

# חלק ראשון

## סעיף א

### א. דרך יצירת קובץ CSV עם הודעות בשכבת היישום

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

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

### ב. תיאור והסבר של תהליך אריזה של מנות (Encapsulation)

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

תהליך האריזה (Encapsulation) בוצע באופן ידני בקוד כדי להמחיש כיצד נבנית חבילת מידע:

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

הפונקציה build\_tcp\_header בקוד מייצרת את כותרת ה-TCP. היא אורזת (באמצעות struct.pack) את פורט המקור ואת פורט היעד, מספרי הרצף (Sequence Number), והדגלים (Flags). בדוגמה שלנו השתמשנו בדגלים PSH + ACK, מה שמורה לצד השני לעבד את המידע מיידית.

#### 2. שכבת הרשת: (Network - IP)

הפונקציה build\_ip\_header עוטפת את ה-TCP Segment שיצרנו. היא מוסיפה את כתובות ה-IP של המקור והיעד (במקרה זה 127.0.0.1 - localhost) ומגדירה את הפרוטוקול כ-TCP.

למעשה הקוד לוקח את ההודעה משכבת האפליקציה ועוטף אותה ב-header המתאימים.

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

1. כיוון שהקוד הוגדר לשלוח מכתובת 127.0.0.1 אל 127.0.0.1 בחרנו בתוך ה-Wireshark להקשיב לממשק Loopback.
2. צירפנו את קובץ ה-CSV אל מחברת jupyter שקיבלנו.
3. הפעלנו את לכידת הפקטות ב-wireshark.
4. הרצנו את הקוד שיצר הדמיה של שליחת פקטות.
5. לבסוף כיבינו את לכידת הפקטות וניתחנו את הלכידה.

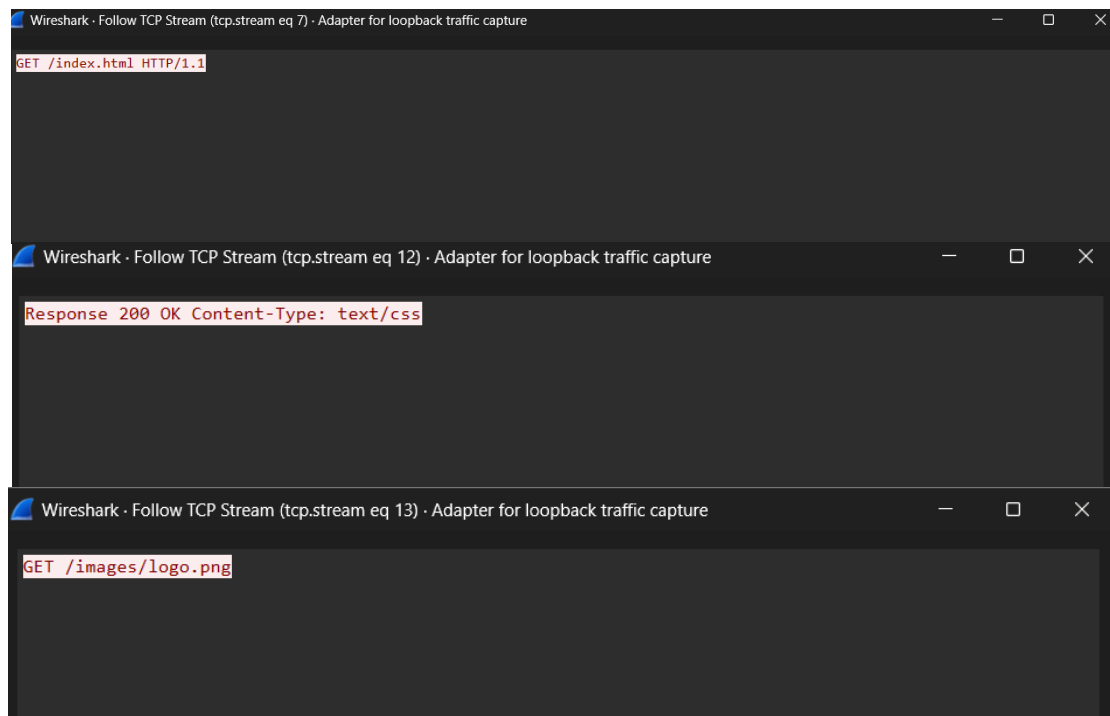
### ד. תיאור והסבר של תעבורה שנלקדה ב-Wireshark

בתמונות הבאות ניתן לראות את תעבורת הרשת שנוצרה על ידי סקריפט ה-Python. הסקריפט קרא את קובץ ה-(02\_http\_input11.csv) וייצר עבור כל שורה פקטה ייחודית באמצעות Raw Sockets. 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	
09/12/2025	GET /index.html HTTP/1.1	80	51234	HTTP	1
09/12/2025	Response 200 OK Content-Length: 1024	80	51235	HTTP	2
09/12/2025	POST /api/login	8080	51236	HTTP	3
09/12/2025	Response 302 Found Location: /home	80	51237	HTTP	4
09/12/2025	GET /styles/main.css	80	51238	HTTP	5
09/12/2025	Response 200 OK Content-Type: text/css	80	51239	HTTP	6
09/12/2025	GET /images/logo.png	80	51240	HTTP	7
09/12/2025	Response 200 OK Content-Type: image/png	80	51241	HTTP	8
09/12/2025	GET /user/profile/123	8080	51242	HTTP	9

## תוכן הפקטות שלכדנו:



ניתן להבחין כי עמודת ה-Source Port משתנה מפקטה לפקטה בהתאם לקובץ הקלט, בעוד פורט היעד משתנה בהתאם לסוג הבקשה . בנוסף הנתונים נקראו מהקובץ כמחרוזות (String), עברו קידוד (Encoding) לבינארי, ושורשרו 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 בפרוטוקול TCP להעברת הודעות אמינה. השרת משמש כמתווך מרכזי: כל ההודעות נשלחות מהלקוח לשרת, והשרת מנתב אותן ליעד המתאים.

#### 1. צד השרת (server.py)

השרת כתוב ב-Python תוך שימוש בספריות socket לתקשורת ו-threading לטיפול במקביליות.

#### מבנה הנתונים והניהול:

- השרת מחזיק מילון גלובלי בשם clients, הממפה 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 השרת:

- אתחול והאזנה (start\_server):** הפונקציה יוצרת socket מסוג TCP (SOCK\_STREAM) ומבצעת קישור (Bind) לכתובת 0.0.0.0 ולפורט 42069. השרת נכנס למצב האזנה (listen) וממתין לחיבורים נכנסים בלולאה אינסופית. ברגע שלקוח מתחבר (accept), השרת יוצר עבורו Thread (תהליכון) נפרד ומעביר את הטיפול בו לפונקציה handle\_client. שיטה זו מבטיחה שהשרת לא ייחסם ויוכל לשרת מספר לקוחות במקביל.
- פרוטוקול ההתחברות (Handshake):** כאשר לקוח מתחבר, השרת שולח לו הודעת בקשה "LOGIN\_REQUEST". הלקוח בתגובה שולח את שם המשתמש שלו. השרת רושם את המשתמש במילון ה-clients ומעדכן את כל

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

- **ניתוב הודעות (handle\_client):** זוהי הלוגיקה המרכזית הרצה עבור כל לקוח. הפונקציה מאזינה להודעות נכנסות. המערכת עובדת לפי פרוטוקול טקסטואלי מוסכם:
  - הודעות פרטיות מגיעות בפורמט  
"TargetUsername:MessageContent".
  - השרת מפרק את ההודעה, בודק אם משתמש היעד (TargetUsername) קיים במילון ה-clients.
  - אם היעד קיים השרת שולח לו את ההודעה בפורמט:  
"SenderName: MessageContent".
  - אם היעד אינו מחובר נשלחת הודעת שגיאה חזרה לשולח.
- **טיפול בניתוקים:** אם מתקבלת הודעה ריקה (המעידה על סגירת חיבור) או מתרחשת שגיאה, השרת מסיר את המשתמש מהמילון, סוגר את ה-Socket ושולח עדכון לכל שאר המשתמשים שהלקוח התנתק.

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

למרות שהלקוח כולל ממשק גרפי, ליבת התקשורת מופרדת ומנוהלת במקביל ל-gui

- **יצירת החיבור:** הלקוח יוצר Socket ומתחבר לכתובת ה-IP והפורט של השרת.
- **האזנה להודעות (Threading):** מיד לאחר החיבור הלקוח מפעיל תהליכון (Thread) נפרד המריץ את הפונקציה receive\_messages. תהליכון זה אחראי להמתין להודעות מהשרת מבלי "להקפיא" את הממשק למשתמש.

- **עיבוד הודעות נכנסות:** הלקוח מפענח את המידע המגיע מהשרת ומבדיל בין

סוגי הודעות:

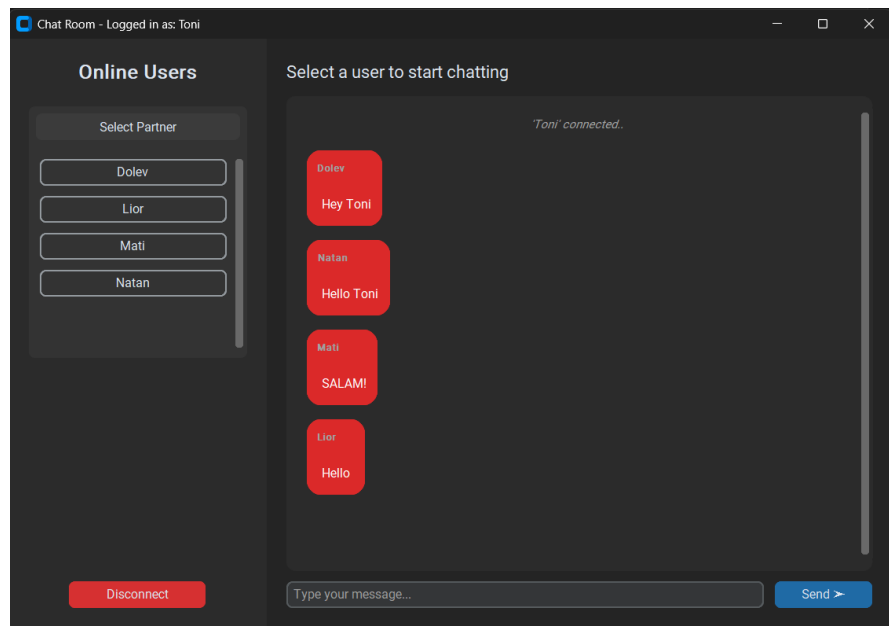
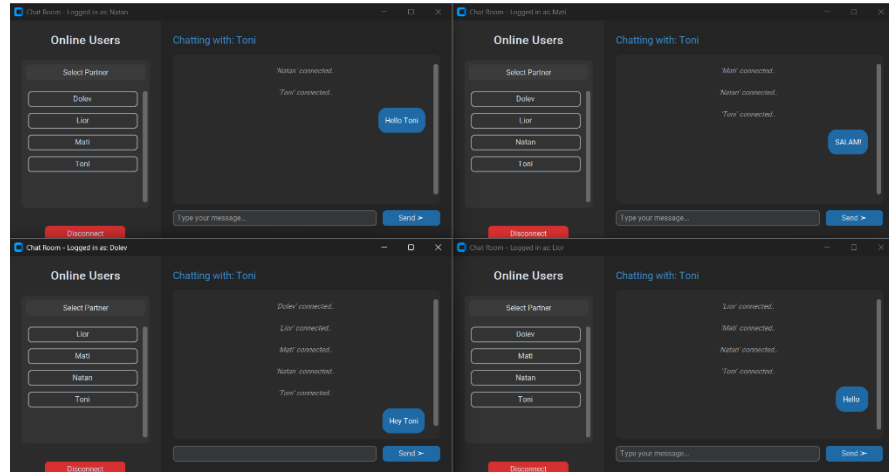
- LOGIN\_REQUEST: שליחת שם המשתמש.
- USERS\_LIST: עדכון רשימת המשתמשים המחוברים.
- הודעות צ'אט רגילות מוצגות בחלון השיחה.

### **סעיף ב- הוראות התקנה והרצה :**

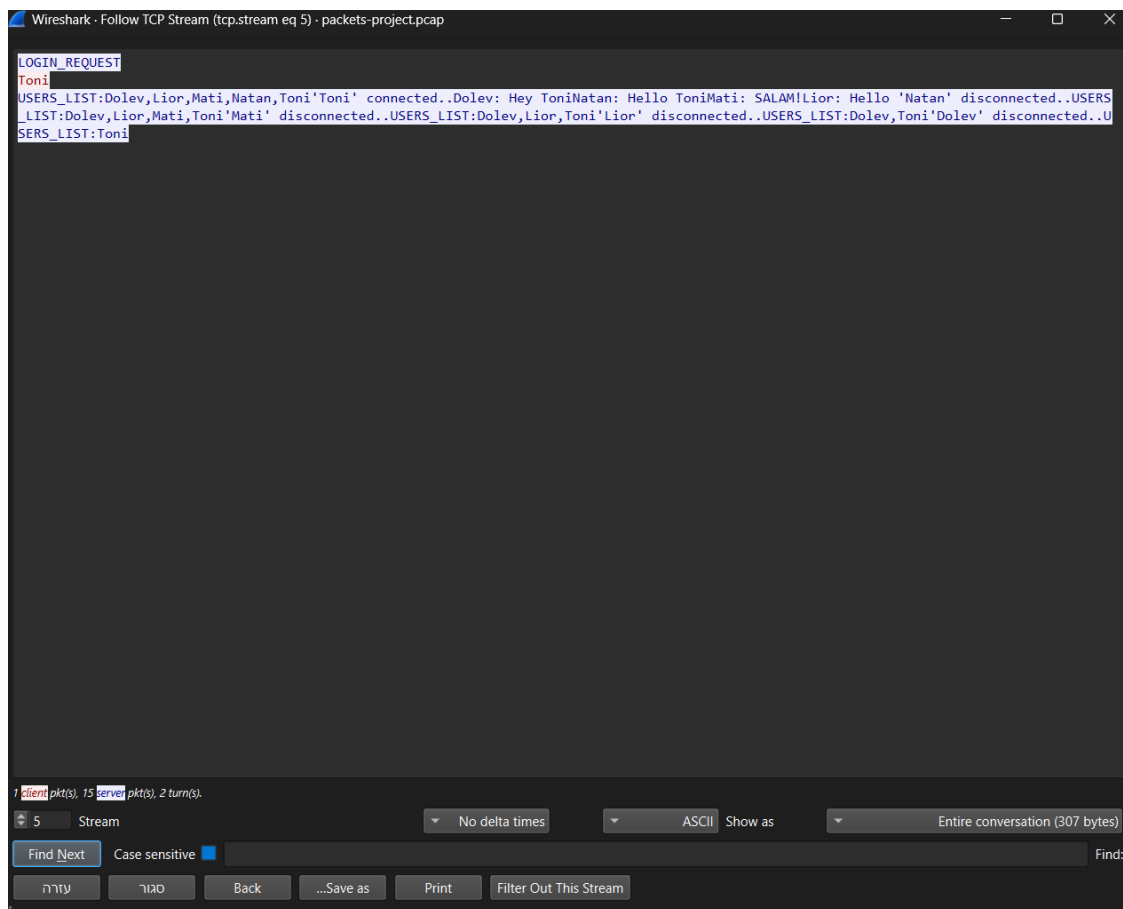
כל ההוראות ההרצה וההתקנה הנדרשות מפורטות בקובץ ה-readme המצורף.

## סעיף ג+ד: דו"ח ניתוח פקטות מהצ'אט

בתמונות אלו אפשר לראות את השיחה המתועדת בין חמישה לקוחות שונים ובנוסף מצורף בהגשת הפרויקט קובץ ה PCAP מה- Wireshark



## ניתוח ראשוני



בתמונה הנ"ל אפשר לראות את כל ההיסטוריה של הלקוח "טוני" במהלך כל שהותו בצ'אט ( התחברות, התנתקות והודעות) ונפרט על כך עכשיו:

- שורה ראשונה והשנייה רואים את ההתחברות של טוני אל הצאט
- בשורה השלישית והרביעית רואים התחברות של שאר הלקוחות והודעות שנשלחו אל טוני
- לבסוף רואים את התנתקות של הלקוח טוני מהצאט.



## ניתוח שכבות הרשת

Frame 419: Packet, 59 bytes on wire (472 bits), 59 bytes captured (472 bits) ▾
Encapsulation type: NULL/Loopback (15)
שטען רגיל ירושלים 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 מהתמונות

### **שכבת האפליקציה:**

מופיעה השורה: Data (15 bytes) זהו התוכן הממשי שעבר ברשת.  
ניתן לראות את המידע כ-Data גולמי, וכאשר נמיר את הקוד ההקסדצימלי לאנגלית  
נקבל את ההודעה: "USERS\_LIST:Toni".

### **שכבת התעבורה (TCP - Transport)**

(מופיע בשורה: Transmission Control Protocol) כאן מנוהלת "השיחה" בין  
התוכנות. אנו רואים שהשולח הוא פורט 42069 והמקבל הוא פורט 62458. הדגלים  
PSH, ACK, דולקים, מה שאומר שהמידע נדחף לאפליקציה מיד וללא עיכובים. אורך  
המידע נטו (ללא כותרות) הוא 15 בתים.

### **שכבת הרשת (IP - Network)**

(מופיע בשורה: Internet Protocol Version 4) שכבה זו אחראית על הכתובות  
והניתוב. הפרטים החשובים כאן הם כתובת המקור (Source) וכתובת היעד  
(Destination), ששתיהן מופיעות כ-127.0.0.1 (Localhost). זה מאשר שהשולח  
והמקבל הם אותו מחשב. בנוסף, השכבה מציינת שהפרוטוקול המוכל בתוכה הוא  
TCP.

## שימוש בבינה מלאכותית

בפרויקט זה נעזרנו בבינה מלאכותית באופן הבא:

1. יצירת קובץ csv לשאלה 1.

הפרומפט שבו השתמשנו: "אשמח שתעזור לי ליצור קובץ csv שמדמה הודעות משכבת האפליקציה." (בנוסף צירפנו את הוראות יצירת קובץ ה-csv)

2. הכוונה ליצירת ה-GUI עבור הצ'אט.

הפרומפט שבו השתמשנו: "אני רוצה לבנות GUI לפרויקט. אשמח שתכונן אותי שלב שלב איך לבנות אותו ובאיזה ספריות מומלץ להשתמש"

3. הכנת קובץ readme (בעיקר בהוראות התקנה).

הפרומפט שבו השתמשנו: "אשמח שתעזור לי לנסח מחדש את קובץ ה readme שיצרתי בצורה ברורה יותר"