

---

# **פרויקט גמר**

## **כיתוח תעבורת בפרוטוקול TCP/IP**

**מוגש על ידי:** לירן דגן

**תעודת זהות:** 215609397

**שם מרצה:** ד"ר קוֹזֶכְבָּ אַנְדָּרִי

**קבוצת הרצאה:** 61305-3

---

---

## **תוכן עניינים**

---

<b>3</b>	<b>חלק ראשון</b>
3.....	מבוא .....
3.....	יצירת קובץ ה-CSV .....
6.....	תיאור והסביר אריזת הפאקטות .....
10.....	ניתוח התעבורה באמצעות WIRESHARK .....
<b>15</b>	<b>חלק שני</b>
15.....	מבוא .....
16.....	ארכיטקטורת הפרוייקט .....
17.....	6.1 אפליקציית השרת .....
21.....	6.2 אפליקציית הלקוון .....
24.....	הוראות התקינה והרצה .....
29.....	דוגמאות קלט ופלט .....

# חלק ראשון

## אריזת נתונים ולבידת מנוגות ב-Wireshark

### 1. מבוא

בחלק הראשון של הפרויקט נדרשים אנו להכין קובץ CSV ובו עמודות שונות לניטוח בסיסי של מידע ברמת היישום. מטרת העל של הקובץ היא להציג הודעות שמקבל המחשב מאפליקציות אחרות שונות בהתאם ל프וטוקול יישומי נבחר. (כדוגמת HTTP, DNS, SMTP)

הודעות אלו הן כאמור מתקבלות על גבי פרוטוקול בשכבה האפליקציה/יישום, ולשם הפרויקט בחרתי ליבא הודעות שהועברו בפרוטוקול DNS (Domain Name System).

### 2. יצירת קובץ ה-CSV

לקובץ ה-CSV שמצוף לו"ח הוספה שדota של פאקטות, בין היתר סוג החיבור (TCP/UDP), גודל הפאקטה וט. Request/Reply

במובן שהקובץ מכיל גם את השדות להם נדרשו בתיאור הפרויקט והם מכילים:

- msg\_id - מספר סידורי המיחיד כל הודעה
- app\_protocol - פרוטוקול ברמת האפליקציה אשר בחרנו הלו הוא DNS
- src\_port – מספר הפורט ממנו יצאתה הפאקטה מצד השולח
- dst\_port – מספר הפורט אליו הגיעו הפאקטה מצד המקלט
- time\_stamp – הרגע שבו הגיעו הפאקטה
- message – הודעה שנרצה להפיק מהפאקטה

הקובץ נוצר בשלהותו באמצעות לכידת תעבורה בתוכנת Wireshark והנתונים שבו הם למעשה פאקטות שהוקלטו ממכשירים ואפליקציות שהיו פועלים על המחשב שלי בעת ההקלטה. (אפשר למשל למצוא בקובץ התעבורת שהתקבלה הודעה משרת DNS של Google ושל Discord, שרצה על המחשב שלי בעת ההקלטה).

כדי לקבל את התעבורת שרציתי עבור פרוטוקול DNS השתמשתי בפילטר 'asap' כדי לסוג את הפאקטות הרלוונטיות. עבר כל שדה בפאקטה שרציתי להוסיף בעמודה ביצעתו לחיצה ימנית על העכבר ולחצתי על "apply as column". בך קיבלתי את כל השדות שרציתילייצא עבורי כל פאקטה. לאחר שאספתי מספיק תעבורת עצות את ההקלטה, לחצתי על תפריט File, ולאחר מכן על Export packet dissections ולאחר מכן על CSV as. בך יצאתי את הטבלה שנוצרה ב-Wireshark אל תוכנת Excel ומשם שמרתי את הטבלה בתור קובץ CSV שנשמר מקומית על המחשב.

עבור השדה message – הרכבתי אותו על ידי שרשור שdots:

( domain name | | is Response | | Transaction ID)

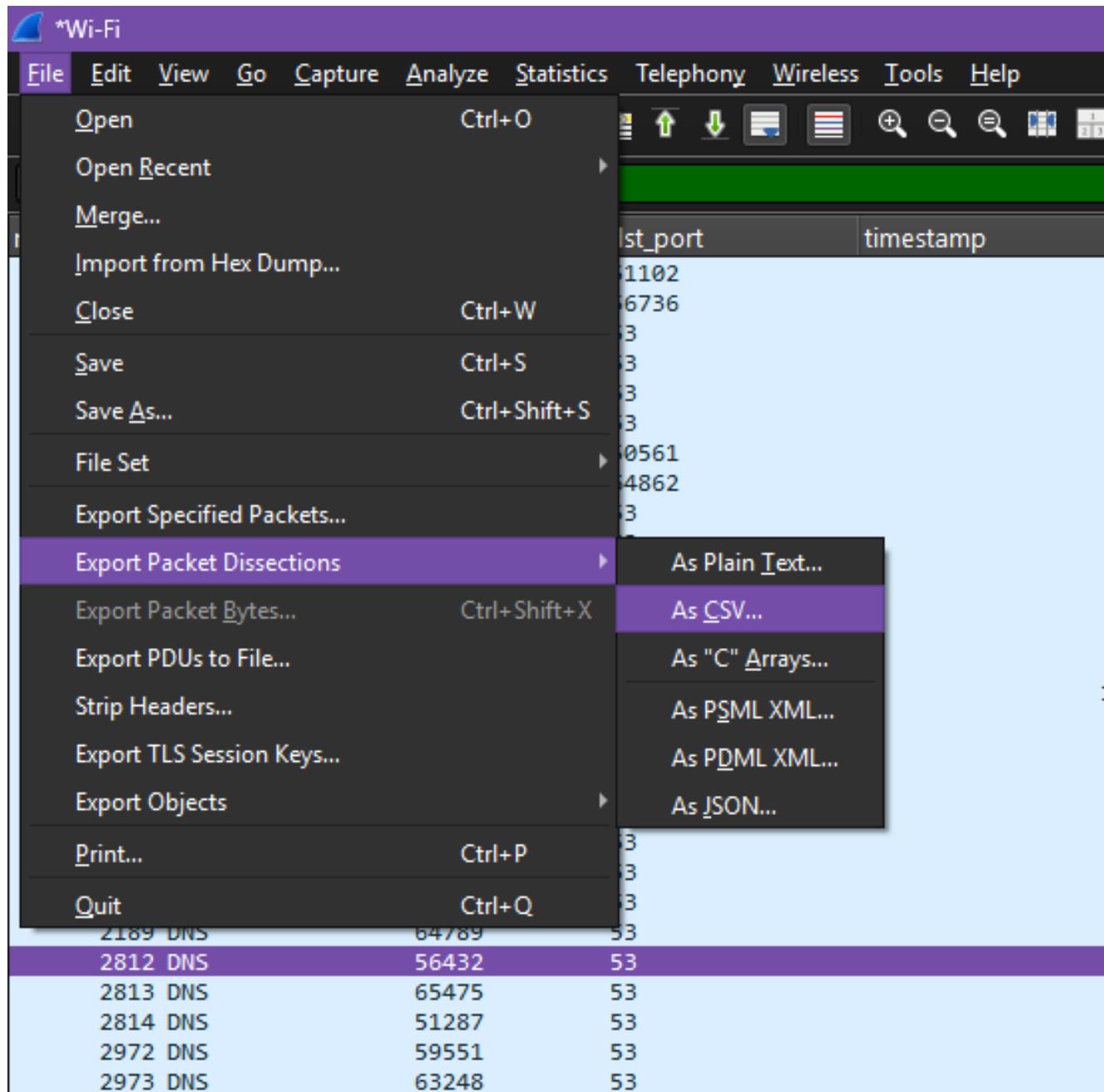
שאלות שודות שמופיעים את פרוטוקול DNS. Domain name משמעו שם הכתובת של האפליקציה, is משמעו האם הפקטה היא בקשה לשרת (אפס) או תגובה של השרת (אחד).

ID Transaction משמעו מספר סידורי של פרוטוקול DNS עבור השאלה.

### יצירת השדה message

Fields: dns.qry.name    dns.flags.response    dns.id		
destination Address	timestamp	message
1.1.1	0.000108	dns.google,0,0x03d8
1.1.1	0.000140	dns.google,0,0x0957
1.1.1	0.000336	dns.google,0,0xb42
1.1.1	0.000071	dns.google,0,0x162f
1.1.1	16.693004	dns.google,0,0x18d5
1.1.1	0.000361	dns.google,0,0x2100
1.1.1	0.895141	dns.google,0,0x3072
1.1.1	6.888908	dns.google,0,0x368f
1.1.1	29.761779	dns.google,0,0x3abc
1.1.1	1.014845	dns.google,0,0x41f8
	2.948365	dns.google,0,0x4261
	0.000235	dns.google,0,0x5954
1.1.1	0.000210	dns.google,0,0x60af
1.1.1	0.000120	dns.google,0,0x6d94
1.1.1	0.000145	dns.google,0,0x6f9b
	0.000129	dns.google,0,0x86f9
1.1.1	0.000216	dns.google,0,0x940d
	0.000334	dns.google,0,0x9473
	0.000388	dns.google,0,0x9cb5
	0.000256	dns.google,0,0xa3fa
	1.012652	dns.google,0,0xa7a7
1.1.1	0.000329	dns.google,0,0xe57d
1.1.1	0.882902	dns.google,0,0xf405
2.168.1.18	0.008372	dns.google,1,0x0957
2.168.1.18	0.000000	dns.google,1,0x162f
2.168.1.18	0.006692	dns.google,1,0x2100
2.168.1.18	0.001023	dns.google,1,0x368f
2.168.1.18	0.029216	dns.google,1,0x3abc
2.168.1.18	0.001168	dns.google,1,0x41f8
2.168.1.18	0.026362	dns.google,1,0x60af
2.168.1.18	0.066347	dns.google,1,0x6d94
2.168.1.18	0.006016	dns.google,1,0x6f9b
2.168.1.18	0.011840	dns.google,1,0x940d
2.168.1.18	0.001820	dns.google,1,0xe57d
2.168.1.18	0.020594	dns.google,1,0xf405
	14.580840	dns.msftncsi.com,0,0x3fc4
	0.100803	dns.msftncsi.com,0,0x3fc4
1.1.1	1.011470	dns.msftncsi.com,0,0x3fc4
	8.962245	

## יצוא הפקטוט אל קובץ CSV



### 3. תיאור והסבר אריזת הפקטוות

השלב השני בחלק זה הוא טיענת קובץ CSV אל מחברת הג'ופיטר המצורפת שבא קוד פיתון לייצור תובורה מקומית של הודעות שנלקחו מה-CSV.

רעיוןינו אנחנו משתמשים בכימוס (Encapsulation) במודל OSI כדי לאזרז את הפקטוות ולהעבירן. האריזה מרכיבה את הפקטה מהשכבה העליונה (האפליקציה) עד לשכבות התחתונות. נתאר באופן כללי את תהליך האריזה ולאחר מכן נפרט.

תחילה ההודעה נוצרת בשכבת האפליקציה (שלוחים והודעה, טקסט שנבחר למסור) ולאחר מכן מועברת אל שכבת התעבורה. במחברת, האפליקציה שולחת את הודעות ה-CSV שלנו (או הודעת דמה). ככלומר המחברת שולחת כל הודעה משירות ה-CSV אל שכבת התעבורה.

שכבת התעבורה עוטפת את ההודעה בתווך SEGMENT לפענוח על ידי שכבת התעבורה מצד המקלט. פרוטוקול TCP משייר לפאקטה כוורת TCP Header שבה הוא מכיל:

Checksum source port, destination port, flags שימושים נוספים. בין הערבים נוסף שדה sum שמשמש את הצד המקלט לפענוח שגיאות. שכבת התעבורה מעבירה את הפקטוות לשכבת הרשות. עד בה הודעת ה-CSV שלנו נעטפה בסגמנט של שכבת התעבורה.

שכבת הרשות עוטפת את הסגמנט בתווך Datagram IP. ככלומר הוא מוסיף כוורת IP Shmcilla: source IP, destination IP, version, TTL משדרת ה-CSV שלנו נעטפה בסגמנט, שנעטף כוורת IP.

רעיוןינו שכבת ה-IP מוסיפה מסגרת לפאקטה והיא השכבה האחורונה שעוטפת אותה לפני שהיא משודרת להלאה. היא מוסיפה בתובות MAC (מקור ויעד), פרוטוקול רשות (6 bytes) ומעבירה את הפאקטה לשכבה הפייזית, ככלומר משדרת אותה מהמכשור אל החוץ. בפועל, במקרה שלנו התהילכים שתעשה המחברת הם לוקאים ולבן שכבת ה-IP לא Tosip בתובות MAC או מידע חדש אבל כן יש לה תפקיד בכך שהוא תshedר את הפקטוות אל לולאת-loopback, ככלומר מקומית בתחום המחשב.

אלו הצעדים הרעיוניים, מכאן והילך נסביר במפורט איך נוצרים כוורות ה-TCP\_Header ו-

למעשה עד כאן פירטנו את הרעיון

שבאריזת המנות ומעבשו נפרט מה בפועל

כל שכבה עשויה בהתאם לקוד שכותב

במחברת. ככלומר ננתח את הקוד של

הfonkציות build\_tcp\_header ו-

build\_ip\_header. מדובר בمعنى העמeka

רחבה הרבה יותר... נתחייב לשכבת

האפליקציה וננד לשכבת הרשות.

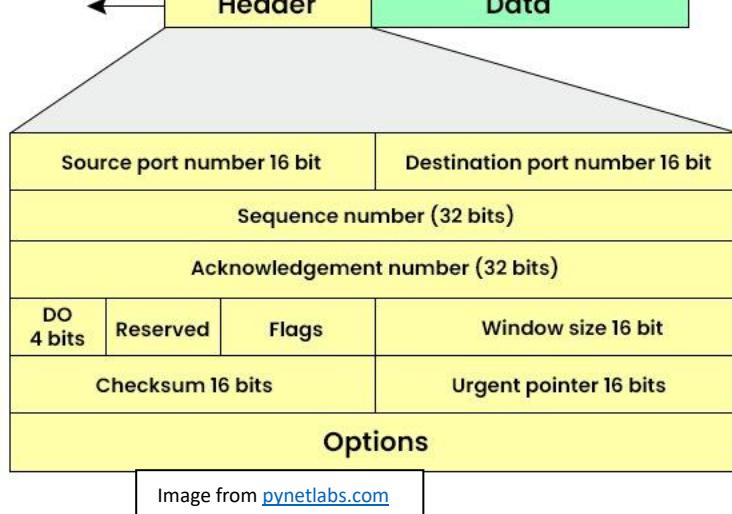
בשכבת האפליקציה תחילת אנחנו טוענים

את ה-CSV המקורי שלנו למחברת עם

שימוש בספריית pandas.

במחברת מוגדרות פונקציות לבניה "ידנית"

של כוורת TCP header של שכבת



התעבורה ולבניית כוורתת header-IP של שכבת הרשת, לשימוש מידת הצורך בהתאם למערכת הפעלה. לאחר מכן המחברת טעונה את ההודעה עבור כל שורה ב-CSV ושולחת אותה לוקאלית עם פונקציית `send`.

כדי לבנות את כוורתת-TCP אנחנו קודם כל בונים כוורתת `tcp_header` ראשונית שמכילה רק חלק מהשדות. הסיבה היא ששדה `checksum` משמש לביקורת המידע נושא על שאר השדות, וכך לחשב אותו נדרש להרכיב את שאר השדות לפני.

אז, בהינתן:

- מספרי הפורט של המקור ושל היעד
- מספר סידורי (number) שסופר כל בית שעובד (ומאותחל רנדומלית אם לא אותחל עידיון)
- מספר סידורי שמאשר את קבלת הנתונים עד הביטח הנוכחי (ack number)
- סוג הפקטה (flags), כולל רצף ביטים התואמים לדגלי Syn, Ack, Fin ודגלים נוספים המתארים את הפעולות שהפקטה אחראית להשלים.
- גודל החלון (Window) שימושו לדעת כמה ביטים ניתן לשלוח ברצף פאקטות אחד ברגע זה למקבל.
- `Data Offset,Reserved`, שדות שימושיים להגדרת אורך הפקטה ולשמירת 4 ביטים נוספים.
- `urgentPointer`, שדה שמשמש לציין ביטים מסוימים שדחופים לעיבוד והוא מצביע על סוף רצף הביטים האלו. לנו אין נתונים שדחופים לעיבוד ולכן השדה ישאיר אפס.

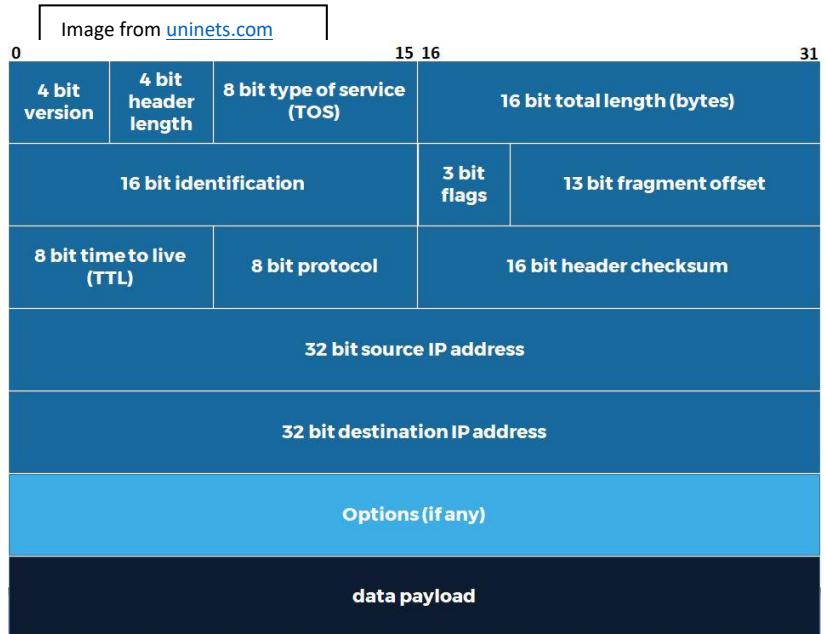
לאחר שאספנו את כל הנתונים אנחנו אורזים אותם לתוך כוורתת `tcp_header` ראשונית.

לאחר מכן אנחנו בונים `pseudo_header` עם חלק מהשדות שנכנסים לכוורתת-IP. (כמו בתובות ה-`checksum`). רק לאחר מכן מחשבים את שדה `checksum`.

- `Checksum`, שדה זה משמש את מקבל כדי לאמת את המידע שבפקטה. הוא עושה זאת על ידי חיבור כל השדות בפקטהividually עם שדות של `pseudo_header`. החיבור הוא של כל 16 ביטים. לאחר קבלת וצאת החיבור הופכים את הביטים זהה `checksum`. אם מקבל מבצע את אותו תהליך ומתקבל תוצאה שונה ממה ששמור בשדה זה הוא יגלה שהמידע שבפקטה השתבש במהלך העברה, וידע לזרוק את הפקטה או לבקש שליחת מחודשת.

רק לאחר שיחסבנו את שדה `checksum` אנחנו אורזים מחדש את `tcp_header` שלנו, הפעם עם שדה `checksum` ובכך השלמנו את אריזת `segment` של שכבת התעבורה.

כדי להסביר את שכבת הרשת ובניית כוורתת-IP, נשים לב למבנה של ה-Header:



בנויות בותרת ה-IP של שכבות הרשות. גם כאן נבנה header התחלתי כדי לבצע את חישוב הchecksum לאחר יותר.

בහינתן:

- כתובות ה-IP של המקור ושל היעד
- גודל הדאטה שהפאקטה נושא (בלומר – ה-payload שקיבלנו בשכבות האפליקציה וגודל הסגמנט שנוסף לה בשכבות התגובה).
- פרוטוקול התגובה (TCP)

אנחנו בונים את ה-header כז':

- אצלנו הגרסה היא 4 (4qv4) ולכן `version=4`.
- header\_length הוא גודל ה-Header\_IP שאנחנו בונים עבשו. הגודל נמדד במילימ' ולא בבייטים. לבן אם הגודל הוא למשל 5 (מילימ'), הגודל בפועל הוא  $20 = 5 \cdot 4$  בייטים.
- TOS הוא שדה שנועד לציין לצד המქבל כיצד לטפל בפאקטה עם רצפי ביטים מיוחדים כלפי (בדרכם כל ציון דרישות דילוי, תפוקה, אמינות מידע וכו'). לנו אין דרישות מיוחדות עבור הפאקטות ולכן השדה יישאר אפס.
- total\_length הוא הגודל של הפאקטה כולה, כלומר הגודל header\_length בבייטים פלוס הגודל של ההודעה והסגמנט שקיבלנו משכבות האפליקציה והתגובה.
- identification השדה זהה משמש את הדאטה להתפצל למקטעים. בלאו, במידה והפאקטה גדולה מדי עבור הקישור שצריך לשדר אותה החוצה, שכבות הרשות תפצל את הדאטה למקטעים נפרדים, עם IP\_headers שונים אך עם שדות identification זהים.
- המקטעים ישלוו בנפרד לקישור, וביעד (שכבות הרשות של הצד המქבל) יתאפסו חזורה אל הפאקטה המקורית שהם מרכיבים ביחיד. הם ידעו להתאסף חזורה אל הפאקטה המקורית לפי השדה identification המשותף שהם מחזקים.
- השדות header\_flags, fragment\_offset הם ביטים שמצוינים האם יש לפצל את הדאטה למקטעים ואם כן, מה המיקום של כל מקטע ביחס לפאקטה המקורית.

- השדה ttl (Time To Live) מגדיר את מספר רכיבי הרשת המxisימי שהפאקטה יכולה לעבור דרכם לפני שתיזרק. בך השדה מגדיר את תוחלת החיים של הפאקטה.

לאחר שהגדכנו את כל השדות הללו אורזים header\_IP ראשוני. לאחר מכן אנחנו מחשבים את שדה checksum על ידי חיבור כל השדות של header\_IP (ב-16 ביטים) והפיכת הביטים של הסכום.

לאחר מכן אנחנו אורזים מחדש את header\_IP ביחד עם שדה checksum ובכך השלמנו את האזיה של שכבת הרשת.

בכך השלמנו את האזיה של שלושת השכבות, אפליקציה (הודעות CSV שהמחברת טוענת ושולחת), התעבורה (הסמנט שנבנה, tcp\_header) והרשת (ה-ip datagram שנבנה, ip\_header). בך כל תעבורה שנמצאה מהמחברת תקרה לאחר Encapsulation של שלושת השכבות האל. התחלנו מرمת האפליקציה וארכזו את ההודעות בשכבות עד ליצירת פאקטות שנושאות את ההודעות.

## 4. ניתוח התעבורה באמצעות WIRESHARK

בשלב הראשון ננתח תעבורה ראשונית שהתקבלה מהפעלת הפונקציה:

```
[30]: def demo_send(num_packets: int=3, delay_sec: float=1.0, flags: int=0x02):
    for i in range(num_packets):
        payload = f'Hello Packet {i}'.encode()
        transport.send(payload, flags=flags)
        time.sleep(delay_sec)
```

עבור כל אחת מהפקודות:

```
demo_send(num_packets=3, delay_sec=1.0, flags=0x02)
demo_send(num_packets=3, flags=0x18)
demo_send(num_packets=3, flags=0x10)
demo_send(num_packets=3, flags=0x01)
demo_send(num_packets=3, flags=0x04)
```

aczin שמספר הפורט 25215 הוא פורט שהוקצה למשתמש (אני, שה:right את Wireshark) על ידי מערכת הפעלה ומספר הפורט 12345 הוא מספר הפורט של האפליקציה (מחברת ה-Jupyter-h).

- `demo_send(num_packets = 3, delay_sec=1.0, flags=0x02)`

זו הגדרה לפאקטת TCP שבה רק הדגל SYN פועל. במיילים אחרים זו פאקטה שמהוות את השלב הראשון בתהליך 3-way-handshake. אבל לפאקטה אין בתובות MADEINA, במיילים אחרים אין כאן באמת חיבור TCP בין שני צדדים, אין תקשורת sockets אלא העברת סתמיית של פאקטות בין בתובות פורט בלבד שהוקצתה על ידי מערכת הפעלה לבין המחברת. וכן נצפה שלאחר הפאקטה הריאונה, תישלח פאקטה שתבהיר על Reset, כי אין אפשרות להתחילה חיבור. ובאמת ניתן לראות שלאחר כל פאקטה כזו שנשלחת למחברת (ומזהות על ידי 0 Hello Packet 1 או Hello Packet 2 או Hello Packet 2 בשדה ה-payload (TCP payload) מקבלים מהמחברת בתגובה פאקטת TCP שבה

.Ack=1, Reset=1

No.	Source Port	Destination Port	Time	Protocol	Syn	Acknowledgment	Push	Reset	Fin	Flags	TCP payload
349	25215	12345	11.084745	TCP	1	0	0	0	0	0x0002	48656c6c6f205061636b65742030
350	12345	25215	11.084795	TCP	0	1	0	1	0	0x0014	
355	25215	12345	12.088255	TCP	1	0	0	0	0	0x0002	48656c6c6f205061636b65742031
356	12345	25215	12.088322	TCP	0	1	0	1	0	0x0014	
374	25215	12345	13.092120	TCP	1	0	0	0	0	0x0002	48656c6c6f205061636b65742032
375	12345	25215	13.092247	TCP	0	1	0	1	0	0x0014	

0000	02 00 00 00 45 00 00 36	00 01 00 00 40 06 7c bf	... E 6 ... @
0010	7f 00 00 01 7f 00 00 01	62 7f 30 39 00 00 00 00	..... b.09....
0020	00 00 00 00 50 02 20 00	a1 b6 00 00 48 65 6c 6c	... P. .... Hell
0030	6f 20 50 61 63 6b 65 74	20 30	o Packet 0

- `demo_send(num_packets=3, flags=0x18)`

זו הגדרה לפאקטת TCP שבה רק הדגלים ack,push,reset מפעילים. כמו מקודם אין באמת תקשורת בין הצדדים וכן נצפה לקבל חזרה Reset=1. הפעם נשים לב שנקבל 0. ack=0. הסיבה היא שהמחברת לא קיבלה לפני כן פאקטת SYN מהמשתמש ולכן היא לא מודעת או לא חושדת בקיום קשר.

20313	25215	12345	2116.209752	TCP	0	1	1	0	0	0x0018	48656c6c6f205061636b65742030
20314	12345	25215	2116.209836	TCP	0	0	0	1	0	0x0004	
20325	25215	12345	2117.214729	TCP	0	1	1	0	0	0x0018	48656c6c6f205061636b65742031
20326	12345	25215	2117.214837	TCP	0	0	0	1	0	0x0004	
20331	25215	12345	2118.217473	TCP	0	1	1	0	0	0x0018	48656c6c6f205061636b65742032
- 20332	12345	25215	2118.217552	TCP	0	0	0	1	0	0x0004	

שאר הפקודות מתנהגות באופן דומה למעט האחרון:

demo\_send(num\_packets=3, flags=0x04) •

זו פקודה שבה רק הדגל Reset פעועל. בשפaket זה נשלחת אפיו לא נקלט פאקטות בתגובה מהמחברת. הסיבה היא שפaket מהסוג זה לא מצפה לתגובה בחזרה אלא מבקשת מהמחברת לסגור את החיבור הנוכחי. אבל למעשה אין חיבור אמיתי ולכן לא נקלט תגובה מהמחברת.

Sequence Number: 0 (relative sequence number)	0000 02 00 00 00 45 00 00 36 00 01 00 00 40 06 7c bf	...E 6 ...@   .
Sequence Number (raw): 0	0010 7f 00 00 01 7f 00 00 01 62 7f 30 39 00 00 00 00	..... b.09.....
[Next Sequence Number: 14]	0020 00 00 00 00 50 04 20 00 a1 b4 00 00 48 65 6c 6c	....P. ....Hello
Acknowledgment Number: 0	0030 6f 20 50 61 63 6b 65 74 20 30	o Packet 0
Acknowledgment number (raw)		

התמונה הבאה מדגימה כיצד נראה הסגמנט של הפאקטוט בפי שמייצגות WIRESHARK.

Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

Transmission Control Protocol, Src Port: 25215, Dst Port: 12345, Seq: 0, Len: 14

Source Port: 25215  
Destination Port: 12345 **PORTS**

[Stream index: 21]  
[Stream Packet Number: 1]

[Conversation completeness: Incomplete (45)]  
[TCP Segment Len: 14]  
Sequence Number: 0 (relative sequence number)  
Sequence Number (raw): 0  
[Next Sequence Number: 15 (relative sequence number)]  
Acknowledgment Number: 0  
Acknowledgment number (raw): 0  
0101 .... = Header Length: 20 bytes (5)

Flags: 0x002 (SYN)

000. .... .... = Reserved: Not set  
...0 .... .... = Accurate ECN: Not set  
.... 0.... .... = Congestion Window Reduced: Not set  
.... .0.... .... = ECN-Echo: Not set  
.... ..0.... .... = Urgent: Not set  
.... ...0.... .... = Acknowledgment: Not set  
.... ....0.... .... = Push: Not set  
.... .... .0.... .... = Reset: Not set

.... .... .1.... .... = Syn: Set  
.... .... .0.... .... = Fin: Not set  
[TCP Flags: .....S.]

Window: 8192  
[Calculated window size: 8192]

Checksum: 0xa1b6 [unverified]  
[Checksum Status: Unverified]  
Urgent Pointer: 0

[Timestamps]  
[Client Contiguous Streams: 0]  
[Server Contiguous Streams: 1]  
TCP payload (14 bytes)

Data (14 bytes)  
Data: 48656c6c6f205061636b65742030  
[Length: 14]

FLAGS

ניתן לראות גם את ה-TCP Payload (בתמונה – “Hello World 0” – TCP Payload

0000 02 00 00 00 45 00 00 36 00 01 00 00 40 06 7c bf	...E 6 ...@   .
0010 7f 00 00 01 7f 00 00 01 62 7f 30 39 00 00 00 00	..... b.09.....
0020 00 00 00 00 50 02 20 00 a1 b6 00 00 48 65 6c 6c	....P. ....Hello
0030 6f 20 50 61 63 6b 65 74 20 30	o Packet 0

בשלב השני ננתה תובורה ממחברת ה-CSV שהתקבלה בתוצאה מהעברת ה-CSV:

## Send Messages from CSV file

Iterate over the rows and send message by message

```
[43]: #Send messages from CSV file
for index, row in messages_df.iterrows():
    # Extract message details from the DataFrame row
    message = row['message']
    message = f"test message {index}" if not message else message
    # Send the message using the RawTcpTransport class
    # (You may need to adjust flags and other parameters as needed)

    #TODO: uncomment the line below to send the messages
    transport.send(message.encode(), flags=0x18) # Example with PSH+ACK flags

    time.sleep(0.1) # Optional delay between messages
```

למעשה שלוחים פאקטות TCP עם הדגמים ack=1,push=1. ניתחנו מוקדם את התנagoות התובורה של פאקטות אלו. ההבדל הוא שבאן אנחנו מחלכים מקובץ ה-CSV שטרפנו בכל פעם את שדה ה-`message` ושולחים אותו. אבן ניתן לראות שככל הפאקטות שמתקבלות בעלות בעלות בוטרת TCP מהצורה

```
[Stream index: 0]
Transmission Control Protocol, Src Port: 25215, Dst Port: 12345, Seq: 1, Ack: 1, Len: 32
  Source Port: 25215
  Destination Port: 12345
  [Stream index: 0]
  [Stream Packet Number: 17]
  ▶ [Conversation completeness: Incomplete (40)]
  [TCP Segment Len: 32]
  Sequence Number: 1      (relative sequence number)
  Sequence Number (raw): 0
  [Next Sequence Number: 33      (relative sequence number)]
  Acknowledgment Number: 1      (relative ack number)
  Acknowledgment number (raw): 0
  0101 .... = Header Length: 20 bytes (5)
  Flags: 0x018 (PSH, ACK)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Accurate ECN: Not set
    ....0.... .... = Congestion Window Reduced: Not set
    ....0... .... = ECN-Echo: Not set
    ....0. .... = Urgent: Not set
    ....1.... = Acknowledgment: Set
    ....1... .... = Push: Set
    ....0... .... = Reset: Not set
    ....0.... = Syn: Not set
    ....0....0 = Fin: Not set
    [TCP Flags: .....AP....]
  Window: 8192
```

קובץ הלכידה לאחר שליחת הودעות CSV בטור המחברת נראה כך:

No.	Source Port	Destination Port	Time	Protocol	Syn	Acknowledgment	Push	Reset	Fin	Flags	TCP payload
1	25215	12345	0.000000	TCP	0	1	1	0	0	0x0018	697076362e6d736674636f6e6e656374746573742e636f6d2c302c307862626430
2	12345	25215	0.000051	TCP	0	0	0	1	0	0x0004	
3	25215	12345	0.104457	TCP	0	1	1	0	0	0x0018	697076362e6d736674636f6e6e656374746573742e636f6d2c302c307865346462
4	12345	25215	0.104562	TCP	0	0	0	1	0	0x0004	
5	25215	12345	0.209886	TCP	0	1	1	0	0	0x0018	697076362e6d736674636f6e6e656374746573742e636f6d2c302c307862626430
6	12345	25215	0.209951	TCP	0	0	0	1	0	0x0004	
7	25215	12345	0.315308	TCP	0	1	1	0	0	0x0018	697076362e6d736674636f6e6e656374746573742e636f6d2c302c307865346462
8	12345	25215	0.315410	TCP	0	0	0	1	0	0x0004	
9	25215	12345	0.419121	TCP	0	1	1	0	0	0x0018	697076362e6d736674636f6e6e656374746573742e636f6d2c302c307862626430
10	12345	25215	0.419220	TCP	0	0	0	1	0	0x0004	
11	25215	12345	0.524244	TCP	0	1	1	0	0	0x0018	697076362e6d736674636f6e6e656374746573742e636f6d2c302c307865346462
12	12345	25215	0.524342	TCP	0	0	0	1	0	0x0004	
13	25215	12345	0.631054	TCP	0	1	1	0	0	0x0018	697076362e6d736674636f6e6e656374746573742e636f6d2c312c307865346462
14	12345	25215	0.631209	TCP	0	0	0	1	0	0x0004	
15	25215	12345	0.734618	TCP	0	1	1	0	0	0x0018	697076362e6d736674636f6e6e656374746573742e636f6d2c312c307862626430
16	12345	25215	0.734695	TCP	0	0	0	1	0	0x0004	
17	25215	12345	0.840747	TCP	0	1	1	0	0	0x0018	7777772e6d736674636f6e6e656374746573742e636f6d2c302c307835633234
18	12345	25215	0.840873	TCP	0	0	0	1	0	0x0004	
19	25215	12345	0.945010	TCP	0	1	1	0	0	0x0018	7777772e6d736674636f6e6e656374746573742e636f6d2c302c307866636531
20	12345	25215	0.945091	TCP	0	0	0	1	0	0x0004	
21	25215	12345	1.048312	TCP	0	1	1	0	0	0x0018	7777772e6d736674636f6e6e656374746573742e636f6d2c302c307835633234
22	12345	25215	1.048476	TCP	0	0	0	1	0	0x0004	
23	25215	12345	1.154639	TCP	0	1	1	0	0	0x0018	7777772e6d736674636f6e6e656374746573742e636f6d2c302c307866636531
24	12345	25215	1.154743	TCP	0	0	0	1	0	0x0004	
25	25215	12345	1.260281	TCP	0	1	1	0	0	0x0018	7777772e6d736674636f6e6e656374746573742e636f6d2c302c307866636531
26	12345	25215	1.260364	TCP	0	0	0	1	0	0x0004	
27	25215	12345	1.365673	TCP	0	1	1	0	0	0x0018	7777772e6d736674636f6e6e656374746573742e636f6d2c302c307835633234
28	12345	25215	1.365783	TCP	0	0	0	1	0	0x0004	
29	25215	12345	1.470111	TCP	0	1	1	0	0	0x0018	7777772e6d736674636f6e6e656374746573742e636f6d2c312c307866636531
30	12345	25215	1.470203	TCP	0	0	0	1	0	0x0004	
31	25215	12345	1.575103	TCP	0	1	1	0	0	0x0018	7777772e6d736674636f6e6e656374746573742e636f6d2c312c307835633234
32	12345	25215	1.575235	TCP	0	0	0	1	0	0x0004	
33	25215	12345	1.680408	TCP	0	1	1	0	0	0x0018	646973636f72642e636f6d2c302c307838306634
34	12345	25215	1.680546	TCP	0	0	0	1	0	0x0004	

ונימן לראות שכל פאקטה בעלת TCP Payload שאינו ריק נושא את שדה message של השורה המתאימה בקובץ CSV. למשל, הפאקטה הראשונה נושא את ההודעה:

0000	02	00	00	00	00	45	00	00	49	00	01	00	00	40	06	7c	ac
0010	7f	00	00	01	7f	00	00	01	62	7f	30	39	00	00	00	00	
0020	00	00	00	00	50	18	20	00	de	f8	00	00	69	70	76	36	
0030	2e	6d	73	66	74	63	6f	6e	6e	65	63	74	74	65	73	74	
0040	2e	63	6f	6d	2c	30	2c	30	78	62	62	64	30				

E I @ |  
b 09 ..  
P ..... ipv6  
.msftcon necttest  
.com,0,0 xbbd0

ושורת CSV הראשונה נושא את שדה message:

L	K	J	I	H	G	F	E	D	C	B	A
Connection Protocol	Response	Type	Length	message	timestamp	Destination Address	Source Address	dst_port	src_port	app_protocol	msg_id
	Message is a query	A	104	ipv6.msftconnecttest.com,0,0xbbd0				53	64155	DNS	1 2

הפאקטה ושורת CSV שתיהן נושאות את אותה ההודעה:

.ipv6.msftconnecttest.com,0,0xbbd0

ולכן שורה מס' 1 בקובץ CSV מתאימה לפאקטה מס' 1 בקובץ הלכידה.

דוגמה נוספת, בקובץ הלכידה (שמצויר לדו"ח), פאקטה מס' 63 נשאת את ה-Payload

0000	02 00 00 00 45 00 00 3b 00 01 00 00 40 06 7c ba	
0010	7f 00 00 01 7f 00 00 01 62 7f 30 39 00 00 00 00	
0020	00 00 00 00 50 18 20 00 7d 2c 00 00 64 6e 73 2e	
0030	67 6f 6f 67 6c 65 2c 31 2c 30 78 36 30 61 66	... ; .. b..09.... ....P.. . },..dns. google,1 ,0x60af

ושורה ה-CSV מס' 32 נשאת את שדה ה-message

Protocol	Type	Port	Source IP	Source Port	Destination IP	Destination Port	Size	Protocol
UDP	Message is a response	AAAA	130		dns.google,1,0x60af		0.026362	192.168.1.18 1.1.1.1 59365 53 DNS 32

הפאקטה ושורת ה-CSV שתיהן נשאות את ההודעה:

dns.google,1,0x60af

ולכן שורה מס' 32 בקובץ ה-CSV מתאימה לפאקטה מס' 63 בקובץ הלכידה.

באופן דומה ניתן לזהות כל הודעה בקובץ CSV גם בקובץ הלכידה.

## חלק שני

# פיתוח יישום רשת ללקוח/שרת ב프וטוקול TCP

### 5. מבוא

נדרשו לבנות פרויקט המימוש צ'אט רב משתמשים שבו קיימים לקוחות שמתקשרים אחד עם השני ושרת שמנהל את החלפת המסרים ביניהם. ככלומר, בהינתן השרת, לקוח א' יציין את שם הלקוח ב' אליו ירצה לפנות, והשרת ידאג להעביר את המסרים של לקוח א' אל לקוח ב'. בנוסף השרת יעביר ללקוח הודעות שקיבל מלקוחות אחרים שכן התקשרות בין הלקוחות אמורה להיות רב-כיוונית; ללקוח אמור להיות היכולת גם לשלוח הודעות לכל לקוח אחר וגם לקבל הודעות מכל לקוח אחר. לדע"ח זה מצורפים שתי אפליקציות שמרכיבות את היישום, האחת היא עבור הלקוח ששלוח בקשות לשרת ומקבל מסרים מהשרת. השנייה היא עבור השרת שמנהל את רוב העבודה בתהילcis. הוא זה שמקבל קלטים ממספר לקוחות ויודיע להתנהל לפיהם, יודיע להעביר הודעות מלקוח פונה א' אל לקוח נמען ב' ומתקשר הוא עצמו עם לקוחות.

את היישום בחרתי לבנות בשפת C++ ובסביבת Visual Studio 2022.

כאמור בנוסך לדע"ח מצורפים האפליקציות בתיקיות:

- Server\_interface
- Client\_interface
- Client\_2

באשר server\_interface היא אפליקציית השרת, client\_interface היא אפליקציית הלקוח, client\_2 הוא עותק של client\_interface שמשמש להרצה בלקוח נוספת נוסף במקביל ל-client\_interface, client\_2 זיהות לחלווטין. פרט לכך התיקיות client\_interface, client\_2\_interface ניתנת להרץ את הפרויקט גם עם יותר לקוחות מהשניים שצורפו, לשם כך ניתן פשוט לשכפל את אחת מהתיקיות האלו ולהרץ את כל העותקים במקביל.  
כל קבצי הקוד בשתי האפליקציות server\_interface, client\_interface מלאוים בתיעודים ובהסבירם לנוחיות הקורא/ת.

## 6. ארכיטקטורת הפרויקט

כדי ליזור תקשורת בין יישומים בשפת C++ אנחנו משתמשים בספריית WinSock2 שמאפשרת לנו גישה לכל פעולות ה-Sockets להן נדרש כדי לאפשר תקשורת. כאשר מדובר בתקשורת מבוססת sockets אנחנו נדרשים לביצוע התהליכים טכניים לפני שנוכל לתקשר עם אפליקציות אחרות ולאחר מכן לתקשר עימן. התהליכים משתנים כמעט לגמרי בין הלקוח לשרת, ואלו הם:

עבור הלקוח נבצע:

- אתחול ספריית Winsock
- יצירת socket שישתמש את הלקוח להתחבר לשרת (פונקציית יצירת socket (socket))
- ציון בתובת השרת וחיבור בין socket הלקוח לבין השרת (פונקציית connect (connect))
- ניהול תקשורת עם השרת באמצעות שליחת וקבלת הודעות (recv, send)
- בסוף השימוש, קרי התنتוקות של הלקוח מהשרת או יצא מהצ'אט, נסגור את ה-socket (פונקציית closesocket)
- קרייה לפונקציית cleanup של ספריית Winsock

עבור השרת נבצע:

- אתחול ספריית Winsock
- יצירת socket שיძין ללקוחות נכנסים (פונקציית socket (socket))
- קישורת ה-socket המאזין אל הכתובת שבה ניצב את השרת, ככלומר אל כתובות ה-ip ואל מספר הפורט שנרצה לו (פונקציית bind (Bind))
- רק לאחר שייצרנו את ה-socket וקשרנו אותו לכתובת אחרת, נתחליל בהازנה ללקוחות נכנסים.
- בעת האזנה, נקבל ללקוחות חדשים ונשמור אותם בסיס נתונים שיפורט בהמשך (פונקציית listen (listen))
- לאחר קבלת הלקוח ננהל תקשורת באמצעות קבלה ושליחת הודעות בין הלקוח לשרת (פונקציות recv, send) (recv, send). חשוב לציין שככל לקוח פועל בצורה א-סינכרונית. ככלומר הפעולות של לקוח אחד אינן תלויות בזמן הפעולה של לקוח שני. התהליכים של כל לקוח קורים במקביל.
- באופן עקרוני השרת אמר לעובוד תמיד (ריצה אינסופית) כי בלבדו הצ'אט לא קיים, בשונה מהלקוח שבן אמר להפסיק מהצ'אט בשלב מסוים (לא רץ אינסופית). עם זאת, אנחנו מתעסקים עם sockets ולבן למן הסדר הטוב נציין שיש לסגור את ה-socket בסוף השימוש. (פונקציית closesocket (closesocket)).
- בהמשך לסעיף הקודם, קרייה לפונקציית cleanup של ספריית Winsock

באמור הפרויקט מורכב משתי אפליקציות, ללקוח ושרת וכן נקייש פירוט לכל אחת.

## 6.1 – אפליקציית השירות Server\_interface

תיקיה זו מורכבת מ-4 קבצי header שם מרוכדים הפונקציות עם תיעודים כלליים ו-4 קבצי cpp גם הם עם תיעודים מפורטים וביהם בפועל מיושמים התהיליכים. פירוט רחוב נמצא בתיעוד שבפרויקט, וכך גם כאן נציג את קבצי header.

header.h – בקובץ זה מוגדר struct עם פרטימ על הלוקוח, שבו אנחנו שומרים את השם של הלוקוח (השם ייחודי לכל לוקוח – דרישות הפרויקט מאפשרות לנו להניח זאת), ואת ה-socket של הלוקוח. נגידר גם פונקציה למציאת לוקוח בודד בתוך רשימה של לוקוחות בהתבסס על השם שלו. כפי שניתן להבין, בסיס הנתונים שבו נשמר את הלוקוחות יהיה רשיימה.

```
1  /* Made by Liran Dagan 215609397 */
2  #pragma once
3  #include <iostream>
4  #include <WinSock2.h>
5  #include <string>
6  #include <list>
7
8  using namespace std;
9
10 // Each client has a name and a socket to which the server sends info to and receives info from
11 struct ClientInfo {
12     string name;
13     SOCKET socket;
14
15     bool operator==(const ClientInfo&);
16 };
17
18 ClientInfo* findByName(const string&, list<ClientInfo>&);
19
```

ההמשר לפרוצדורות שצינו בעמודים הקודמים, קובץ זה מכיל פועלות מקידימות שאנו צריכים לקיים כדי לאפשר תקשורת מבוססת sockets. בקובץ מופיעימוש בסיסי של אתחול ספריית Winsock, יצורת socket המאזין של השרת וקשרתו לכתובת השרת. ההאזנה וקבלת הלקוחות מתבצעות בנפרד בפונקציית ה-main שבקובץ server.h שיצג מיד.

```
1  /* Made by Liran Dagan 215609397 */
2  #pragma once
3  #include <WinSock2.h>
4  #include <WS2tcpip.h>
5  #include <string>
6  #pragma comment(lib, "ws2_32.lib")
7
8  using namespace std;
9
10 /*
11  These are the steps the server handles to perform socket interactions:
12  1. Initialize winsock lib
13  2. Create the server socket
14  3. Bind ip (Which is 0.0.0.0) and port (which is 12345) to the socket
15  -----
16  4. Listen on the socket
17  5. Accept clients
18  -----
19  6. 'recv' for receiving messages from clients and 'send' for forwarding them to other clients
20  -----
21  7. Close the server socket when finished
22  8. Cleanup
23  |
24  | The goal: Use 'recv' to get inputs from the users, Use 'send' to instruct users and
25  | forward their messages onwards
26  */
27
28 // step 1: Initialize WinSock version 2.2
29 bool initialize();
30
31 // step 2: Create a listening TCP socket for the server to accept clients
32 bool createTCPsocket(SOCKET& listener);
33
34 // step 3.1: Create the server's Address details, ip and port
35 bool createAddress(sockaddr_in& serverAddr, const string& ip, const int portNumber);
36
37 // step 3.2: Bind our TCP listening socket to the address details we created
38 bool assignAddress(SOCKET& listenSocket, sockaddr_in& serverAddr);
39
```

– בקובץ זה מרכזות הפקציית שמהוות את הלוגיקה שבתקשרות עם כל לקוחות באשר אנחנו משתמשים בפקציית על interact בפקציית נוספת כדי לנהל את התהליכי מול הלקוחות.

```

1  /* Made by Liran Dagan 215609397 */
2  #pragma once
3  #include <iostream>
4  #include <string>
5  #include <list>
6  #include "client_info.h"
7  using namespace std;
8
9  /* Given 2 strings, compare them case insensitively(for example "liRaN" == "LIRan") */
10 bool equalStrings(const string& str1, const string& str2);
11
12 /* Given a client's name and socket we create and save his info in a 'ClientInfo' struct and add him to
13 | a list containing all active chatters */
14 ClientInfo createClient(string& name, SOCKET& socket, list<ClientInfo>& all_clients);
15
16 /* Given any client's message to the server we first have to check if he entered 'quit'
17 | If he did, then we notify the client on his disconnection and return true
18 | If he didn't, we return false */
19 bool isQuit(string& message, ClientInfo& client);
20
21 /* Given a client's input of a target he wants to message, the server checks if the target
22 | is valid and sends feedback to the client. Errors to be detected are:
23 | 1. target is the client himself (logical error, client is trying to message himself).
24 | 2. target doesn't exist (technical error, the target can not be found)
25 | If there is no error we notify the client that we found the target, else we
26 | send the error to the client. */
27 bool isError(string& targetName, ClientInfo& client, list<ClientInfo>& all_clients);
28
29 /* If the client disconnected (due to 'quit' message or due to connection error)
30 | Then we discard his socket and remove him from the list of active chatters */
31 void deleteClient(ClientInfo& client, list<ClientInfo>& all_clients);
32
33
34 /* 'Interact' is the main function that manages the interactions between the server and the client.
35 | It Handles the server's actions with a single client and forwards messages from the client
36 | to a target client.
37 | If needed, the server sends feedback messages regarding errors.
38 | Interact utilizes all the functions from above.
39
40 In short, we do the following:
41 0. We got a message from the client
42 1. Check for 'quit' (isQuit), if the client didn't send quit we proceed to 2., else we stop running
43 and remove him (deleteClient).
44 2. Use a 'mode' variable that gets 0, 1 or 2 to distinguish between actions as followed:
45 mode = 0 -> We receive client's name and save his details (createClient)
46 mode = 1 -> we receive the target client and check for errors (isError)
47 mode = 2 -> We got a message to deliver and we send it to the target client by finding his socket in the clients list
48 3. return to 0.
49 */
50
51 void Interact(SOCKET clientSocket, list<ClientInfo>& all_clients);

```

– בקובץ זה פונקציית ה-main, בה אנחנו מבצעים את כל פעולות ה-socket המקידימות כדי לאפשר לשרת לתקשר עם הלוקחות. לעומת זאת, אנחנו מבצעים את כל הפעולות הקיימות בקובץ socket\_setups.h ומוסיפים האזנה וקבלת ליקוחות.

כדי לנו להל את כל הלוקחות במקביל אנחנו צריכים לשמר מידע על כולן. כמו כן אנחנו צריכים לבנות מנגנון שיאפשר להוסיף, למחוק ולהפוך ליקוחות בתוך המאגר. נוכל לבצע זאת עם שולון נדדק לבנייה נתונים שיאפשר להוסיף, למחוק ולהפוך ליקוחות בתוך המאגר. כאמור שסדר את הלוקחות לפי סדר האלף-בית המוונ סוגים של מבנים. אפשר למשה לשמר עץ AVL מאוזן שמסדר את הלוקחות לפי סדר האלף-בית של שמותיהם, או להשתמש בדרכים מתוחכבות יותר לניהול המאגר. עם זאת, זו לא גולת הכוחות של הפרויקט הזה, שכן נפשט מעט את השימוש ואת ההסבירים. כאמור נשמר את הלוקחות בתוך רשיימה מוקורת (list) מסוג ClientInfo אותה נאתחל בהתחלה בראיקה. כל לקוח שמתחבר לשרת יתווסף לרשיימה, כל לקוח שייתנק מהשירות ימחק מהרשימה ואם נרצה להעביר הודעה אל לקוח נמען יוכל פשוט לחפש אותו ברשיימה ולקבל פוינטර אל הפרטים שלו.

חשוב לציין – עברו כל לקוח שנבוצע עליו accept thread שטרתו לפצל את הקש של הרשת לנמה ליקוחות במקביל, פונקציית ה-thread היא Interact מהקובץ interact\_with\_client.h לה אנחנו נותנים את ה-socket של הלוקח הנכנס וறרנס לבנייה הנתונים, הרשיימה, ששומרת את כל הלוקחות. בדרך זו נוצרת א-סינכרוניות ומשתמשים אינם תלויים זה זה!

```

1  /* Made by Liran Dagan 215609397 */
2  #pragma once
3  #include <iostream>
4  #include <tchar.h>
5  #include <thread>
6  #include <list>
7  #include "client_info.h"
8  #include "socket_setups.h"
9  #define MAX_USERS 100
10 using namespace std;
11
12 /* In main we do all socket procedures from socket_setups.h along with listening and
   accepting clients. We check for errors in each procedure. We then create an empty
   list of type ClientInfo in which we will store all the active clients.
   Once a client comes in and we accept him, we send his socket to the 'Interact'
   function from interact_with_client.h by a thread to isolate the server and him, along with a
   reference to the list of clients we created.
   By using threads we split the server's attention to each client in particular creating desynchronization
   which is essential for chatting between one and another */
13 int main();
14
15
16
17
18
19
20

```

מתיק :server.cpp

```

list<ClientInfo> clients; // Stores the names & sockets of all clients participating in chat
while (true) {
    // step 5: Accept and start communicating with clients
    SOCKET clientSocket = accept(listenSocket, NULL, NULL); // A client tries to access the server
    if (clientSocket == INVALID_SOCKET)
    {
        cout << "Client socket is invalid" << endl;
    }
    else // We split the server's attention to each reaching client, so we get a multi client system
    {
        thread T(Interact, clientSocket, ref(clients)); // Each client gets his own treatment simultaneously with threads
        T.detach();
    }
}
closesocket(listenSocket);
WSACleanup();
return 0;

```

## 6.2 – אפליקציית הלוקו Client\_interface

תיקיה זו מורכבת מ-3 קבצי header שם מרוכדים הפונקציות עם תיעודים כלליים ומ-3 קבצי cpp גם הם עם תיעודים מפורטים וביהם בפועל מיושמים התהיליכים. פירוט רחוב נמצא בתיעוד שבפרויקט, וכך נציג את קבצי ה-.header.

socket\_setups.h – בדומה לאפליקציית השרת, גם באפליקציית הלוקו נגידר קובץ עם פעולות מקדימות שאנו צריכים לקיים כדי לאפשר תקשורת מבוססת sockets. בקובץ מופיע מימוש בסיסי של אתחול ספריית Winsock, יצירה socket התחברות לשרת והתחברות לשרת.

```
1  /* Made by Liran Dagan 215609397 */
2  #pragma once
3  #include <WinSock2.h>
4  #include <WS2tcpip.h>
5  #include <string>
6  #pragma comment(lib, "ws2_32.lib")
7  using namespace std;
8
9  /*
10 These are the steps the client handles to perform socket interactions:
11 1. Initialize winsock
12 2. Create comuunicating socket
13 3. Connect to server
14 4. 'send' for delivering messages to server and 'recv' for receiving messages from server
15 5. Close the socket when finished
16 6. Cleanup
17
18 The goal: Use 'send' and 'recv' methods with the server in order to
19     1. forward messages to a specified client
20     2. get messages from other clients
21
22 */
23
24 // step 1: Initialize WinSock version 2.2
25 bool initialize();
26
27 // step 2: Create a TCP socket to contact the server with
28 bool createTCPsocket(SOCKET& serverSocket);
29
30 // step 3.1: Create the address details of the server the client connects - ip and port
31 bool createAddress(sockaddr_in& serverAddr, const string& ip, const int portNumber);
32
33 // step 3.2: Connect to the server using the 'connect' function
34 bool connectToServer(SOCKET& serverSocket, sockaddr_in& serverAddr);
35
```

בקובץ זה מרכזת כל הלוגיקה של אפליקציית הלקוח משום שבקובץ זה מפורטות כל הפעולות בין הלקוח לשרת מצד הלקוח. הפונקציות שבקובץ זה מהוות את הלוגיקה שבתקשרות באשר אנחנו משתמשים בפונקציות על `sendMessage`, `receiveMessage` שנעזרות בפונקציות נוספות כדי לנהל תהליכי מול השרת.

```

1  /* Made by Liran Dagan 215609397 */
2  #pragma once
3  #include <WinSock2.h>
4  #include <iostream>
5  #include <thread>
6  #include <string>
7
8  using namespace std;
9
10 // return str1==str2 case insensitive ('a'=='A' for example)
11 bool equalStrings(const string&, const string&);
12
13 // Send client's name to the server to be known. Return success (true) or fail (false)
14 bool registerClient(SOCKET& server);
15
16
17 /* Send name of another client we want to message. Server will respond with a feedback message.
18 | Return success (true) or fail (false) */
19 bool searchTarget(SOCKET& server);
20
21
22 /* A thread function. Here we handle all of client's messages to the server in correlation to 'mode'
23 'sendMessage' utilizes all the functions above.
24 In short we do the following:
25 1. Use the 'mode' variable that gets 0, 1 or 2 to distinguish between actions as followed:
26    mode = 0 -> Send client's name to server (registerClient)
27    mode = 1 -> Specify target (searchTarget)
28    mode = 2 -> Send message to server to deliver to target
29 2. If connected==true meaning client didn't quit and no connection error occurred then go to 1.
30    else quit function */
31 void sendMessage(SOCKET server);
32
33
34 /* A thread function. Here we handle all of server's messages to the client including error feedbacks
35 and messages from other clients.
36 In short we do the following:
37 0. Get message/prompt from server
38 1. if got sameUser or UserNotFound prompts then stay in mode = 1 for sendMessage
39 2. if got userFound prompt then proceed to mode = 2 for sendMessage
40 3. if got Quit prompt or a disconnection error then connected=false and 6.
41 4. if none of the above then we actually got a message from another client and not a prompt, so we display it
42 5. Go to 0.
43 6. quit both thread functions sendMessage, receiveMessage and return to main to finish program */
44 void receiveMessage(SOCKET server);
45

```

בקובץ זה פונקציית ה-main, בה אנחנו מבצעים את כלפעולות ה-socket המקידימות כדי לאפשר ללקוח לתקשר עם השירות. אנחנו מבצעים את כל הפעולות הקיימות בקובץ.h.socket\_setups.h: חשוב לציין – אנחנו יוצרים שני thread כאשר פונקציות ה-thread הן: interact\_with\_server.h, sendMessage וreceiveMessage. שילוב שני ה-threads מאפשר לנו לשלוח הודעות ולקבל הודעות socket שאיתו מתקשרים עם השירות. בז'מינית.

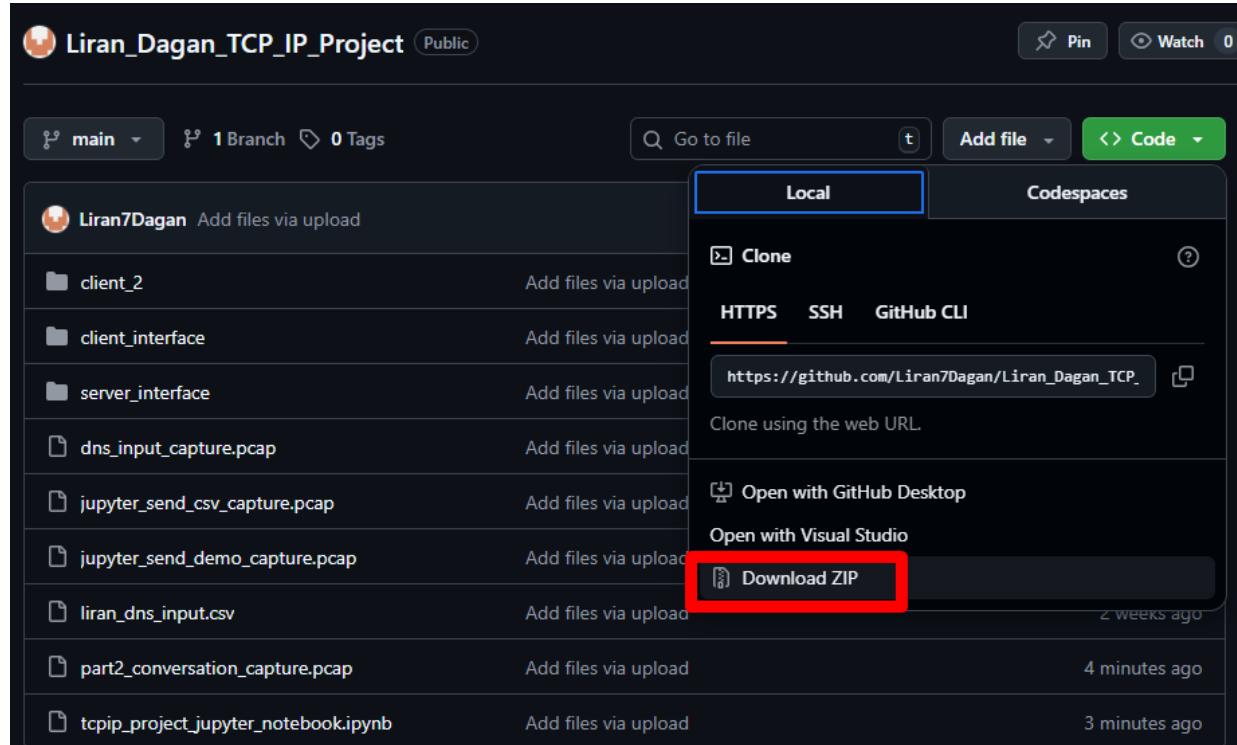
```
1  /* Made by Liran Dagan 215609397 */
2  #include "socket_setups.h"
3  #include "interact_with_server.h"
4  using namespace std;
5
6  /* In main we do all socket procedures from socket_setups.h.
7   * We check for errors in each procedure.
8   * We then create 2 threads, one for each function: sendMessage, receiveMessage
9   * and we give both of them the socket with which we communicate with the server.
10  By joining the threads we communicate with the server while having the two functions enable
11  each other. receiveMessage reacts to the server's feedback to sendMessage, and
12  sendMessage adjusts it's actions by reacting to receiveMessage's reponse to the server's
13  feedback. As a result we get one coherent application that sends and receives messages
14  at the same time. */
15 int main();
16
```

## 7. הוראות התקנה והרצה

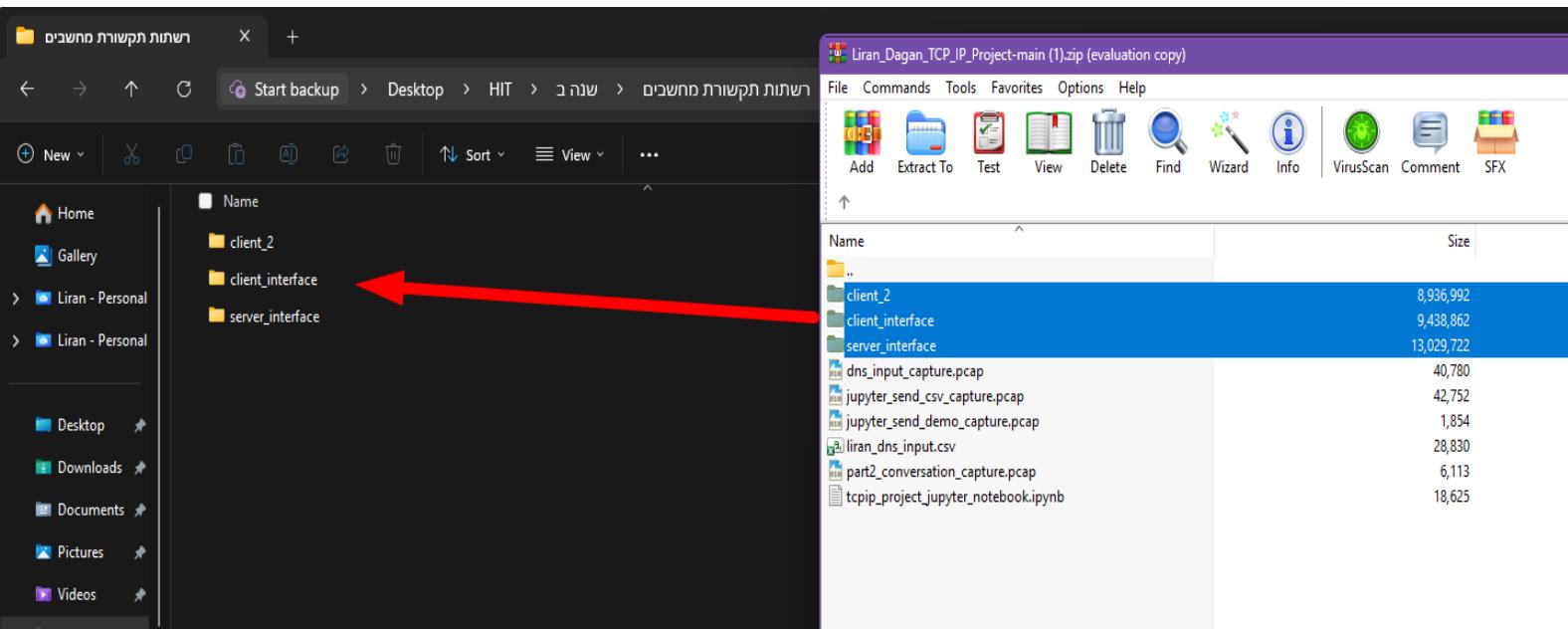
לגייטהאב מצורפים התקיות client\_interface, server\_interface, client\_2\_interface שצויינו קודם לכן. כאמור שלושת האפליקציות נכתבו בשפת C++ ולכן ניתן על המחשב יש לפתח את קבצי ה-solution שלhn בויזואל סטודיו.

צריך לפתח חלון ויזואל סטודיו אחד שירץ את השרת, חלון אחד שירץ את לוקח 1 וחלון אחד שירץ את לוקח 2. במידת הרצון ניתן לשכפל את תקיות הלוקוח ולפתח עוד חלונות שיריצו עוד לוקחות.

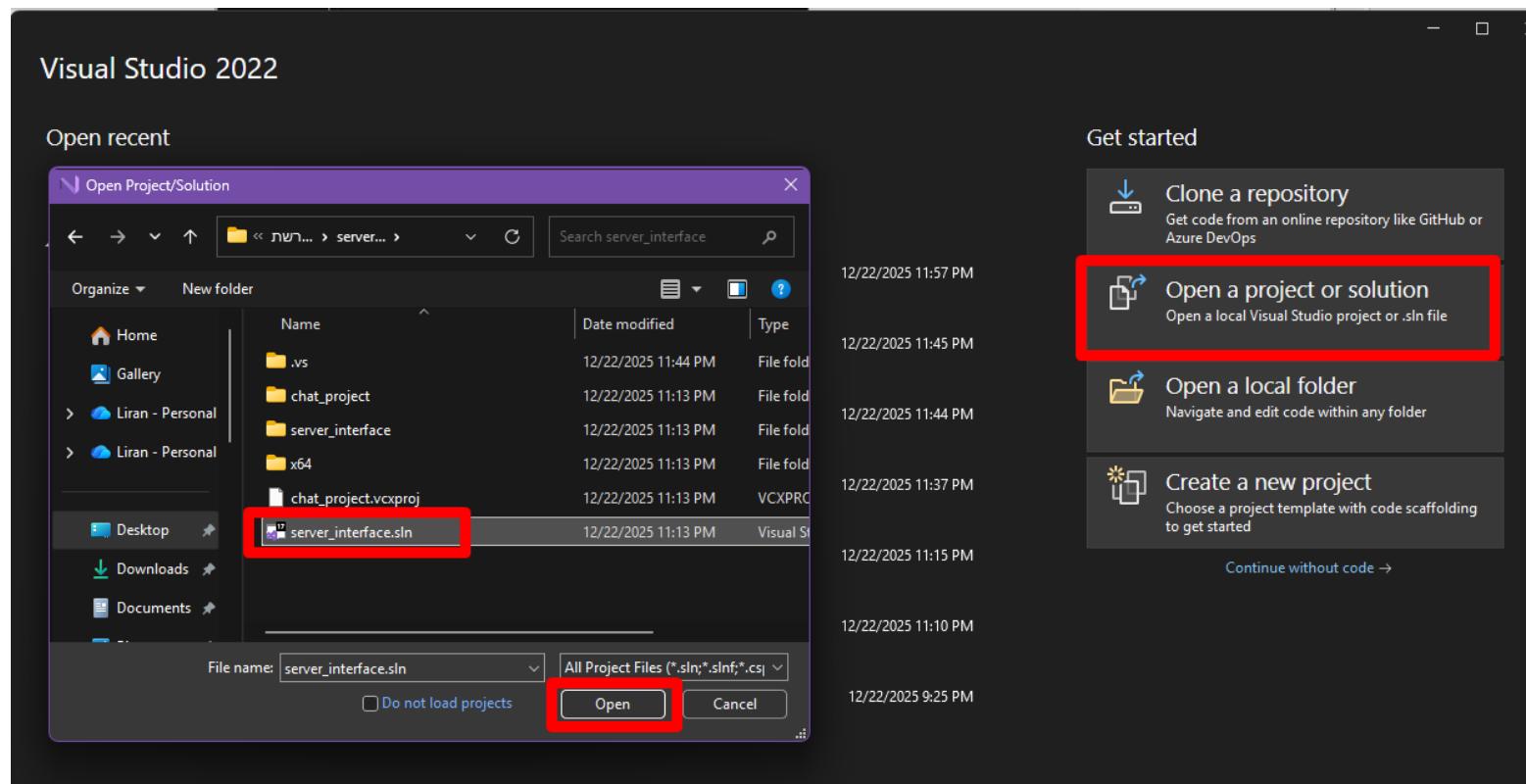
לכן אני מציע להורד בקובץ ZIP את תוכן הגיטהאב ולהלץ / Lager או שתי התקיות אל נתיב מועדף במחשב.



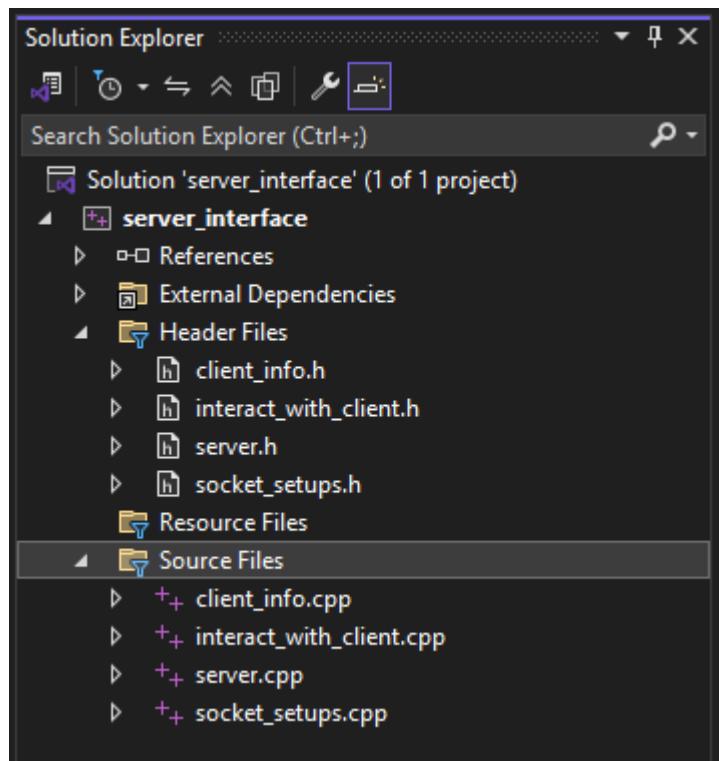
לאחר הורדת ה-ZIP, גירית התקינות:



לאחר מכון, בחלוןית של ייזואל סטודיו יש לפתוח את `server_interface.sln` שנמצא בתיקייה שנגירה.

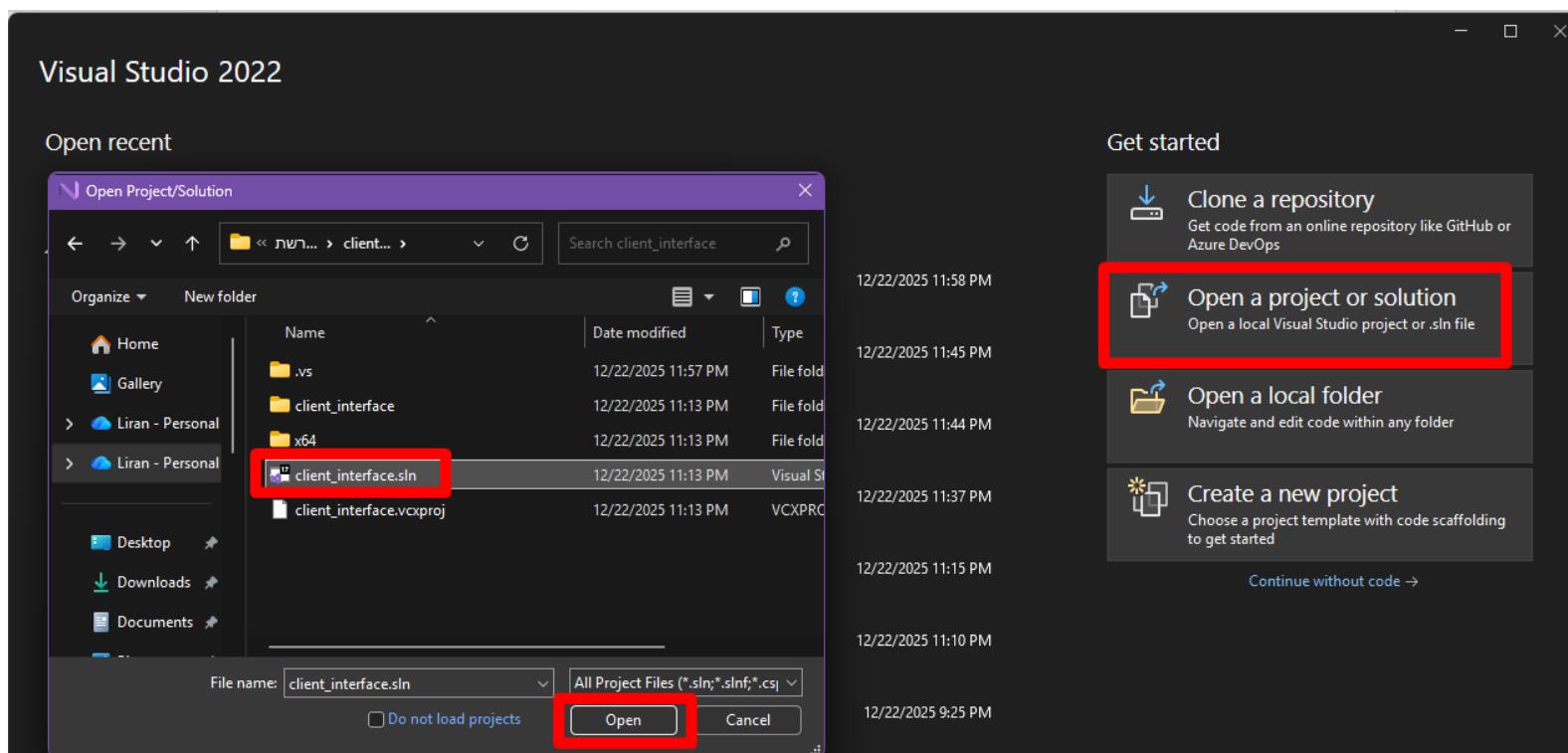


בשלב זה ניתן לפתח ולראות את כל קבצי הקוד - ה-.h header וה-.cpp של השרת בתוך ייזואל סטודיו.



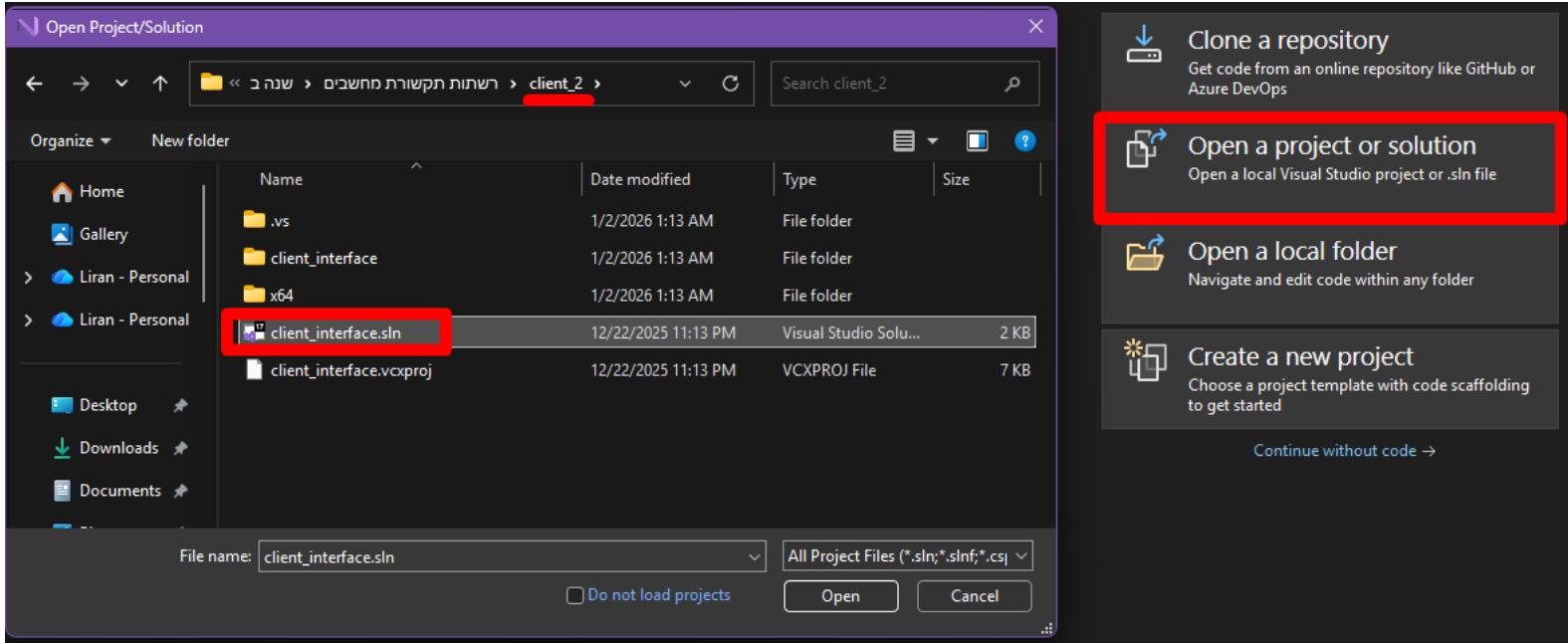
עבשו ציר לפתח את אפליקציית הלוקו, בשבייל ברן ציר לפתח חלון נוסף של ויזואל סטודיו שייעבוד במקביל לחלון שכבר נפתח עבור השרת.

לאחר שנפתח חלון ויזואל סטודיו חדש נפתח את אפליקציית הלוקו באותו אוףן, נפתח את client\_interface.sln



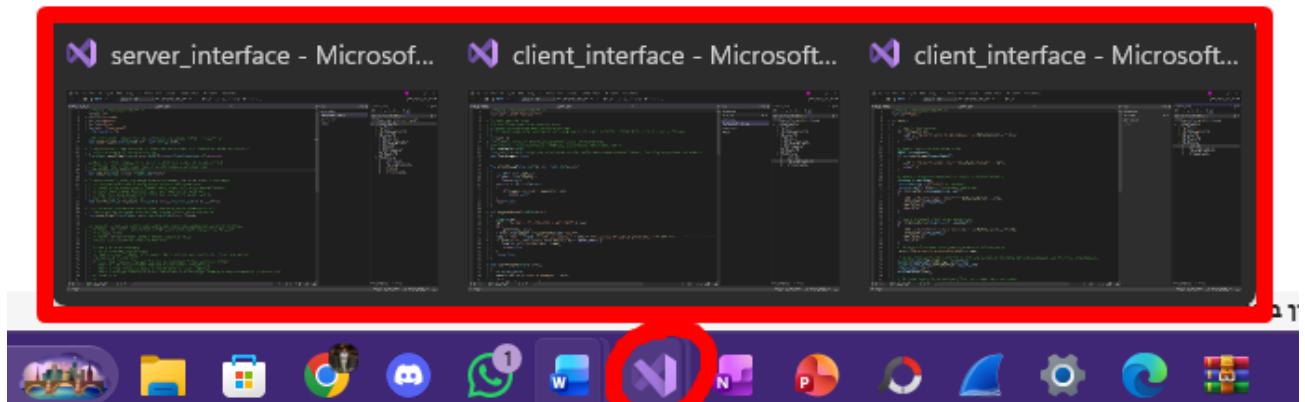
בשלב זה ניתן לפתח ולראות את כל קבצי הקוד – ה-.h ו- cpp של הלוקו בתוך ויזואל סטודיו.

ובאופן זהה נפתח בחלון נפרד את `.sln` שמצא בתיקייה שנגירה:



אם נרצה לפתח עוד לköחות, כאמור נשכפל את תיקיית `client_interface` ונפתח בחלונות ויזואל סטודיו נפרדים את קובץ ה-`.sln`. של כל אחד.

עד לקבלת של לפחות שלושה חלונות ויזואל סטודיו שפועלם וניתנים להרצה במקביל. אחד בלבד עבור השרת ושניים או יותר עבור הלköחות.



בכך מסתכם שלב ההתקנה, נעבור לשלב ההרצה.

יש להריץ את אפליקציית השרת ראשונה, ורק לאחר מכן את אפליקציות הלוקוח.

ממשק הלוקוח מכיל בעצמו הודעות על המסך שמדריכות את המשתמש ולבן אני מציע פשוט להריץ כל אפליקציה בנפרד, חלון הדיבאגר / מסך הלוקוח שייפתח כבר יוכל להודיעו שידרכו את המשתמש כמו "הכנס את שマー", "הזןquit כדי לצאת" וכו'.

חשוב לציין, אפליקציית הלוקוח משתמשת על כך שהשרת עובד בעירפון ולכן יש להריץ קודם את השרת ורק אז את הלוקוח/ות. אם ההריצה תקרה בסדר הפוך (קודם ללקוח ואז שרת), השרת יעבוד ברגע אבל הלוקוח יקבל את הדרישה (שהיא חלק מהקוד): Connection failed to establish: 10061, בলומר הודעה לכך שהשרת לא נמצא ולא ניתן להתחבר אליו. (השרת הורץ רק אחרי הלוקוח ולכן הוא עדין לא קיים בזמן הריצת הלוקוח).

דוגמה לשגיאת הריצת הלוקוח לפני השרת:

```
C:\Users\liran\Desktop\ > Server is now listening on port 12345
Microsoft Visual Studio > Connection failed to establish: 10061
C:\Users\liran\Desktop\client_interface\x64\Debug\client_interface.exe (process 50404) exited with code -1 (0xffffffff).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

בנוסף, האפליקציות מטפלות בשגיאות (כמו זו שבדוגמה) וכך אמורות לҚරот קሪסוט בלתי צפויות.

אם קורות שגיאות כלשהן האפליקציות לא יקרסו אלא ידפיסו על המסך הודעות מתאימות.

עוד יש לציין כי המסר שמקבלים מאפליקציית השרת אינו אינטראקטיבי אלא אינפורטטיבי, הוא עוזר להמחיש התהlications שקיימים בזמןאמת בין לקוחות לשרת ובין לקוחות ללוקוחות וכך לא מקבל בעצמו קלט מהמשתמש אלא בסך הכל מציג מידע שעובר דרכו.

המסר שמקבלים מאפליקציית הלוקוח לעומת זאת הוא מבוסן אינטראקטיבי וинформационטיבי.

## 8. דוגמאות קלט ופלט

דוגמיה לשיחה בין שני משתמשים:

The image shows two adjacent windows of the Microsoft Visual Studio Debug console. Both windows have the title 'Microsoft Visual Studio Debug' and a close button ('X'). The left window contains messages from user 'Liran' and the right window contains messages from user 'Maor'. The messages are as follows:

**Liran's Window (Left):**

- Connection to server established
- Welcome to the chat! What is your name?
- Liran
- Hello Liran! Have fun chatting.
- You can quit any time by typing 'QUIT'.
- 
- Who do you want to message?
- itzik
- User Was Not Found.
- 
- Who do you want to message?
- yaron
- User Was Not Found.
- 
- Who do you want to message?
- Maor
- Connection created
- hey maor
- Maor: whats uppppppppp liran
- I'm good
- Today is Real Madrid vs Barcelona
- Maor: yeah right it will be a great game for sure
- Wanna come over? we can order pizza
- Maor: of course
- be here at 21:30
- Maor: alright I'll see you then
- good, until then!
- Maor: bye!
- quit

**Maor's Window (Right):**

- Connection to server established
- Welcome to the chat! What is your name?
- Maor
- Hello Maor! Have fun chatting.
- You can quit any time by typing 'QUIT'.
- 
- Who do you want to message?
- gal
- User Was Not Found.
- 
- Who do you want to message?
- Liran
- Connection created
- Liran: hey maor
- whats uppppppppp liran
- Liran: I'm good
- Liran: Today is Real Madrid vs Barcelona
- yeah right it will be a great game for sure
- Liran: Wanna come over? we can order pizza
- of course
- Liran: be here at 21:30
- alright I'll see you then
- Liran: good, until then!
- bye!
- quit

At the bottom of both windows, there is a status message: 'C:\Users\liran\Desktop\HIT\??? ?\????? ?????? ??????' followed by instructions to automatically close the console when debugging starts and to press any key to close the window.

עבור השיחה שלמעלה, פלט השירות הוא:

C:\Users\liran\Desktop\HIT\> + | -

```
Server is now listening on port 12345
New client joined: 'Liran'
New client joined: 'Maor'
'Liran' Tried to reach a non existent client: 'itzik'
'Maor' Tried to reach a non existent client: 'gal'
'Liran' Tried to reach a non existent client: 'yaron'
'Liran' Reaches 'Maor'. Server is ready to transmit messages!
'Maor' Reaches 'Liran'. Server is ready to transmit messages!
'Liran' TO 'Maor': hey maor
'Maor' TO 'Liran': whats upppppppppp liran
'Liran' TO 'Maor': I'm good
'Liran' TO 'Maor': Today is Real Madrid vs Barcelona
'Maor' TO 'Liran': yeah right it will be a great game for sure
'Liran' TO 'Maor': Wanna come over? we can order pizza
'Maor' TO 'Liran': of course
'Liran' TO 'Maor': be here at 21:30
'Maor' TO 'Liran': alright I'll see you then
'Liran' TO 'Maor': good, until then!
'Maor' TO 'Liran': bye!
'Liran' Disconnected
'Maor' Disconnected
```

ניתוח השיחה הנ"ל ב-WireShark

No.	Time	Source Port	Destination Port	Protocol	TCP payload	Syn	Acknowledgment	Fin	Push	Sequence Number	Acknowledgment Number	ip version	Time to Live
295	8.333307	63974	12345	TCP		1	0	0	0	0	0	4	128
296	8.333386	12345	63974	TCP		1	1	0	0	0	1	4	128
297	8.333425	63974	12345	TCP		0	1	0	0	1	1	4	128
471	14.018493	63992	12345	TCP		1	0	0	0	0	0	4	128
472	14.018565	12345	63992	TCP		1	1	0	0	0	1	4	128
473	14.018592	63992	12345	TCP		0	1	0	0	1	1	4	128
838	60.531866	63974	12345	TCP	4c6972616e	0	1	0	1	1	1	4	128
839	60.531894	12345	63974	TCP		0	1	0	0	1	6	4	128
840	64.083197	63992	12345	TCP	4d616f72	0	1	0	1	1	1	4	128
841	64.083226	12345	63992	TCP		0	1	0	0	1	5	4	128
983	71.983371	63974	12345	TCP	69747a696b	0	1	0	1	6	1	4	128
984	71.983421	12345	63974	TCP		0	1	0	0	1	11	4	128
985	71.983615	12345	63974	TCP	533a55736572...	0	1	0	1	1	11	4	128
986	71.983633	63974	12345	TCP		0	1	0	0	11	15	4	128
987	75.264535	63992	12345	TCP	67616c	0	1	0	1	5	1	4	128
988	75.264590	12345	63992	TCP		0	1	0	0	1	8	4	128
989	75.264798	12345	63992	TCP	533a55736572...	0	1	0	1	1	8	4	128
990	75.264821	63992	12345	TCP		0	1	0	0	8	15	4	128
1009	81.062028	63974	12345	TCP	7961726f6e	0	1	0	1	11	15	4	128
1010	81.062065	12345	63974	TCP		0	1	0	0	15	16	4	128
1011	81.062286	12345	63974	TCP	533a55736572...	0	1	0	1	15	16	4	128
1012	81.062309	63974	12345	TCP		0	1	0	0	16	29	4	128
1132	106.289869	63992	12345	TCP	4c6972616e	0	1	0	1	8	15	4	128
1133	106.289899	12345	63992	TCP		0	1	0	0	15	13	4	128
1134	106.290399	12345	63992	TCP	533a55736572...	0	1	0	1	15	13	4	128
1135	106.290419	63992	12345	TCP		0	1	0	0	13	26	4	128
1163	111.394157	63974	12345	TCP	4d616f72	0	1	0	1	16	29	4	128
1164	111.394187	12345	63974	TCP		0	1	0	0	29	20	4	128
1165	111.394434	12345	63974	TCP	533a55736572...	0	1	0	1	29	20	4	128
1166	111.394449	63974	12345	TCP		0	1	0	0	20	40	4	128
1210	119.144548	63974	12345	TCP	686579206d61...	0	1	0	1	20	40	4	128
1211	119.144582	12345	63974	TCP		0	1	0	0	40	28	4	128
1212	119.144668	12345	63992	TCP	433a4c697261...	0	1	0	1	26	13	4	128
1213	119.144695	63992	12345	TCP		0	1	0	0	13	43	4	128
1232	128.222671	63992	12345	TCP	776861747320...	0	1	0	1	13	43	4	128
1233	128.222703	12345	63992	TCP		0	1	0	0	43	37	4	128
1234	128.222759	12345	63974	TCP	433a4d616f72...	0	1	0	1	40	28	4	128
1235	128.222783	63974	12345	TCP		0	1	0	0	28	72	4	128
1312	134.185624	63974	12345	TCP	49276d20676f...	0	1	0	1	28	72	4	128
1313	134.185661	12345	63974	TCP		0	1	0	0	72	36	4	128
1314	134.185743	12345	63992	TCP	433a4c697261...	0	1	0	1	43	37	4	128
1315	134.185772	63992	12345	TCP		0	1	0	0	37	60	4	128
1394	156.629091	63974	12345	TCP	546f64617920...	0	1	0	1	36	72	4	128
1395	156.629122	12345	63974	TCP		0	1	0	0	72	69	4	128
1396	156.629225	12345	63992	TCP	433a4c697261...	0	1	0	1	60	37	4	128
1397	156.629249	63992	12345	TCP		0	1	0	0	37	102	4	128
1459	173.928065	63992	12345	TCP	796561682072...	0	1	0	1	37	102	4	128
1460	173.928095	12345	63992	TCP		0	1	0	0	102	80	4	128
1461	173.928177	12345	63974	TCP	433a4d616f72...	0	1	0	1	72	69	4	128

1462	173.928200	63974	12345	TCP		0	1	0	0	69	123	4	128
1532	187.517529	63974	12345	TCP	57616e6e6120..	0	1	0	1	69	123	4	128
1533	187.517556	12345	63974	TCP		0	1	0	0	123	104	4	128
1534	187.517636	12345	63992	TCP	433a4c697261..	0	1	0	1	102	80	4	128
1535	187.517663	63992	12345	TCP		0	1	0	0	80	146	4	128
1567	192.504605	63992	12345	TCP	6f6620636f75..	0	1	0	1	80	146	4	128
1568	192.504648	12345	63992	TCP		0	1	0	0	146	89	4	128
1569	192.504707	12345	63974	TCP	433a4d616f72..	0	1	0	1	123	104	4	128
1570	192.504737	63974	12345	TCP		0	1	0	0	104	140	4	128
1609	200.904598	63974	12345	TCP	626520686572..	0	1	0	1	104	140	4	128
1610	200.904638	12345	63974	TCP		0	1	0	0	140	120	4	128
1611	200.904699	12345	63992	TCP	433a4c697261..	0	1	0	1	146	89	4	128
1612	200.904727	63992	12345	TCP		0	1	0	0	89	171	4	128
1658	212.417877	63992	12345	TCP	616c72696768..	0	1	0	1	89	171	4	128
1659	212.417911	12345	63992	TCP		0	1	0	0	171	114	4	128
1660	212.417987	12345	63974	TCP	433a4d616f72..	0	1	0	1	140	120	4	128
1661	212.418016	63974	12345	TCP		0	1	0	0	120	173	4	128
1678	218.746600	63974	12345	TCP	756e74696c20..	0	1	0	1	120	173	4	128
1679	218.746630	12345	63974	TCP		0	1	0	0	173	131	4	128
1680	218.746709	12345	63992	TCP	433a4c697261..	0	1	0	1	171	114	4	128
1681	218.746750	63992	12345	TCP		0	1	0	0	114	191	4	128
1810	237.999628	63992	12345	TCP	62796521	0	1	0	1	114	191	4	128
1811	237.999657	12345	63992	TCP		0	1	0	0	191	118	4	128
1812	237.999749	12345	63974	TCP	433a4d616f72..	0	1	0	1	173	131	4	128
1813	237.999776	63974	12345	TCP		0	1	0	0	131	185	4	128
1839	241.170703	63974	12345	TCP	71756974	0	1	0	1	131	185	4	128
1840	241.170751	12345	63974	TCP		0	1	0	0	185	135	4	128
1841	241.170811	12345	63974	TCP	533a71756974	0	1	0	1	185	135	4	128
1842	241.170838	63974	12345	TCP		0	1	0	0	135	191	4	128
1843	241.171110	12345	63974	TCP		0	1	1	0	191	135	4	128
1844	241.171130	63974	12345	TCP		0	1	0	0	135	192	4	128
1845	241.171763	63974	12345	TCP		0	1	1	0	135	192	4	128
1846	241.171821	12345	63974	TCP		0	1	0	0	192	136	4	128
1917	244.104911	63992	12345	TCP	71756974	0	1	0	1	118	191	4	128
1918	244.104940	12345	63992	TCP		0	1	0	0	191	122	4	128
1919	244.104999	12345	63992	TCP	533a71756974	0	1	0	1	191	122	4	128
1920	244.105028	63992	12345	TCP		0	1	0	0	122	197	4	128
1921	244.105287	12345	63992	TCP		0	1	1	0	197	122	4	128
1922	244.105308	63992	12345	TCP		0	1	0	0	122	198	4	128
1923	244.107007	63992	12345	TCP		0	1	1	0	122	198	4	128
1924	244.107064	12345	63992	TCP		0	1	0	0	198	123	4	128

הדבר הראשון שנראה לעין הוא בMOVED סוג התעבורה – TCP בכל הפקודות. ובאמת זה המצב מכיוון שהגדכנו את השרת ברמת האפליקציה לעובוד עם חיבור TCP.

דבר נוסף שראים הוא בשכבות התעבורה, מספר הפורט 12345 חוזר על עצמו בכל הפקודות, ולא בצד מכיוון שהוא מספר הפורט שהקיצינו לשרת. לכן כל פאקטה source port שהיא 12345, משמעות הדבר שהשרת שלח את הפקודה וכל פאקטה destination port שהיא 12345, משמעות הדבר שהשרת קיבל את הפקודה מלוקה שלוח אותה.

כמו כן במספרי הפורט 63974 וגם 63992 חוזרים על עצמם בכל הפקודות, לעיתים מספר אחד ולעתים במספר השני. זה בגלל שמספרם הפורט האלוי מתקשרים המשתמשים לירן ומאור ולכן הם קבועים וחוזרים על עצמם.

כיצד נדע איזה מספר פорт שייר לאיזה משתמש? נעה על כך בהמשך, לבינתיים נניח שהמספר 63974 שייך לירן והמספר 63992 שייך למאור.

נשים לב לחיבור של המשתמשים "ירן" ו"מאור" לשרת. בחבילות הראשונות נוכל בבירור לראות את ה-*handshake 3 way* של כל אחד מהמשתמשים עם השרת:

No.	Time	Source Port	Destination Port	Protocol	TCP payload	Syn	Acknowledgment
295	8.333307	63974	12345	TCP		1	0
296	8.333386	12345	63974	TCP		1	1
297	8.333425	63974	12345	TCP		0	1
471	14.018493	63992	12345	TCP		1	0

בחבילה מס' 295 קורה 295 כוונה על נושא של ירן, כלומר בבקשת חיבור מהлокה מהמשתמש "ירן" והוא מהויה בבקשת חיבור מהлокה.

בחבילה מס' 296 קורה 296 כוונה על נושא של מאור, כלומר בבקשת חיבור של ירן למשתמש "ירן" והוא מהויה אישור בבקשת החיבור של המשתמש.

בחבילה מס' 297 קורה 297 כוונה על נושא של מאור, כלומר בבקשת חיבור מהлокה מהמשתמש "ירן" שמהויה את אישור התחברות הлокה, תהליך *3 way handshake* הסתיים ומכאן והילך כל הפקודות ששולח המשתמש "ירן" הן מהוצאה זו.

נשים לב שבפקודות 471, 472, 473 קורה אותו התהליך בבדיקה עבור המשתמש "מאור" שגם הוא התחבר לשרת.

471	14.018493	63992	12345	TCP	1	0
472	14.018565	12345	63992	TCP	1	1
473	14.018592	63992	12345	TCP	0	1

שאלה שנשאלת היא כיצד ידועי לשער את הפורט 63974 לחיבור של ירן ואת הפורט 63992 דואק לחיבור של מאור? הרי הסדר יכול להיות הפוך? תשובה אחת לשאלה היא אני, בתור מי שהריץ את הדוגמה מן הסתם יודע מי התחבר קודם... אז הפורט שמוסיע ראשון שייר למשתמש שהתחבר ראשון...

אבל זו לא התשובה שאנו מכחשים; התשובה המקצועית היא, נשים לב לפאקטה מס' 838:

838	60.531866	63974	12345	TCP	4c6972616e	0
839	60.531894	12345	63974	TCP		
▶ 0000	02 00 00 00 45 00 00 2d	63 83 40 00 80 06 00 00			E - - c @ .....	
▶ 0010	7f 00 00 01 7f 00 00 01	f9 e6 30 39 7f 87 1b ff			..... 09 .....	
▶ 0020	ae 28 e3 2d 50 18 00 ff	2c fe 00 00 4c 69 72 61			( --P -- , ... Lira	
▼ 0030	6e				n	

שמות לב שהפקטה נשאת את ה-payload בטור "Liran", זו הודעה הראשונה בהחלה שנשלחת לשרת, וזה כי הדבר הראשון שמשתמש עשו כשהוא פותח את האפליקציה זה להקליד את השם שלו לשרת.

בשים לב שפורט השולח הוא 63974 והמקבל הוא מבון השירות. אם כן בעת אנחנו יכולים בזוזאות לשירות את מספר הפורט 63974 למשתמש "ליין" ואילו את מספר הפורט الآخر 63992 לשירות למשתמש "מאור". ובאמת לפיו פאקטה מספר 840 רואים שהפורט 63992 שייר ל"מאור" לפי אותו:

840	64.083197	63992	12345	TCP	4d616f72	0
▶ 0000	02 00 00 00 45 00 00 2c	63 85 40 00 80 06 00 00			E - , c @ .....	
▶ 0010	7f 00 00 01 7f 00 00 01	f9 f8 30 39 d7 74 96 3c			..... 09 t < .....	
▶ 0020	ac b4 a9 5b 50 18 00 ff	06 00 00 00 4d 61 6f 72			[P ..... Maor	
▼ 0030					n	

באוטו אופן אפשר לבדוק את ניתוק המשתמשים מהשירות, נשים לב לפאקטות 1843-1846. נזכיר שבתמונה הבאה, הבית הימני הוא השדה `Fin`, משמאלו `Ack` ומשמאלו `Syn` שמואפס בכל השירות.

מספר מסמך	כתובת IP	פורט של השולח	פורט של מקבל	טיפוס	טבלה	טבלה	טבלה
1843	241.171110	12345	63974	TCP	0	1	1
1844	241.171130	63974	12345	TCP	0	1	0
1845	241.171763	63974	12345	TCP	0	1	1
- 1846	241.171821	12345	63974	TCP	0	1	0

אלו הפאקטות האחרונות שהוקלטו עבור המשתמש "ליין" שבפורט 63974.

בשים לב לתהליך `4 way handshake` 4 עבור הניתוק של ליין מהשירות:

בחיבור מס' 1843 קורה `fin=1, ack=1`, וזו חיבורה שהשרת שולח לליין. כאן השירות שולח בקשה לניתוק החיבור. (בתגובה לפאקטה קודמת שבה ליין מבון שלח הודעת `quit` לשרת).

בחיבור מס' 1844 קורה `fin=0, ack=1`, וזו חיבורה ששלוח ליין לשרת. כאן ליין מאשר את בקשת השירות לניתוק החיבור.

בחיבור מס' 1845 קורה `fin=1, ack=1`, וזו חיבורה ששלוח ליין לשרת. כאן ליין שולח בקשה לניתוק החיבור.

בחיבור מס' 1846 קורה `fin=0, ack=1`, וזו חיבורה ששלוח השירות לליין. כאן השירות מאשר את>b>kusha של ליין והחיבור נסגר סופית. זו הפאקטה האחורונה שעברה בין ליין לשרת או להיפר.

התהליך חוזר על עצמו עבור המשתמש "מאור" שבפורט 63992 ובאמת 4 הפאקטות האחרונות של ההקלטה הן הניתוק של מאור מהשירות:

1921	244.105287	12345	63992	TCP	0	1	1
1922	244.105308	63992	12345	TCP	0	1	0
1923	244.107007	63992	12345	TCP	0	1	1
1924	244.107064	12345	63992	TCP	0	1	0

בשכבה הרשת, למעשה source ip = destination ip = 127.0.0.1 כי התעבורה היא על המחשב המקומי ובכתוות loopback.

No.	Time	Source Address	Destination Address	Source Port	Destination Port	Protocol	TCP payload	Syn	Ack
295	8.333307	127.0.0.1	127.0.0.1	63974	12345	TCP		1	0
296	8.333386	127.0.0.1	127.0.0.1	12345	63974	TCP		1	1
297	8.333425	127.0.0.1	127.0.0.1	63974	12345	TCP		0	1
471	14.018493	127.0.0.1	127.0.0.1	63992	12345	TCP		1	0
472	14.018565	127.0.0.1	127.0.0.1	12345	63992	TCP		1	1
473	14.018592	127.0.0.1	127.0.0.1	63992	12345	TCP		0	1
838	60.531866	127.0.0.1	127.0.0.1	63974	12345	TCP	4c6972616e	0	1
840	64.083197	127.0.0.1	127.0.0.1	63992	12345	TCP	4d616f72	0	1
841	64.083226	127.0.0.1	127.0.0.1	12345	63992	TCP		0	1
983	71.983371	127.0.0.1	127.0.0.1	63974	12345	TCP	69747a696b	0	1
984	71.983421	127.0.0.1	127.0.0.1	12345	63974	TCP		0	1
985	71.983615	127.0.0.1	127.0.0.1	12345	63974	TCP	533a55736572...	0	1
986	71.983633	127.0.0.1	127.0.0.1	63974	12345	TCP		0	1
987	75.264535	127.0.0.1	127.0.0.1	63992	12345	TCP	67616c	0	1
988	75.264590	127.0.0.1	127.0.0.1	12345	63992	TCP		0	1
989	75.264798	127.0.0.1	127.0.0.1	12345	63992	TCP	533a55736572...	0	1
990	75.264821	127.0.0.1	127.0.0.1	63992	12345	TCP		0	1
1009	81.062028	127.0.0.1	127.0.0.1	63974	12345	TCP	7961726f6e	0	1
1010	81.062065	127.0.0.1	127.0.0.1	12345	63974	TCP		0	1
1011	81.062286	127.0.0.1	127.0.0.1	12345	63974	TCP	533a55736572...	0	1
1012	81.062309	127.0.0.1	127.0.0.1	63974	12345	TCP		0	1
1132	106.289869	127.0.0.1	127.0.0.1	63992	12345	TCP	4c6972616e	0	1
1133	106.289899	127.0.0.1	127.0.0.1	12345	63992	TCP		0	1
1134	106.290399	127.0.0.1	127.0.0.1	12345	63992	TCP	533a55736572...	0	1
1135	106.290419	127.0.0.1	127.0.0.1	63992	12345	TCP		0	1
1163	111.394157	127.0.0.1	127.0.0.1	63974	12345	TCP	4d616f72	0	1
1164	111.394187	127.0.0.1	127.0.0.1	12345	63974	TCP		0	1
1165	111.394434	127.0.0.1	127.0.0.1	12345	63974	TCP	533a55736572...	0	1
1166	111.394449	127.0.0.1	127.0.0.1	63974	12345	TCP		0	1
1210	119.144548	127.0.0.1	127.0.0.1	63974	12345	TCP	686579206d61...	0	1
1211	119.144582	127.0.0.1	127.0.0.1	12345	63974	TCP		0	1
1212	119.144668	127.0.0.1	127.0.0.1	12345	63992	TCP	433a4c697261...	0	1
1213	119.144695	127.0.0.1	127.0.0.1	63992	12345	TCP		0	1
1232	128.222671	127.0.0.1	127.0.0.1	63992	12345	TCP	776861747320...	0	1
1233	128.222703	127.0.0.1	127.0.0.1	12345	63992	TCP		0	1
1234	128.222759	127.0.0.1	127.0.0.1	12345	63974	TCP	433a4d616f72...	0	1
1235	128.222783	127.0.0.1	127.0.0.1	63974	12345	TCP		0	1
1312	134.185624	127.0.0.1	127.0.0.1	63974	12345	TCP	49276d20676f...	0	1
1313	134.185661	127.0.0.1	127.0.0.1	12345	63974	TCP		0	1
1314	134.185743	127.0.0.1	127.0.0.1	12345	63992	TCP	433a4c697261...	0	1
1315	134.185772	127.0.0.1	127.0.0.1	63992	12345	TCP		0	1
1394	156.629091	127.0.0.1	127.0.0.1	63974	12345	TCP	546f64617920...	0	1
1395	156.629122	127.0.0.1	127.0.0.1	12345	63974	TCP		0	1
1396	156.629225	127.0.0.1	127.0.0.1	12345	63992	TCP	433a4c697261...	0	1
1397	156.629249	127.0.0.1	127.0.0.1	63992	12345	TCP		0	1
1459	173.928065	127.0.0.1	127.0.0.1	63992	12345	TCP	796561682072...	0	1
1460	173.928095	127.0.0.1	127.0.0.1	12345	63992	TCP		0	1
1461	173.928177	127.0.0.1	127.0.0.1	12345	63974	TCP	433a4d616f72...	0	1
1462	173.928200	127.0.0.1	127.0.0.1	63974	12345	TCP		0	1
1532	187.517529	127.0.0.1	127.0.0.1	63974	12345	TCP	57616e6e6120...	0	1

צינה מקודם הפאקטה עם ה-payload : "Liran".

במובן שבקובץ הפקלה שמצורף אפשר לעין בכל ההודעות ברמת האפליקציה, כמו:

ליין שלוח לשרת "Itzik" בפקטה 983 (משתמש אליו רוצה לשלוח הודעה)

No.	Time	Source Address	Destination Address	Source Port	Destination Port	Protocol	TCP payload
295	8.333307	127.0.0.1	127.0.0.1	63974	12345	TCP	
296	8.333386	127.0.0.1	127.0.0.1	12345	63974	TCP	
297	8.333425	127.0.0.1	127.0.0.1	63974	12345	TCP	
471	14.018493	127.0.0.1	127.0.0.1	63992	12345	TCP	
472	14.018565	127.0.0.1	127.0.0.1	12345	63992	TCP	
473	14.018592	127.0.0.1	127.0.0.1	63992	12345	TCP	
838	60.531866	127.0.0.1	127.0.0.1	63974	12345	TCP	4c6972616e
839	60.531894	127.0.0.1	127.0.0.1	12345	63974	TCP	
840	64.083197	127.0.0.1	127.0.0.1	63992	12345	TCP	4d616f72
841	64.083226	127.0.0.1	127.0.0.1	12345	63992	TCP	
983	71.983371	127.0.0.1	127.0.0.1	63974	12345	TCP	69747a696b
984	71.983421	127.0.0.1	127.0.0.1	12345	63974	TCP	
...0 0000 0000 0000 = Fra				0000 02 00 00 00 45 00 00 2d 64 0e 40 00 80 06 00 00			E - - d @ .....
Time to Live: 128				0010 7f 00 00 01 7f 00 00 01 f9 e6 30 39 7f 87 1c 04			..... 09 .....
Protocol: TCP (6)				0020 ae 28 e3 2d 50 18 00 ff 0a e6 00 00 69 74 7a 69			( -P ..... itzi
Header Checksum: 0x0000 [				0030 6b			k
[Header checksum status:							

הشرط משב לירן: UserNotFound: S בפקטה 985 כי איזיק הוא לא משתמש קיים

No.	Time	Source Address	Destination Address	Source Port	Destination Port	Protocol	TCP payload	Sy
295	8.333307	127.0.0.1	127.0.0.1	63974	12345	TCP		1
296	8.333386	127.0.0.1	127.0.0.1	12345	63974	TCP		1
297	8.333425	127.0.0.1	127.0.0.1	63974	12345	TCP		0
471	14.018493	127.0.0.1	127.0.0.1	63992	12345	TCP		1
472	14.018565	127.0.0.1	127.0.0.1	12345	63992	TCP		1
473	14.018592	127.0.0.1	127.0.0.1	63992	12345	TCP		0
838	60.531866	127.0.0.1	127.0.0.1	63974	12345	TCP	4c6972616e	0
839	60.531894	127.0.0.1	127.0.0.1	12345	63974	TCP		0
840	64.083197	127.0.0.1	127.0.0.1	63992	12345	TCP	4d616f72	0
841	64.083226	127.0.0.1	127.0.0.1	12345	63992	TCP		0
983	71.983371	127.0.0.1	127.0.0.1	63974	12345	TCP	69747a696b	0
984	71.983421	127.0.0.1	127.0.0.1	12345	63974	TCP		0
985	71.983615	127.0.0.1	127.0.0.1	12345	63974	TCP	533a55736572...	0

“ההודעה שלין שלח למאור דרך השרת בפקטה 1532 wanna come over? We can order pizza”

1532	187.517529	127.0.0.1	127.0.0.1	63974	12345	TCP	57616e6e6120...	0
1533	187.517556	127.0.0.1	127.0.0.1	12345	63974	TCP		0
...0 0000 0000 0000 = Fra	0000	02 00 00 00 45 00 00 4b	65 fa 40 00 80 06 00 00	... E K e @ ...				
Time to Live: 128	0010	7f 00 00 01 7f 00 00 01	f9 e6 30 39 7f 87 1c 43	..... 09 .. C				
Protocol: TCP (6)	0020	ae 28 e3 a7 50 18 00 ff	6f 1e 00 00 57 61 6e 6e	( P o Wann				
Header Checksum: 0x0000 [	0030	61 20 63 6f 6d 65 20 6f	76 65 72 3f 20 77 65 20	a come o ver? we				
[Header checksum status:	0040	63 61 6e 20 6f 72 64 65	72 20 70 69 7a 7a 61	can orde r pizza				
Source Address: 127.0.0.1								

מלבד בדוגמאות האלו ניתן למצוא את כל שאר הבודדות והפआקומות בהזוא הלכידם המאוחר.

נשים לב בשכבת הרשת גם לעובדה ששדה Time To Live המהווה את המספר המקסימלי של קפיצות בין רכיבי רשת עד שהפאקטעה נזרקת, תמיד קבוע, ותמיד מספר גדול יחסית, 128. הסיבה היא שאנו מוצאים פעולות על המחשב באופן מקומי ולבן הפקטות לא קופצות בין רכיבי רשת, لكن השדה מקבל ערך גובה וקבוע.