

חלק 1: אריזה ולכידת מנות

a. דרך יצירת קובץ CSV:

בחלק זה יצרנו קובץ CSV ע"י לכידה מ-WireShark המכילה הודעות בשכבת היישום בין מחשב לבין חיבור הרשת האלחוטית של המכון הטכנולוגי חולון. במהלך התפיסה ניתן לזהות בקשות בין המחשב לבין רכיבים נוספים על אותה רשת להזדהות ובנוסף גם בדיקות של המחשב על מנת לוודא את המשכיות החיבור שלו לרשת. עבור קובץ ה-CSV עשינו שימוש במספר כותרים על מנת שהקובץ יעבוד עם המחברת שסופקה לנו

1. Msg_id - מספור של ההודעות באופן יחסי לתחילת הלכידה.
2. Timestamp - ספירה יחסית של זמן בהתאם לתחילת תהליך הלכידה.
3. App_protocol - זה הוא הפרוטוקול בשכבת היישום שנעשה בו שימוש עבור הבקשה.
4. Message - תוכן הבקשה שמתארת מה בעצם רכיב ביקש מרכיב אחר.
5. Src_post - פורט המקור ממנה הבקשה הגיעה ברכיב השולח.
6. Dst_port - פורט היעד אליו מגיעה הבקשה ברכיב המקבל.

b. תיאור תהליך הכימוס:

1. **שכבת האפליקציה** - שכבה זו מכינה בקשה עבור שליחה המכילה את תוכן ההודעה לדוגמא: GET /ssdp/device-desc.xml HTTP/1.1
2. **שכבת התעבורה**: מערכת ההפעלה לוקחת את ההודעה שהתקבלה משכבת האפליקציה ומוסיפה לו כותרים של TCP. המטרה: לדעת באיזה פורט להעביר את ההודעה ובאיזה פורט מצופה לקבל את התגובה. כעת אנחנו מוסיפים: src:64409, dst:8008. האות המתקבל משלב 2 נראה כך:
[HTTP Data] + [TCP Header (Src: 64409 | Dst: 8008)]
בשלב זה החבילה כבר נראית יותר כאות (segment).
3. **שכבת הרשת** - שכבה זו מוסיפה על החבילה את כתובת היעד של ההודעה ע"י הוספת כתובת מקור ובנוסף כתובת יעד דבר שיגרום לחבילה להראות כך:
[IP header (source: 127.0.0.1,)] + [transport layer header] + [app layer data]
[(destination: 127.0.0.1
4. **שכבת הקו** - שכבה זו מטפלת בתקשורת בין רכיבים פיזיים על אותה רשת מקומית והכוונה של בקשות ליעד הרצוי. בנוסף בשכבה זו יש טיפול בתקלות שונות כמו FCS-בדיקת סדר מסגרות וגם כן בקרת זרימה.
5. **השכבה הפיזית** - שכבה זו מכילה את כל העצמים הפיזיים כגון: כבלי נחושת, סיבים אופטיים ותקשורת אלחוטית. תפקידה הוא להעביר את המידע מחוץ ליחידת התקשורת של משתמש הקצה לפי פורמט התקשורת הפיזיקלי המתאים.

c. תיאור מפורט על תהליך לכידת מידע בWireshark :

בניסוי זה, השתמשנו במחברת Jupyter שסופקה לנו כחלק ממטלת הקורס, במטרה לבצע תהליך **Packet Crafting** (בניית חבילות ידנית). התהליך מדגים למעשה את מודל 5 השכבות של TCP/IP כפי שניתחנו בתיאוריה, כאשר כל שלב בקוד מדמה פעולת "עטיפה" (Encapsulation) המתבצעת בדרך כלל על ידי מערכת ההפעלה.

להלן שלבי הניסוי והממצאים:

שלב 1: שכבת היישום - טעינת המידע

הפעולה: טענו את קובץ csv וחילצנו את המידע הגולמי (Payload). בשלב זה המידע הוא טקסט נקי (לדוגמא: GET), ללא המידע המלא לתעבורה בשלב זה.

שלב 2: תהליך הכימוס (שכבות תעבורה ורשת) - בניית הכותרות בקוד

במקום לתת למחשב לנהל את התקשורת, השתמשנו בספריית struct בפייתון כדי ליישם את הלוגיקה של השכבות התחתונות:

שלב 3: שכבת התעבורה:

הפונקציה build_tcp_header יצרה את כותרת ה-TCP. בעוד שבמקור נלכדו פורטים אחרים, המחברת **חייבה שימוש בפורט יעד 12345** לצורך הניסוי. כפי שניתן לראות בצילום המסך מ-Wireshark, המנה אכן נלכדה עם פורט יעד זה, מה שמוכיח שהגדרת הפורט במחברת עברה בהצלחה את כל תהליך הכימוס עד להזרקה לרשת.

שלב 4: שכבת הרשת:

הפונקציה build_tcp_header עטפה את מקטע ה-TCP בכותרת IP. הגדרנו ידנית את כתובת המקור והיעד כ-127.0.0.1.

שלב 5: שכבות הערוץ והפיזית - השידור והלכידה

מכיוון שהרצנו את הניסוי על windows, המחברת השתמשה בScapy כדי להזריק את החבילה ישירות לממשק ה- Loopback.

ניתוח ממצאים

כפי שניתן לראות בתמונות המצורפות בעמוד הבא, הצלחנו במשימה לקלוט את כלל החבילות. עם זאת, זיהינו מאפיין ייחודי למימוש של שכבה 2 בניסוי זה : במקום לראות כותרת Ethernet (עם כתובת MAC), אנו רואים בתמונת לכידת הבקשה רצף בבסיס HEX את הרצף הבא 00 00 00 02. זהו Loopback Header, המעיד על כך שהשידור לא יצא פיזית לכבל רשת אלא נשאר בגבולות המעבד (השכבה הפיזית). במילים אחרות האות לא יצא מגבולות המכשיר אלא נשאר בסביבה הפרטית שלנו. בנוסף, מכיוון שבניסוי לא מימשנו את התחום אחריות של שכבה 2, אז המימוש נעשה על ידי מערכת ההפעלה/Scapy בזמן השידור.

תיעוד ההודעות שנשלחו דרך המחברת:

```
# Preview packet structure
src_ip = '127.0.0.1'
dst_ip = '127.0.0.1'
src_port = random.randint(1024, 65535)
dst_port = 12345
payload = b'Hello Packet (preview)'
pkt_preview = build_ip_header(src_ip, dst_ip, 20 + len(payload)) + build_tcp_header(src_ip, dst_ip, src_port, dst_port, payload) + payload
hexdump(pkt_preview)

✓ [10] 12ms

0000  45 00 00 3e d4 4b 00 00 40 06 a8 6c 7f 00 00 01  E..>.K..@..l....
0010  7f 00 00 01 5a 0c 30 39 5f 5e 67 cd 00 00 00 00  ....Z.09^g.....
0020  50 02 ff ff 4c 86 00 00 48 65 6c 6c 6f 20 50 61  P...L...Hello Pa
0030  63 6b 65 74 20 28 70 72 65 76 69 65 77 29      cket (preview)
```

תיעוד ההודעות שנשלחו בצורתם הגולמית:

Destination Address	Source Address	Destination Port	Source Port Length	Destination	Source
127.0.0.1	127.0.0.1	12345	5153 79	127.0.0.1	127.0.0.1
127.0.0.1	127.0.0.1	5153	12345 44	127.0.0.1	127.0.0.1
127.0.0.1	127.0.0.1	12345	5153 79	127.0.0.1	127.0.0.1
127.0.0.1	127.0.0.1	5153	12345 44	127.0.0.1	127.0.0.1
127.0.0.1	127.0.0.1	12345	5153 65	127.0.0.1	127.0.0.1
127.0.0.1	127.0.0.1	5153	12345 44	127.0.0.1	127.0.0.1
127.0.0.1	127.0.0.1	12345	5153 79	127.0.0.1	127.0.0.1
127.0.0.1	127.0.0.1	5153	12345 44	127.0.0.1	127.0.0.1
127.0.0.1	127.0.0.1	12345	5153 65	127.0.0.1	127.0.0.1
127.0.0.1	127.0.0.1	5153	12345 44	127.0.0.1	127.0.0.1
127.0.0.1	127.0.0.1	12345	5153 60	127.0.0.1	127.0.0.1
127.0.0.1	127.0.0.1	5153	12345 44	127.0.0.1	127.0.0.1
127.0.0.1	127.0.0.1	12345	5153 60	127.0.0.1	127.0.0.1
127.0.0.1	127.0.0.1	5153	12345 44	127.0.0.1	127.0.0.1
127.0.0.1	127.0.0.1	12345	5153 60	127.0.0.1	127.0.0.1
127.0.0.1	127.0.0.1	5153	12345 44	127.0.0.1	127.0.0.1
127.0.0.1	127.0.0.1	12345	5153 60	127.0.0.1	127.0.0.1
127.0.0.1	127.0.0.1	5153	12345 44	127.0.0.1	127.0.0.1
127.0.0.1	127.0.0.1	12345	5153 60	127.0.0.1	127.0.0.1

תיעוד לכידה של הודעה באWireshark:

0000 02 00 00 00 45 00 00 4b 00 01 00 00 40 06 7c aaE K ...@ |
0010 7f 00 00 01 7f 00 00 01 14 21 30 39 00 00 00 00!09....
0020 00 00 00 00 50 18 20 00 6c 11 00 00 47 45 54 20P...l...GET
0030 2f 73 73 64 70 2f 64 65 76 69 63 65 2d 64 65 73 /ssdp/de vice-des
0040 63 2e 78 6d 6c 20 48 54 54 50 2f 31 2e 31 20 c.xml HT TP/1.1

Frame 13377: Packet, 79 bytes on wire (632 bits), 79 bytes captured (632 bits) on interface \Device\NPF_{...} Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 5153, Dst Port: 12345, Seq: 1, Ack: 1, Len: 35
Data (35 bytes)
Data: 474554202f737364702f6465766963652d646573632e786d6c20485454502f312e3120
[Length: 35]

חלק 2: יישום רשת צ'אט

1. הסבר כללי על המערכת

בנינו אפליקציית צ'אט רב-משתמשים שעובדת במודל **Client-Server** מעל פרוטוקול **TCP**. המערכת מאפשרת למספר לקוחות להתחבר בו-זמנית לשרת מרכזי, לראות מי מחובר ולשלוח הודעות פרטיות. בחרנו להשתמש ב-TCP כדי להבטיח שההודעות יגיעו בצורה אמינה, ללא שגיאות ובסדר שבו הן נשלחו.

2. מבנה הקוד וחלוקת קבצים

כדי לשמור על קוד מאורגן והפרדה בין שכבת התצוגה לשכבת הרשת, חילקנו את הפרויקט לשלושה קבצים:

- **השרת (Server.py)** : אחראי על ניהול המשתמשים. הוא שומר את כל הלקוחות במילון (Dictionary), מנתב הודעות בין משתמשים ומעדכן את כולם מי נמצא כרגע אונליין. השרת משתמש ב-**Threads** כדי לטפל בכל לקוח בנפרד בלי לחסום את האחרים.
- **לוגיקת הלקוח (client_logic.py)** : מנהלת את ה-Socket של המשתמש. היא אחראית על החיבור לשרת, שליחת הודעות והאזנה מתמדת להודעות נכנסות ברקע.
- **הממשק הגרפי (client_gui.py)** : נבנה בעזרת ספריית **CustomTkinter**.
 - 1. להקליד את שמו של המשתמש.
 - 2. לראות בצד המסך רשימת המשתמשים הפעילים.
 - 3. לשלוח הודעות ע"י הקלדת שם משתמש היעד או לחיצה עליו ברשימה.

3. תכונות מיוחדות וטיפול בשגיאות

במהלך הפיתוח הוספנו כמה פונקציות חכמות כדי לשפר את האפליקציה:

- **ניהול שמות:** המערכת מוודאת שלא יהיו שני אנשים עם אותו שם. היא יודעת לזהות ש-"Avi" ו-"avi" זה אותו משתמש (Case-Insensitive) וחוסמת כניסה כפולה.
- **מניעת הודעות עצמיות** : הוספנו הגנה בקוד שמונעת ממשתמש לשלוח הודעה לעצמו (Self: Message).
- **רשימת מחוברים דינאמית** : הרשימה בצד מתעדכנת אוטומטית כשמישהו נכנס או יוצא. דאגנו שהמשתמש לא יראה את עצמו ברשימה כדי למנוע בלבול.
- **שיפורי נוחות (UX)** : כשלוחצים על שם של מישהו ברשימה, השם שלו נכנס אוטומטית לשדה ההקלדה יחד עם נקודתיים ורווח (Name:), המאפשר להתחיל לכתוב מיד.

4. הוראות התקנה והרצה

1. יש להתקין את ספריית העיצוב על ידי הפקודה `pip install customtkinter`.
בטרמינל

```
pip install customtkinter
```

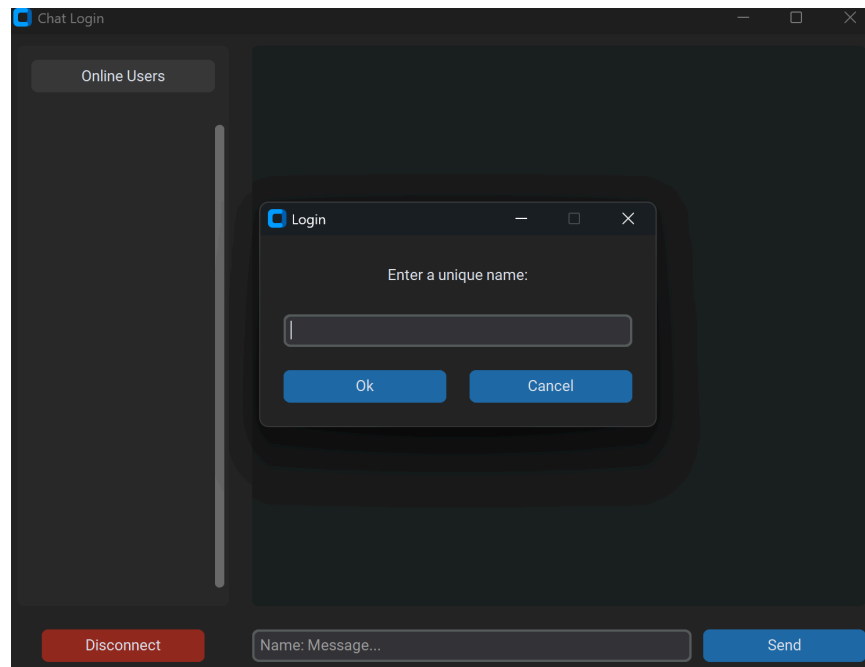
2. קודם כל מפעילים את השרת (`Server.py`). בעת הפעלת השרת נראה את השורה:
`Server is running on 127.0.0.1:55555`

```
Server is running on 127.0.0.1:55555
```

3. לאחר מכן מפעילים את הלקוח (`client_gui.py`) פעם אחת או יותר (לכל משתמש שרוצים להוסיף לצ'אט).

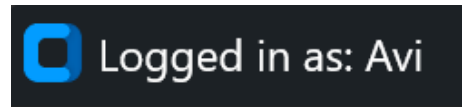
הערה: חשוב מאוד לשים לב כי הקבצים של `client_gui.py` ו `client_logic.py` חייבים להימצא באותה תיקייה.

4. לאחר מכן נקבל את המסך הבא:



בחלון זה נצטרך להקליד את שם המשתמש. כפי שציינו קודם לכן השם צריך להיות ייחודי לכל משתמש, אחרת המערכת לא תקלוט ותפלוט שגיאה. לצורך נוחות השם של המשתמש יהיה רשום בצד השמאלי העליון של החלון.

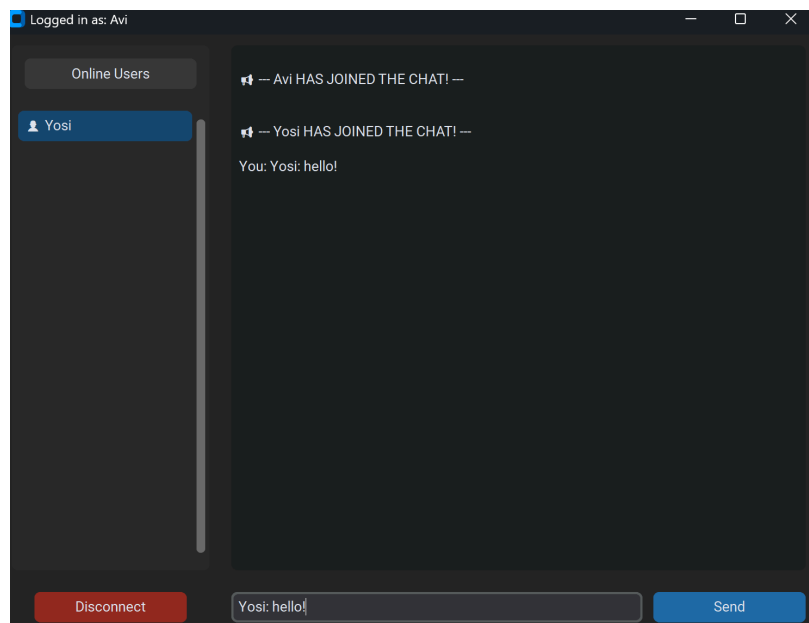
למשל אם אני משתמש בשם Avi אוכל לדעת שאני נמצא בחלון שלי כך:



וזוהו כרגע אתה מחובר.

5. על מנת לתקשר עם משתמשים נוספים יש ללחוץ על שמם בעמודת משתמשים מחוברים (online users) או לרשום את שמם בחלון ההודעה עם נקודותיים (מימין לשם) + רווח, וכעת יש להקליד את ההודעה.
לדוגמא:

User: {message}




6. על מנת להתנתק מהשרת יוכל המשתמש ללחוץ על כפתור הניתוק האדום (disconnect) או ללחוץ על כפתור האיקס למעלה מימין.

לאחר שמשתמש מתנתק, ניתן יהיה לראות בטרמינל של השרת ובצ'אט עצמו כי משתמש התנתק.

User chat

You: Yosi: hello!

 -- Yosi HAS LEFT THE CHAT! --

Server terminal

```
[CONNECTED] Avi from ('127.0.0.1', 62409)
[CONNECTED] Yosi from ('127.0.0.1', 51854)
[DISCONNECTED] Yosi
```

תהליך הלכידה בWireshark :

התמונות המצורפות ממחישות את הלכידה בWireshark. התמונות בתיאום.

בתמונות הצורפות ניתן לראות את תהליך הקמת הקשר (Handshake) בין אפליקציית הלקוח לשרת בפורט 55555.

Destination Address	Source Address	Destination Port	Source Port	Length	Destination	Source
127.0.0.1	127.0.0.1	55555	51773	56	127.0.0.1	127.0.0.1
127.0.0.1	127.0.0.1	51773	55555	56	127.0.0.1	127.0.0.1
127.0.0.1	127.0.0.1	55555	51773	44	127.0.0.1	127.0.0.1

בנוסף ניתן לזהות בבירור את שלושת המנות: ACK, SYN, ACK-SYN.

message
Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM [SYN] 55555 → 51773
Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM [SYN, ACK] 51773 → 55555
Seq=1 Ack=1 Win=65280 Len=0 [ACK] 55555 → 51773

127.0.0.1	127.0.0.1	55555	51773	44	127.0.0.1	127.0.0.1
-----------	-----------	-------	-------	----	-----------	-----------

בצילום המסך של Wireshark זיהינו מנת RST, ACK. מנה זו מעידה על ניתוק לא צפוי של הקשר (Unexpected Disconnection). כפי שנדרש במטלה, הקוד שלנו יודע לזהות מקרים כאלו – כאשר הלקוח נסגר בפתאומיות, השרת מקבל את ה-Reset, סוגר את הסוקט בצד שלו ומוחק את המשתמש במילון כדי לשמור על משאבי המערכת.

Seq=3 Ack=39 Win=0 Len=0 [RST, ACK] 55555 → 51773

השימוש ב-AI

במהלך העבודה על הפרוייקט עשינו שימוש ב-AI על מנת להבין כיצד יוצרים ממשק גרפי. הנושא הזה חדש עבורנו ולא היה לנו שימוש בו לפני כן. נושא נוסף ששאלנו את הבינה המלכותית לגביו הוא ניהול רשימת משתמשים דינאמית. דוגמאות לשאלות שלנו ל-AI היו:

"can you teach me how to make a UI object via the customtkinter library" -1
"also tell me how to make objects within objects

"?can you expand about customization within customtkinter" -2

"can you tell me about headers and titles" -3

in python while working on connections and sockets how do i keep it dynamic adding" -4
"and removing users and their sockets for every connection and disconnection

מבנה בסיס הנתונים שבחרנו

במהלך המטלה עשינו שימוש במבנה הנתונים מילון (dictionary) במילון זה שם המשתמש משמש בתור מפתח לחיפוש יעד והערכים המוחזרים מהמילון יהיו כתובת IP והפורט עליו יש לשלוח את ההודעה. שימוש במבנה זה מאפשר למצוא ולשלוח הודעות באופן כמעט מיידי (בסיבוכיות של $O(1)$) זאת מכיוון שהוא אינו צריך לעבור על רשימה ארוכה של שמות עבור כל הודעה שנכנסת למערכת או עבור כל כניסה נוספת של משתמש.