

חלק 1: אריזה ולכידת מנוגת

a. דרך יצירת קובץ CSV:

ב חלק זה יצרנו קובץ CSV ע"י לכידה Wireshark המכיליה הודעות בשכבה היחסם בין מחשב לבין חיבור הרשות האלחוטית של המ_connologique חולון. במהלך התפיסה ניתן לזרות בקשוט בין המחשב לבין רכיבים נוספים על אותה רשות להזדהות ובנוסף גם בדיקות של המחשב על מנת לוודא את המשיכויות החיבור שלו לרשות. עבור קובץ CSV עשינו שימוש במספר כוורות על מנת שהקובץ יעבד עם המחברת שספקה לנו

.1. **Msg_id** - מספור של ההודעות באופן ייחודי לתחילת הלכידה.

.2. **Timestamp** - ספירה יחסית של זמן בהתאם לתחילת תחילת הלכידה.

.3. **App_protocol** - זה הוא הпрוטוקול בשכבה היחסם שנעשה בו שימוש עבור הבקשה.

.4. **Message** - תוכן הבקשה שמתארת מה בעצם רכיב ביקש מרכיב אחר.

.5. **Src_port** - פורט המקור ממנו הבקשה הגיעה ברכיב השולח.

.6. **Dst_port** - פורט היעד אליו מגיעה הבקשה ברכיב המתקבל.

b. תיאור תחילת היכום:

1. **שכבת האפליקציה** - שכבה זו מכינה בקשה עבורה שליחת המכילה את תוכן ההודעה

לדוגמא: GET /ssdp/device-desc.xml HTTP/1.1

2. **שכבת התעבורה**: מערכת הפעלה לוקחת את ההודעה שהתקבלה משכנת האפליקציה ומוסיפה לו כוורות של TCP .

המטרה: לדעת באיזה פורט להעביר את ההודעה ובאיזה פורט מצופה לקבל את התגובה.

כעת אנחנו מוסיפים : dst:8008 , src:64409 .

האות המתתקבל משלב 2 נראה כך:

[TCP Header (Src: 64409 | Dst: 8008 | + [HTTP Data

.].(segment) .

3. **שכבת הרשות** - שכבה זו מוסיפה על החביליה את כתובות היעד של ההודעה ע"י הוספת

כתובת מקור ובנוסף כתובת יעד דבר שייגרום לחביליה להראות כך:

[[app layer data] + [transport layer header] (source: 127.0.0.1, [IP header] [destination: 127.0.0.1

][]]

4. **שכבת הקו** - שכבה זו מטפלת בתקשורת בין רכיבים פיזיים על אותה רשות מקומית והכוונה של בקשוט לעד הרצוי.

בנוסף בשכבה זו יש טיפול בתקלות שונות כמו FCS-בדיקה סדר מסגרות וגם-CN 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 ב-Wireshark**

בניסוי זה, השתמשנו במחברת `scapy` שסופקה לנו חלק ממילת הקוד, במטרה לבצע תהליך **Packet Crafting** (בנייה חבילות ידנית). התהליך מדגים למעשה את מודל 5 השכבות של IP/TCP כפי שניתחנו בתיאוריה, כאשר כל שלב בקוד מדמה פעולה "עטיפה" (Encapsulation) המבצעת בדרך כלל על ידי מערכת הפעלה.

להלן שלבי הניסוי והמצאים:

שלב 1: שכבת היישום - טעינת המידע

הפעולה: טענו את קובץ `cs` וחילצנו את המידע הגלמי (Payload). בשלב זה המידע הוא טקסט נקי (לדוגמא: GET), ללא המידע המלא לטעורה בשלב זה.

שלב 2: תהליכי הכינוס (שכבות תעבורת ורשות) - בניית הכותרות בקוד
במקרה לתחזק לנו את התקשרות, השתמשנו בספריית `struct` בפייתון כדי לישם את הלוגיקה של השכבות התחתונות:

שלב 3: שכבת התעborות:

הfonקציה `build_tcp_header` יוצרת את כוורתת TCP. בעוד שבמקור נלכנו פורטים אחרים, המחברת **חייבת שימוש בפורט יעד 12345** לצורך הניסוי. כפי שניתן לראות ביציומם המופיע Wireshark, המנה אכן נלכדה עם פорт יעד זה, מה שמכוכיח שהגדרת הפורט במחברת עברה בהצלחה את כל תהליכי הכינוס עד להזרקה לרשות.

שלב 4: שכבת הרשות:

הfonקציה `build_tcp_header` עטפה את מקטע ה-TCP בכוורת IP.
הגדרנו ידנית את כתובות המקור והיעד כ-127.0.0.1.

שלב 5: שכבות העוזץ והפיזית - השידור והלכידה

מכיוון שהרצינו את הניסוי על windows, המחברת השתמשה בעמוד Scapy כדי להזניק את החבילה **ישירות למסוק ה- Loopback**.

ניתוח ממצאים

כפי שניתן לראות בתמונות המצורפות בעמוד הבא, הצלחנו במשימה לקלוט את כל החבילות. עם זאת, זיהינו מאפיין ייחודי למימוש של שכבה 2 בניסוי זה :

במקרה לראות כוורת Ethernet (עט כתובות MAC), אנו רואים בתמונה לכידת הבקשה רצף בבסיס HEX את הרצף הבא 00 00 00 00 00 00.

זהו header `Loopback Header`, המעיד על כך שהשידור לא יצא פיזית לקבל רשות אלא נושא בגבולות המעבד (השכבה הפיזית).

במילים אחרות האות לא יצא מגבולות המכשיר אלא נושא בסביבה הפרטית שלנו. לבסוף, מכיוון שבניסוי לא מימושו את התוחום אחריות של שכבה 2, אז המימוש נעשה על ידי מערכת הפעלה 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 בצוותם הגלויים:

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 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

תיעוד לכידה של הودעה בwireshark:

```
0000  02 00 00 00 45 00 00 4b 00 01 00 00 40 06 7c aa  ... E..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 20  ... P.. 1...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
```

ame 13377: Packet, 79 bytes on wire (632 bits), 79 bytes captured (632 bits) on interface \Device\NPF_Loopback, id 0
Null/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 רשות 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 ושלוח הודעות פרטיות. בחרנו להשתמש ב-TCP כדי להבטיח שההודעות יגיעו בצורה אמינה, ללא שגיאות ובסדר שבו הן נשלחו.

2. מבנה הקוד וחלוקת קבצים

כדי לשמר על קוד מאורגן והפרדה בין שכבות התצוגה לשכבה הרשת, חילקו את הפרויקט לשלווה קבצים:

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

3. תכונות מיוחדות וטיפול בשגיאות

במהלך הפיתוח הוספנו כמה פונקציות חכמות כדי לשפר את האפליקציה:

- **ניהול שמות**: המערכת מודדת שלא יהיה שני אנשים עם אותו שם. היא יודעת לזהות ש-*"Avi"* ו-*"avi"* זה אותו משתמש (Case-Insensitive) וחוסמת כניסה כפולה.
- **מניעת הודעות עצמאיות**: הוספנו הגנה בקוד שמונעת משתמש משולח הודעה לעצמו (הודעה מהסוג **Self: Message**). אם המשתמש ינסה לעשות זאת, הוא יקבל הודעה שגיאה במערכת.
- **רשימת מחוברים דינמית** : הרשימה מצד מתעדכנת אוטומטית כשמשימושו נכנס או יוצא. דאגנו שהמשתמש לא יראה את עצמו ברשימה כדי למנוע בלבול.
- **SHIPORI נוחות (XU)** : כשלוחצים על שם של משתמש ברשימה, השם שלו נכנס אוטומטית לשדה ההקלדה יחד עם נקודותים ורווח (Name:), המאפשר להתחיל לכתוב מיד.

4. הוראות התקנה והרצה

- יש להתקין את ספריית העיבוד על ידי הפקודה `pip install customtkinter` בטרמינל.

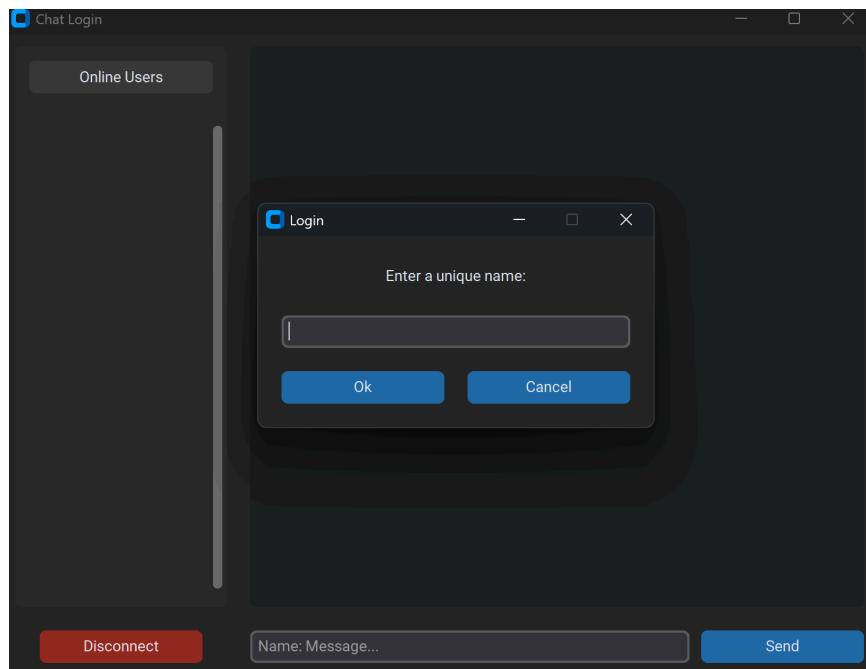
```
pip install customtkinter
```

- קדם 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
```

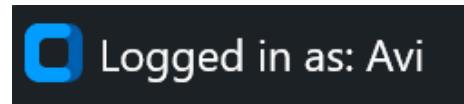
- לאחר מכן מפעילים את הלקוב (`client_gui.py`) פעם אחת או יותר (לכל משתמש שורצים להוסיף ליצ'ט).
הערה: חשוב מאד לשים לב Ci הקבצים של `client_logic.py` ו `client_gui.py` ו `client_logic.py` חיבים להימצא באותו תיקיה.

- לאחר מכן נקבל את המסר הבא:



בחילון זה נדרש 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 לצד השמאלי העליון של החילון.

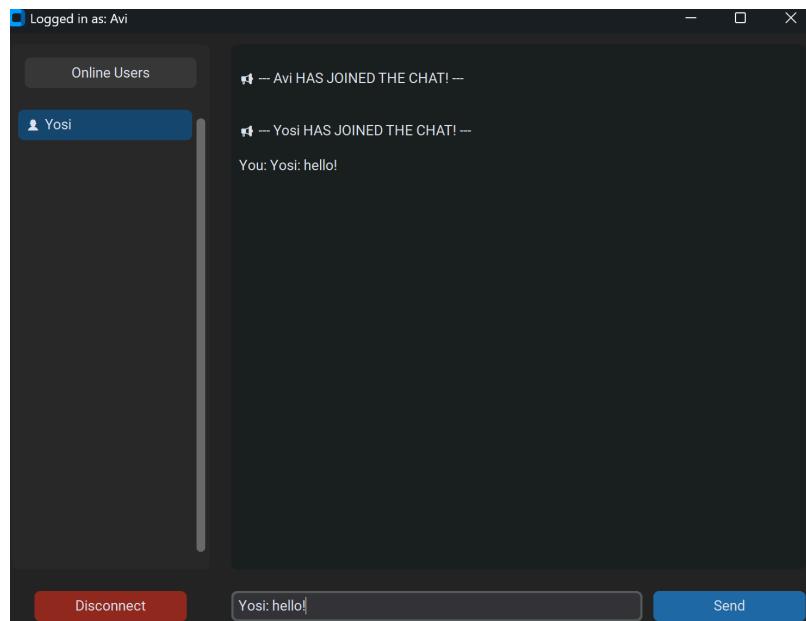
למשל אם אני משתמש בשם Avi אוכל לדעת שאני נמצא בחלון שלי כך:



זהו כרגע אתה מחובר.

5. על מנת לתקשר עם משתמשים נוספים יש להזוז על שם בעמודת משתמשים מחוברים (online users) או לרשום את שם בחלון הודעה עם נקודותים (מיין לשם) + רווח, וcut יש להקליד את הודעה.
לדוגמא:

User: {message}



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

לאחר שימוש מתנתק, ניתן יהיה לראות בטרמינל של השרת וביצ'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 ב-Wireshark :

התמונות המצורפות ממחישות את הלכידה Wireshark. התמונות בתיאום. בתמונה הצורפות ניתן לראות את תהליך הקמת הקשר (Handshake) בין אפליקציות 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 להזיהות 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

בצלום המסך של Wireshark זיהינו מנת RST, ACK, SYN. מנה זו מעידה על ניתוק לא צפוי של הקשר (Unexpected Disconnection). כפי שנדרש במתלה, הקוד שלנו יודיע להזיהות מקרים אלו – כאשר הלקוח נסגר בפתאומיות, השרת מקבל את ה-Reset, סגור את הsocket בצד שלו ומוחק את המשתמש במילון כדי לשמור על משאבי המערכת.

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

השימוש בAI

במהלך העבודה על הפרויקט עשינו שימוש ב-AI על מנת להבין כיצד יוצרים ממשק גרפי. הנושא זהה חדש עבורנו ולא היה לנו שימוש בו לפני כן. נושא נוסף ששאלנו את הבינה המילכנית לגבי הוא ניהול רשות משתמשים דינמית. דוגמאות לשאלות שלמו 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 המשמש משמש בתור מפתח לחיפוש יעד והערכים המוחזרים מהמילון יהיו כתובות פאץ' והפורט עליו יש לשלוח את הודעה.

שימוש במבנה זה מאפשר למצוא ולשלוח הודעות באופן כמעט מיידי (בסיסיות 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.