ CSV עם הודעות שכבת היעשום**

בשלב הראשון, יצרנו קובץ הנתונים באמצעות בינה מלאכותית והוא מייצג את המידע שאוטו אנו רוצים להעביר ברשות. הקוד עושה שימוש בספרייה pandas כדי לטעון ולעבד מידע זה.

יצרנו קובץ CSV המכיל את ה הודעות שכבת היעשום. המבנה של הקובץ כולל עמודות (בגון timestamp, msg\_id, message) טענו את הקובץ באמצעות הפקודה pd.read\_csv. שלב זה מדמה את **שכבת היעשום (Application Layer)** במודל OSI זהו המידע הטהור אותו משתמש רוצה להעביר, לפני שנוספו לו בותרות תקשורת כלשהן.

#### **ב. תיאור והסבר של תהליכי אריזה של מנוגט(Encapsulation)**

הקוד בונה ידנית את ה בותרות Headers (Headers) בעזרת הספרייה struct ומשתמש במחלקה RawTcpTransport .

תהליכי האריזה (Encapsulation) בוצע באופן ידני בקוד כדי להמחיש כיצד נבנית חבילה מידע:

##### **1. שכבת התעבורה:(Transport - TCP)**

הfonקציה build\_tcp\_header בקוד מייצרת את בותרת TCP. היא אורצת (באמצעות struct.pack) את פורט המקור ואת פורט היעד , מספרי הרץ (Flags), והדלים (Sequence Number). בדוגמה שלנו השתמשנו בדלים PSH, מה שמורה לצד השני לעבד את המידע מיידית.

##### **2. שכבת הרשת:(IP - Network)**

הfonקציה build\_ip\_header עוטפת את ה TCP Segment שיצרנו. היא מוסיפה את כתובת ה-IP של המקור והיעד (במקרה זה Localhost - 127.0.0.1) ומגדירה את הпрוטוקול ב-TCP

למעשה הקוד לוקח את ההודעה משכבה האפליקציה ועוטף אותה ב-header המתאים.ם.

## ג. תיאור והסבר של תהליך הלכידה (Capture)

1. ביעון שהקוד הוגדר לשלוח מכתובת 127.0.0.1 אל 127.0.0.1 בחרנו בתווך ה-. Loopback Wireshark להקשיב למשחק
2. צירפנו את קובץ ה CSV אל מחברת jupyter שקיבלנו.
3. הפעילנו את לכידת הפקודות ב-Wireshark
4. הריצנו את הקוד שיצר הדמיה של שליחת פקודות.
5. לבסוף כיבינו את לכידת הפקודות וניתחנו את הלכידה.

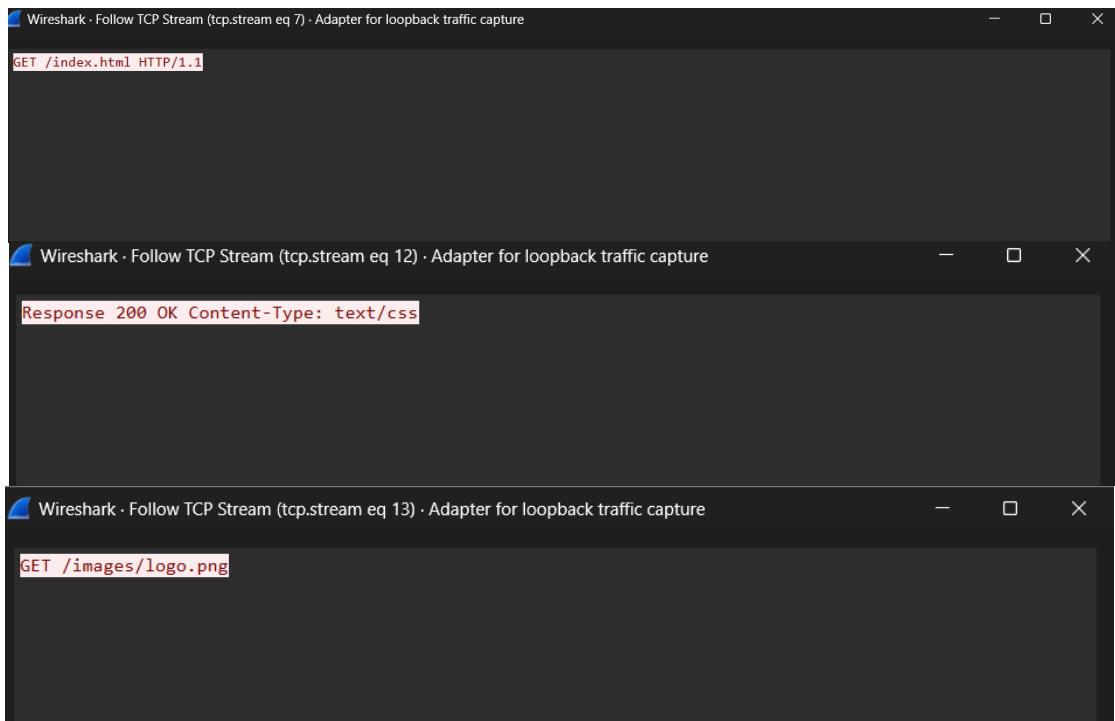
## ד. תיאור והסבר של תעבורת שנלקדה ב-Wireshark

בתמונות הבאות ניתן לראות את תעבורת הרשות שנוצרה על ידי סקריפט ה-`http_input11.py`.  
הסקריפט קרא את קובץ ה-CSV (02\_http\_input11.csv) ויצר עبور כל שורה  
פקטה ייחודית באמצעות Sockets Raw. ניתן להבחין כי תוכן ההודעות (Payload)  
זהה לטקסט המופיע בקובץ הקלט

### תוכן הקובץ:

E	D	C	B	A
timestamp	message	dst_port	src_port	app_protocol
				1
				2
09/12/2025	GET /index.html HTTP/1.1	80	51234	HTTP
				3
09/12/2025	Response 200 OK Content-Length: 1024	80	51235	HTTP
				4
09/12/2025	POST /api/login	8080	51236	HTTP
				5
09/12/2025	Response 302 Found Location: /home	80	51237	HTTP
				6
09/12/2025	GET /styles/main.css	80	51238	HTTP
				7
09/12/2025	Response 200 OK Content-Type: text/css	80	51239	HTTP
				8
09/12/2025	GET /images/logo.png	80	51240	HTTP
				9
09/12/2025	Response 200 OK Content-Type: image/png	80	51241	HTTP
				10
09/12/2025	GET /user/profile/123	8080	51242	HTTP
				11
				12
				13
				14
				15
				16
				17
				18
				19
				20

## תוכן הפקטו של בדנו:



ניתן להבחן כי עמודת **Source Port** משתנה מפקטה לפקטה בהתאם לקובץ הקלט, בעוד פורט היעד משתנה בהתאם לסוג הבקשה .

בנוסף הנתונים נקבעו מהקובץ במחוזות (**String**), עברו קידוד (**Encoding**) לבינארי, ושורשו לסוף ה-**TCP Header** כ-**Payload**.

## חלק שני

### סעיף א: הסבר כללי על המערכת ומבנה הקוד

המערכת היא אפליקציית צ'אט המבוססת על ארכיטקטורת (Client-Server) ומשתמשת ב프וטוקול TCP להעברת הודעות אמינה. השירות משתמש במתוך מרכז: כל ההודעות נשלחות מהלך לשרת, והשרת מנtab אותן לעד המתאים.

#### 1. צד השירות (server.py)

השירות בתוכן Python תור שימוש בספריות socket לתקשורת ו-threading לטיפול במקביליות.

##### מבנה הנתונים והניהול:

- השירות מחזיק מילון גלובלי בשם clients, הממחיה בין **שם המשתמש** (Key) לבין אובייקט ה-**Socket** הפעיל של (Value). מבנה זה מאפשר שליפת חיבור של משתמש ספציפי לצורך שליחת הודעה פרטית ברגע.

##### רכיבים מרכזיים בקוד השירות:

- אתחול והאזנה (start\_server):** הפונקציה יוצרת Socket מסוג TCP (SOCK\_STREAM) ומבצעת קישור (Bind) לכתובת 0.0.0.0 ולפורט 42069. השירות נבנש למצוות האזנה (listen) וממתין לחיבורים נוספיםים בולאה אינסופית. ברגע שהלקוח מתחבר (accept), השירות ייצור עבورو Thread (תהליכון) נפרד ו מעביר את הטיפול בו לפונקציה handle\_client. שיטה זו מבטיחה שהשירות לא ייחסם ו יוכל לשירות לקוחות במקביל.
- פרוטוקול ההתחברות (Handshake):** כאשר לקוח מתחבר, השירות שולח לו הודעה בקשה "LOGIN\_REQUEST". הלקוח בתגובה שולח את שם המשתמש שלו. השירות רושם את המשתמש במילון clients ו מעדכן את כל

שאר המחברים על ידי שליחת רשימת המשתמשים המעודכנת (USERS\_LIST).

- **ניתוב הודעות (handle\_client):** זהה הלוגיקה המרכזית הרצה עבור כל לקוחות. הפונקציה מאזינה להודעות כניסה. המערכת עובדת לפי פרוטוקול טקסטואלי מוסכם:
  - הודעות פרטיות מגיעות בפורמט ."TargetUsername:MessageContent"
  - השירות מפרק את ההודעה, בודק אם משתמש היעד (clients) קיים במיילו (TargetUsername)
  - אם היעד קיים השירות שולח לו את ההודעה בפורמטו ."SenderName: MessageContent"
  - אם היעד אינו מחובר נשלחת הודעה שגיאה חוזרת לשולח.
- **טיפול בניתוקים:** אם מתאפשרת הودעה ריקה (המעידה על סגירת חיבור) או מתרחשת שגיאה, השירות מסיר את המשתמש מהמיילון, סגור את ה-Socket ושולח עדכון לכל שאר המשתמשים שהליך התנתק.

## 2. צד הלקוח - (client.py) היבטי תקשורת

למרות שהליך כולל ממשק גרפי, ליבת התקשרות מופרדת ומנווהת במקביל ל-*gui*

- **יצירת החיבור:** הלקוח יוצר Socket ומחבר לבתוות ה-IP והפורט של השירות.
- **האזנה להודעות (Threading):** מיד לאחר החיבור הלקוח מפעיל תהליכון (Thread) נפרד המ裏ץ את הפונקציה receive\_messages. תהליכון זה אחראי להמתין להודעות מהשרת מוביל "להקפיא" את הממשק למשתמש.

• **עיבוד הودעות נכונות:** הלקוח מפענח את המידע המגיע מהשרת וմבדיל בין

סוגי הודעות:

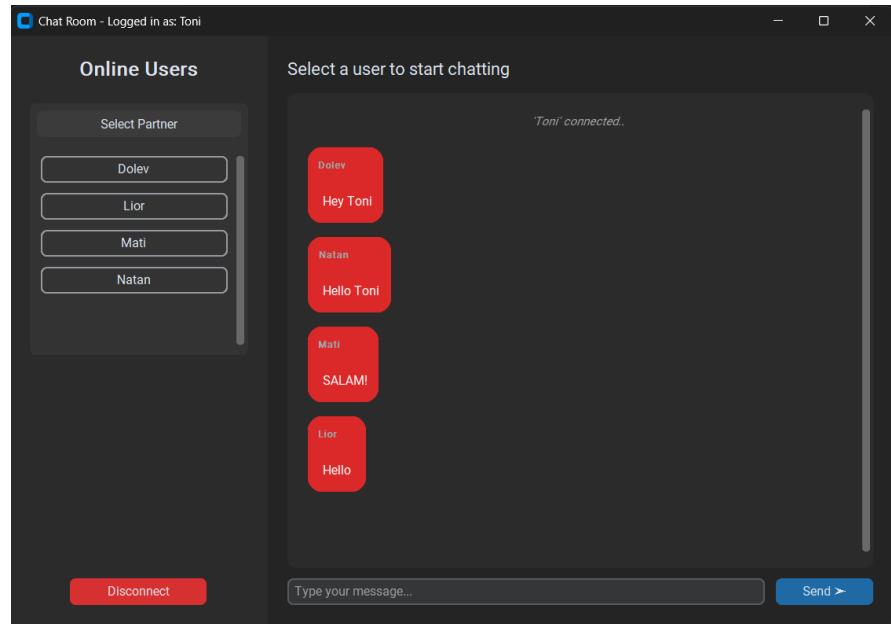
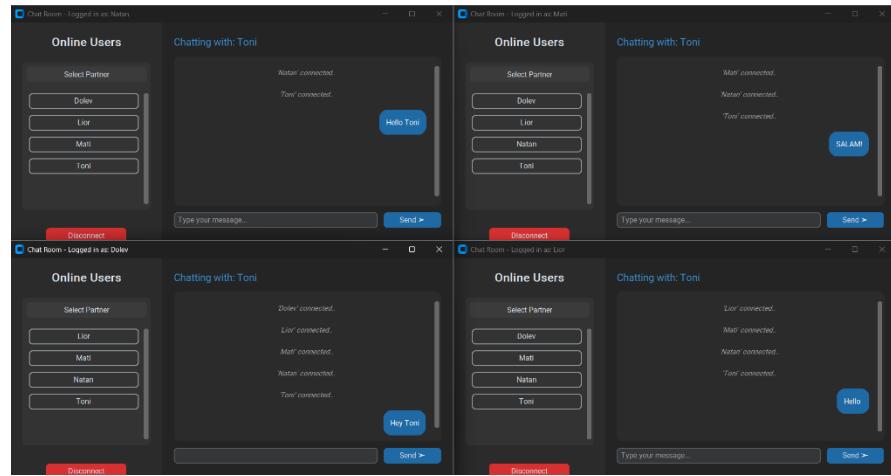
- . LOGIN\_REQUEST: ○  
עדכן רשימת המשתמשים המתחברים.
- . USERS\_LIST: ○  
הודעות צ'אט וריגילות מוצגות בחלון השיחה.

**סעיף ב- הוראות התקינה והרצה :**

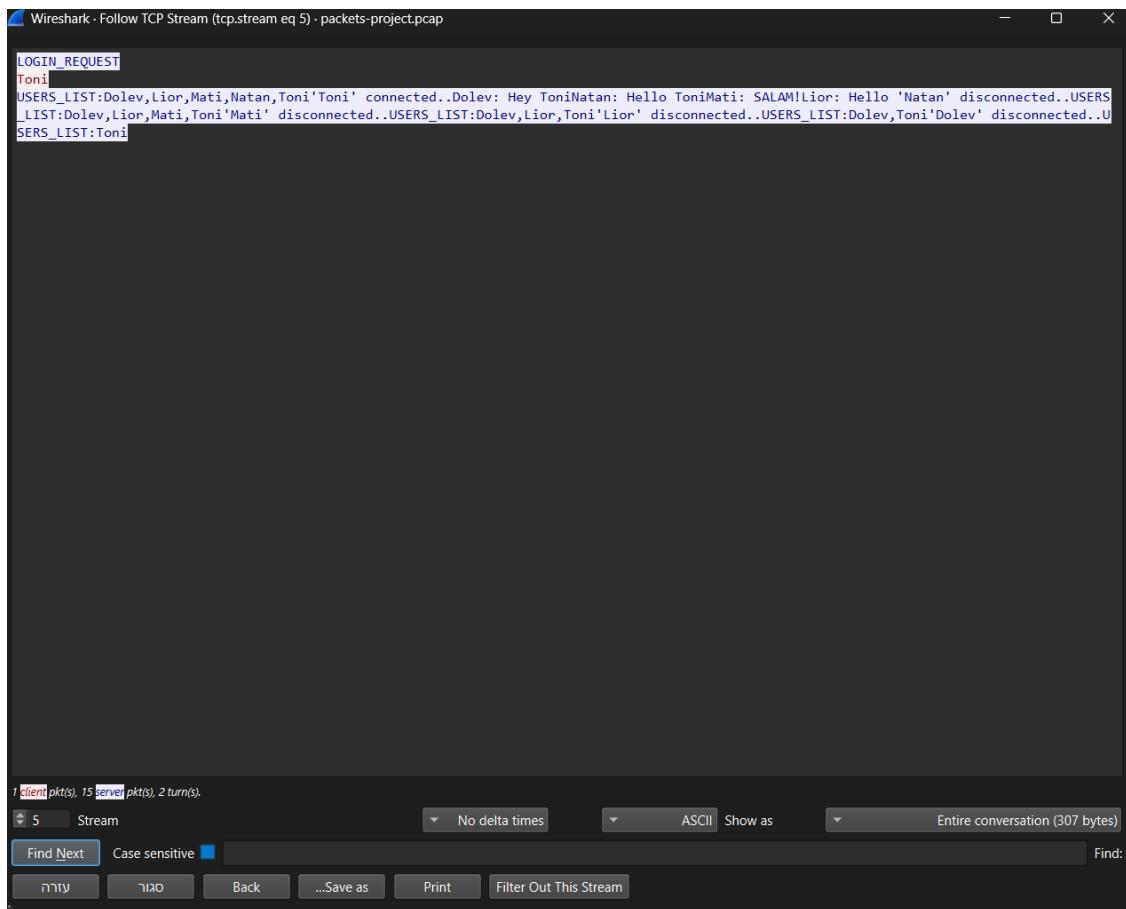
כל ההוראות ההרצה והתקינה הנדרשות מפורטוות בקובץ readme המצורף.

## סעיף ג+ד: דוגמאות לתקינות בפניות מוחזק' אט

בתמונה אלו אפשר לראות את השיטה המתועדת בין חמישה ללקוחות שונים ובנוסף מצורף בהגשת הפרויקט קובץ PCAP מה-Wireshark



## פיתוח ראשוני



בתמונה הנ"ל אפשר לראות את כל ההיסטוריה של הלcoil "טוני" במהלך כל שהותו בצד'אט (התחברות, התנטקות והודעות) ונפרט על כך עבשיו:

- שורה ראשונה והשנייה רואים את התחברות של טוני אל הצד'אט
- בשורה השלישית והרביעית רואים התחברות של שאר הלcoilות וההודעות שנשלחו אל טוני
- לבסוף רואים את התנטקות של הלcoil טוני מהצד'אט.

## ניתוח שכבות הרשת

```
Frame 419: Packet, 59 bytes on wire (472 bits), 59 bytes captured (472 bits) ▾
  Encapsulation type: NULL/Loopback (15)
  Arrival Time: Jan 6, 2026 15:46:49.165011000
    UTC Arrival Time: Jan 6, 2026 13:46:49.165011000 UTC
    Epoch Arrival Time: 1767707209.165011000
      [Time shift for this packet: 0.000000000 seconds]
      [Time delta from previous captured frame: 23.000 microseconds]
      [Time delta from previous displayed frame: 23.000 microseconds]
      [Time since reference or first frame: 3 minutes, 17.420057000 seconds]
        Frame Number: 419
        Frame Length: 59 bytes (472 bits)
        Capture Length: 59 bytes (472 bits)
          [Frame is marked: False]
          [Frame is ignored: False]
        [Protocols in frame: null:ip:tcp:data]
          Character encoding: ASCII (0)
            [Coloring Rule Name: TCP]
            [Coloring Rule String: tcp]
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1 ▾
  Version: 4 = .... 0100
  Header Length: 20 bytes (5) = 0101 ....
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT) ↴
    Total Length: 55
    Identification: 0xf0c1 (61633)
    Flags: 0x2, Don't fragment = .... .010 ↴
    Fragment Offset: 0 = 0000 0000 0000 0...
      Time to Live: 128
      Protocol: TCP (6)
    Header Checksum: 0x0000 [validation disabled]
      [Header checksum status: Unverified]
      Source Address: 127.0.0.1
      Destination Address: 127.0.0.1
      [Stream index: 2]
Transmission Control Protocol, Src Port: 42069, Dst Port: 62458, Seq: 289, Ack: 5, Len: 15 ▾
  Source Port: 42069
  Destination Port: 62458
  [Stream index: 5]
  [Stream Packet Number: 34]
  [Conversation completeness: Complete, WITH_DATA (47)] ↴
    [TCP Segment Len: 15]
    Sequence Number: 289 (relative sequence number)
    Sequence Number (raw): 1076949061
    [Next Sequence Number: 304 (relative sequence number)]
    Acknowledgment Number: 5 (relative ack number)
    Acknowledgment number (raw): 3708362092
    Header Length: 20 bytes (5) = .... 0101
    Flags: 0x018 (PSH, ACK) ↴
      Window: 255
      [Calculated window size: 65280]
      [Window size scaling factor: 256]
      Checksum: 0x4d1a [unverified]
      [Checksum Status: Unverified]
      Urgent Pointer: 0
      [Timestamps] ↴
      [SEQ/ACK analysis] ↴
      [Client Contiguous Streams: 1]
      [Server Contiguous Streams: 1]
      TCP payload (15 bytes)
        Data (15 bytes) ▾
        Data: 55534552535f4c4953543a546f6e69
        [Length: 15]
```

## הסבר כללי על השכבות ומה אפשר להסיק מהתמונה

### **שכבת האפליקציה:**

מופיעת השורה: (Data 15 bytes) זהו התוכן המקורי ש עבר ברשות. ניתן לראות את המידע ב-Data גולמי, ובאשר נמיר את הקוד הקסדצימלי לאנגלית קיבל את הודעה: "USERS\_LIST:Toni".

### **(Transport - TCP)**

(מודיע בשורה: Transmission Control Protocol) בין מנהלת "השיכחה" לבין התוכנות. אנו רואים שהשלוח הוא פורט 42069 והמקבל הוא פורט 62458. הדגמים Dolkim, מה שאומר שהמידע נדחף לאפליקציה מיד ולא עיבוביים. אורק המידע נטו (ללא כותרות) הוא 15 בתים.

### **(Network - IP)**

(מודיע בשורה: Internet Protocol Version 4) שכבה זו אחראית על הכתובות והניטוב. הפרטים החשובים כאן הם כתובות המקור (Source) וכתוות היעד (Destination), שתיהן מופיעות ב-127.0.0.1 (localhost). זה מאשר שהשלוח והמקבל הם אותו מחשב. בנוסף, השכבה מצינית שהפרוטוקול המוביל בתוכה הוא TCP.

## **שימוש במבנה מלאכותית**

בפרויקט זה נעזרנו במבנה מלאכותית באופן הבא:

1. יצרת קובץ `csv` לשאלת 1.

הפרומפט שבו השתמשנו: "אשmach שתעדזor ליצור קובץ `csv` שمدמה הודעות משבבת האפליקציה." (בנוסף צירפנו את הוראות יצרת קובץ `the-csv`)

2. הכוונה ליצירת ה-`UI` עבור הצ'אט.

הפרומפט שבו השתמשנו: "אני רוצה לבנות `UI` לפרויקט. אשmach שתכזון אותו שלב שלב איך לבנות אותו ובאיזה ספריות מומלץ להשתמש"

3. הכתנת קובץ `readme` (בעיקר בהוראות התקינה).

הפרומפט שבו השתמשנו: "אשmach שתעדזor לנסה מחדש את קובץ `readme` שיצרתי בצורה ברורה יותר"