

---

## עבודה עם Wireshark - חלק א'

מאת עידו קנר

---

### הקדמה

מאמר זה הינו חלק ראשון בסדרת מאמרים אשר נועדה לתת דגשים לעבודה עם Wireshark רבים רואים כלי זה כאמצעי להאזין (להסניף) לחבילות מידע העוברות ברשת, אך דווקא לפעולה זו יש עוד תכנות אשר יכולות להיות לעזר רב יותר לפעמים, כדוגמת fiddler ב-Windows כאשר רוצים לטפל ב-HTTP. הכוח של Wireshark הוא בסינון ובניתוח המידע, והיכולת להשתמש בו בשביל להבין מעבר לרשימה פשוטה של פקטות מה קרה בתקשורת, וזאת מה שסדרת המאמרים המובאת מנסה ללמד.

העבודה ב-Wireshark דורשת הכירות כלשהי עם עולם ה-TCP/IP וכיצד מבנה של כמוסות ה-IP ו-TCP בנויות. גם סדרת המאמרים שלי בנושא יוצאת מנקודת הנחה זו.

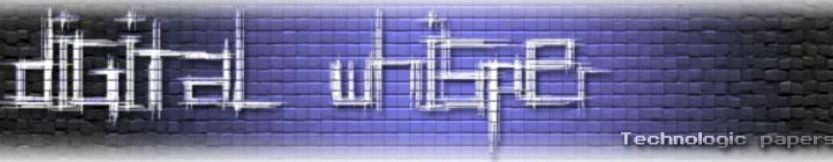
### התחלת עבודה

Wireshark משתמש בספרייה הנקראת PCAP על מנת להאזין למידע. ניתן להשתמש במספר "טעמים", לשם העבודה עם Wireshark, תלוי כמובן במערכת ההפעלה שלכם.

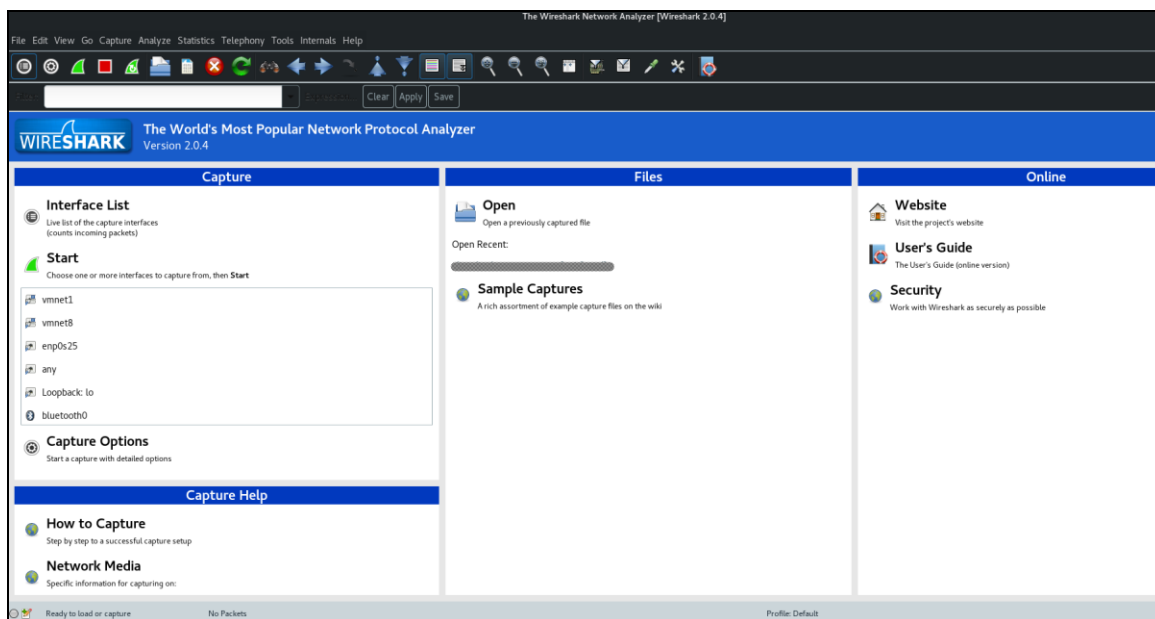
כאשר מדובר בלינוקס:

- Wireshark GTK
- Wireshark Qt
- tshark/Wireshark cli

גרסת tshark/Wireshark cli, מאפשרת להשתמש בספריית ה-PCAP על מנת לתפוס תעבורה ולנתח אותה בצורה פשוטה מאוד, אך מאמר זה ישתמש בגרסה הגרפית של Wireshark על מנת להבין את המידע.



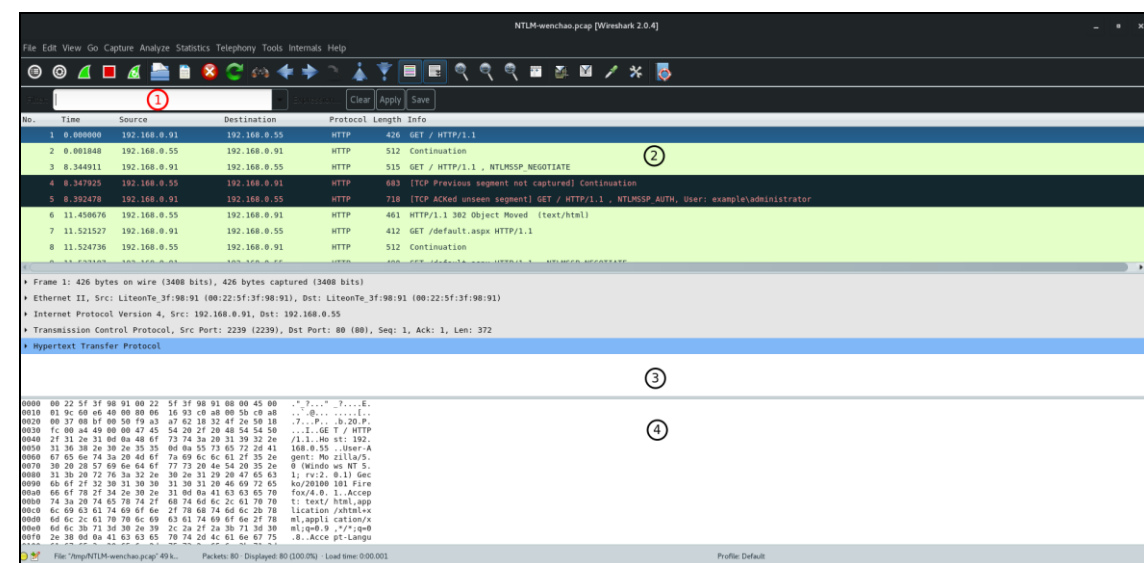
ברירת המחדל הממשק יראה כך כאשר אנחנו פותחים אותו (ה-Device השונים יכולים להשתנות ממחשב למחשב ומערכת הפעלה אחת לשניה):



[אני משתמש כאן בגרסת GTK בלינוקס, אך גם לווינדוז יש ממשק גרפי ל-Wireshark]

במידה ויש צורך להאזין לכל התקשורת, תבחר האפשרות ב-Interface List של Any, במידה ויש צורך ב-Device מסוים, יש לבחור בו, כל עוד יש צורך להאזין לתעבורה. במאמר זה לא אדבר על האזנה, אלא על ניתוח המידע, ולכן אשתמש ביכולת לטעון קבצי PCAP.

בשביל מאמר זה ניגשתי [לאתר של Wireshark המכיל דוגמאות](#) של האזנות שנאספו, והורדתי משם מספר דוגמאות, בהם הדגמה של צורת ההזדהות וניהול אבטחה הנקראת [NTLM](#). טענתי את קובץ ה-PCAP וקיבלתי מספר שמספרי אותו ל-4 חלקים:



עבודה עם Wireshark חלק א'

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)

הנה הסבר קצר על ארבעת החלקים שסימנתי:

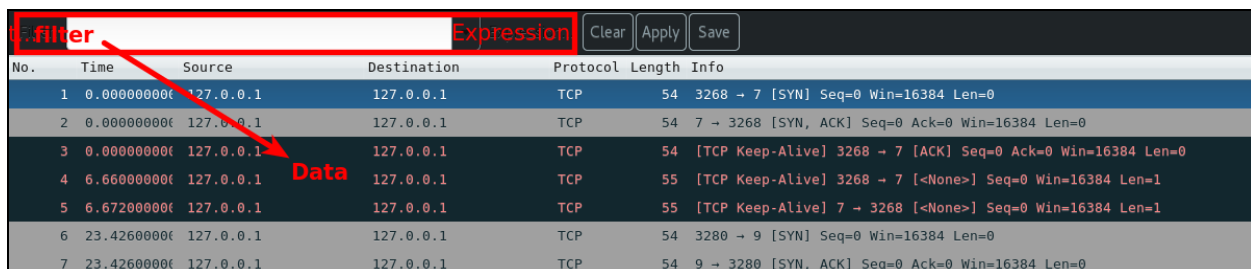
1. חלק זה הוא חלק המאפשר לבצע filter למידע. אפשר להשתמש בכל אחד מהחלקים של OSI בשביל ליצור סינון. אבל זה לא בדיוק נכון! בהמשך אסביר זאת יותר לעומק.
2. חלק זה הוא רשימה של תעבורה. החלק מכיל גם Raw Packets וגם את האיסוף שלהם, ובכך למשל נראה 4 פקטות של TCP למשל, אך רשומה אחת של HTTP שהורכבה מארבעת הפקטות הללו.
3. חלק זה הוא חלק המראה כיצד המידע בנוי, וכאן אנחנו רואים כיצד ה-OSI מגיע לידי ביטוי, אך שוב, זה לא מדויק, ולימוד טוב יותר בנושא יינתן בהמשך.
4. חלק זה הוא חלק המסוגל להראות מידע בצורה הטבעית שהוא הגיע עם מידע של ביטים או הקסה-דצימלי של הערכים. למשל במידה והיה צריך להגיע \חל\ ואתם רואים שמפרש שלכם נכשל? כאן ניתן לאמת האם זה המצב.

בחלקים 2 עד 4, במידה ותהיה לחיצה על המקש הימני של העכבר, יפתח תפריט המאפשר להתמודד עם החלקים בצורה שונה, לא את כל האפשרויות אוכל לכסות במאמר זה (על כלל חלקיו).

## סינון מידע

לרוב, כאשר מקבלים מידע מקבצי PCAP או האזנה למידע, הוא מכיל המון סוגים של פרוטוקולים, פורטים, כתובות IP וכיו"ב.

במערכת לא עמוסה, אולי אפשר להתמודד עם זה, אך ככול שיש יותר מידע מתעבורה, ככה קשה יותר למצוא את התעבורה המתאימה לניתוח. לשם כך יש את היכולת לבצע סינון למידע:



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	54	3268 → 7 [SYN] Seq=0 Win=16384 Len=0
2	0.000000000	127.0.0.1	127.0.0.1	TCP	54	7 → 3268 [SYN, ACK] Seq=0 Ack=0 Win=16384 Len=0
3	0.000000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3268 → 7 [ACK] Seq=0 Ack=0 Win=16384 Len=0
4	6.660000000	127.0.0.1	127.0.0.1	TCP	55	[TCP Keep-Alive] 3268 → 7 [None] Seq=0 Win=16384 Len=1
5	6.672000000	127.0.0.1	127.0.0.1	TCP	55	[TCP Keep-Alive] 7 → 3268 [None] Seq=0 Win=16384 Len=1
6	23.426000000	127.0.0.1	127.0.0.1	TCP	54	3280 → 9 [SYN] Seq=0 Win=16384 Len=0
7	23.426000000	127.0.0.1	127.0.0.1	TCP	54	9 → 3280 [SYN, ACK] Seq=0 Ack=0 Win=16384 Len=0

היות ויש בממשק אצלי בעיית תצוגה קלה, הדגשתי דברים באדום היכן שהטקסט אינו ברור.

התחביר לסינון שייך לספריית ה-PCAP, וככזו, כל תוכנה המשתמשת ב-PCAP ולא רק Wireshark, המאפשרת לבצע סינון, תשתמש באותו התחביר, אלא אם צוין בה אחרת. יש ב-PCAP אשר בחרתי מספר תעבורות של TCP, למרות שבתמונה רואים רק סוג אחד.

במידה ויש צורך לראות את כל פעולות Three Way Handshake ניתן יהיה לחפש את פעולת ה-syn:

No.	Time	Source	Destination	Protocol	Length	Info
27	64.2880000	127.0.0.1	127.0.0.1	TCP	300	[TCP Retransmission] 443 → 3304 [<None>] Seq=3909156864 Win=16384 Len=246
28	64.2990000	127.0.0.1	127.0.0.1	TCP	77	[TCP Retransmission] 443 → 3304 [<None>] Seq=3741450240 Win=16384 Len=23
29	152.3930000	127.0.0.1	127.0.0.1	TCP	54	3454 → 5222 [SYN] Seq=0 Win=16384 Len=0
30	152.3930000	127.0.0.1	127.0.0.1	TCP	54	5222 → 3454 [SYN, ACK] Seq=0 Ack=0 Win=16384 Len=0
31	152.3930000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3454 → 5222 [ACK] Seq=0 Ack=0 Win=16384 Len=0
32	152.3940000	127.0.0.1	127.0.0.1	TCP	93	[TCP Out-Of-Order] 3454 → 5222 [<None>] Seq=0 Win=16384 Len=39
33	152.4050000	127.0.0.1	127.0.0.1	XMPP/XML	154	[TCP Previous segment not captured] STREAM → localhost

ואכן יש פקטות שהן מכילות פעולות syn אבל לא רק. כלומר הסינון שבוצע אינו היה מספיק טוב, ובפרק "סינון נכון" אסביר מדוע.

חשוב להדגיש כי בשביל להפעיל את הפילטר יש ללחוץ על Apply או על Enter בשורת הטקסט. הסיבה שבה השתמשתי ב-tcp.ack היא על מנת להגדיר לפי שדה או תכונה של הפרוטוקול מה אני מעוניין לחפש. היות ואין למשל ברמת ה-IP או ב-UDP פעולה של Three Way Handshake אני לא אשתמש בהם לשם כך.

היות והפרוטוקול XMPP מבוסס TCP ונמצא בשכבה 7, התקבל עוד מידע אשר לא בהכרח מעניין, ולכן יש צורך לבצע "זיקוק" של המידע.

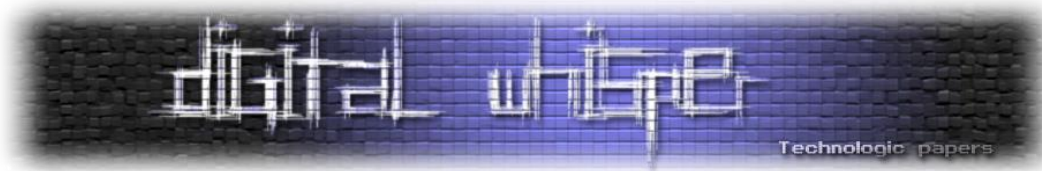
No.	Time	Source	Destination	Protocol	Length	Info
1	0.0000000	127.0.0.1	127.0.0.1	TCP	54	3268 → 7 [SYN] Seq=0 Win=16384 Len=0
2	0.0000000	127.0.0.1	127.0.0.1	TCP	54	7 → 3268 [SYN, ACK] Seq=0 Ack=0 Win=16384 Len=0
6	23.4260000	127.0.0.1	127.0.0.1	TCP	54	3280 → 9 [SYN] Seq=0 Win=16384 Len=0
7	23.4260000	127.0.0.1	127.0.0.1	TCP	54	9 → 3280 [SYN, ACK] Seq=0 Ack=0 Win=16384 Len=0
10	25.7480000	127.0.0.1	127.0.0.1	TCP	56	[TCP Previous segment not captured] 3280 → 9 [<None>] Seq=16777216 Win=16384 Len=2
11	62.0170000	127.0.0.1	127.0.0.1	TCP	54	3302 → 443 [FIN] Seq=0 Win=16384 Len=0
12	62.0170000	127.0.0.1	127.0.0.1	TCP	54	443 → 3302 [SYN, ACK] Seq=0 Ack=0 Win=16384 Len=0
14	62.9260000	127.0.0.1	127.0.0.1	TCP	132	[TCP Retransmission] 3302 → 443 [<None>] Seq=0 Win=16384 Len=78
15	62.9520000	127.0.0.1	127.0.0.1	TCP	1139	[TCP Retransmission] 443 → 3302 [<None>] Seq=0 Win=16384 Len=1085
16	63.0150000	127.0.0.1	127.0.0.1	SSLv3	250	[TCP Previous segment not captured] Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
17	63.0310000	127.0.0.1	127.0.0.1	SSLv3	121	[TCP Previous segment not captured] Change Cipher Spec, Encrypted Handshake Message
18	63.2210000	127.0.0.1	127.0.0.1	SSLv3	77	[TCP Previous segment not captured] Encrypted Alert
19	64.1820000	127.0.0.1	127.0.0.1	TCP	54	3304 → 443 [SYN] Seq=0 Win=16384 Len=0

חשוב להדגיש כי ככול שהפילטר כללי יותר, כך יתקבל עוד מידע, אלא אם יודעים להשתמש נכון בסינון עצמו.

במקרה הנוכחי, שרשרתי מספר בדיקות:

- העבר רק פקטת TCP שיש לה ACK
- אל תציג פקטות שהם עם keep alive גם מתבצע עם SYN
- ואל תספק לי מידע שהוא לפרוטוקול XMPP.

אך גם זה אינו נכון, ומי שמכיר ממש טוב כיצד TCP עובד, כבר בטוח שמנחש את הסיבה לכך. הסיבה לבעיית הסינון תקבל מענה כאמור, בפרק "סינון נכון".



## תחביר

התחביר הקיים בפילטר מכיל את התחביר הבא:

### טיפוסי נתונים:

לכל שדה בפרוטוקולים, יש סוג של טיפוס נתונים. טיפוסי הנתונים יכולים להיות אחד מאלו:

- ASN.1 object identifier
- Boolean
- Character string
- Compiled Perl-Compatible Regular Expression (GRegex) object
- Date and time
- Ethernet or other MAC address
- EUI64 address
- Floating point (double-precision)
- Floating point (single-precision)
- Frame number
- Globally Unique Identifier
- IPv4 address
- IPv6 address
- IPX network number
- Label
- Protocol
- Sequence of bytes
- Signed integer, 1, 2, 3, 4, or 8 bytes
- Time offset
- Unsigned integer, 1, 2, 3, 4, or 8 bytes

## אופרטורים:

סימנים	פעולה	הסבר
eq, ==	שוויון	האופרטור eq והאופרטור == שניהם מבצעים בדקת שוויון
ne, !=	אי שוויון	האופרטור ne והאופרטור != (שמאל לימין) שניהם מבצעים בדיקות לאי שוויון
gt, >	גדול מ	האופרטור gt והאופרטור > (שמאל לימין) בודקים האם ערך מסוים גדול מערך אחר
lt, <	קטן מ	האופרטור lt והאופרטור < (שמאל לימין) בודקים האם ערך מסוים קטן מערך אחר
ge, >=	גדול שווה	האופרטור ge והאופרטור >= (שמאל לימין) בודקים האם ערך גדול או שווה לערך אחר
<= Le,	קטן שווה	האופרטור le והאופרטור <= (שמאל לימין) בודקים האם ערך קטן או שווה לערך אחר
and, &&	וגם	האופרטור and (אותיות קטנות) והאופרטור && מאפשרים לכלול 2 ביטויים אשר יתחברו ביחד.
or,	או	האופרטור or (אותיות קטנות) והאופרטור    מאפשרים לכלול ביטוי אחד או ביטוי אחר כחלק מהפילטר
()	קבוצה	האופרטור () מאפשר לכלול בתוכו מספר ביטויים אשר התוצאה שלהם תתאים לתנאים ביחד, כמו ביטוי מתמטי עם סוגריים.
not, !	שלילה	האופרטור not (אותיות קטנות) והאופרטור ! מאפשרים לשלול תחביר
[]	חיתוך	האופרטור [] מאפשר לחתוך חלק ממחרזת או מערך של בתיים. דוגמה: 00:00:83 == th.src[0:3]
{}	סדרה	האופרטור {} מאפשר ליצור סדרה של מידע ולבדוק עליו אם קיים. דוגמה: tcp.port in {80 443 8080}
&	חיבור bitwise	האופרטור & מאפשרים לעשות פעולות bitwise של חיבור. שימוש באופרטור יכול להראות כך: tcp.flags & 0x02

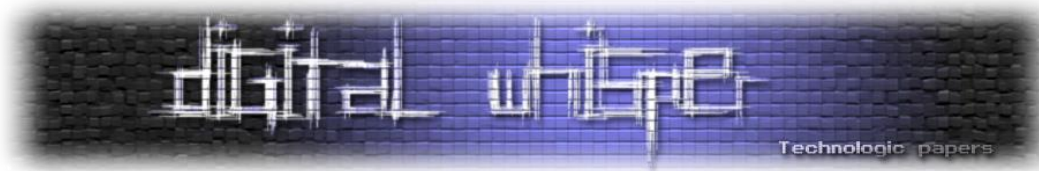
הדגמה:

```
ip.addr == 192.168.4.32
```

בדיקה האם כתובת כלשהי (בין אם נכנסת או יוצאת) היא 192.168.4.32. רק פקטות שמכילות את הכתובת הזו יוצגו, ובמידה ואינן קיימות בכלל, לא תהיה רשימה בכלל.

עבודה עם - Wireshark חלק א'

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)



## חיפוש מחרוזות:

ניתן לחפש מחרוזות ב-Wireshark באמצעות אחת משתי הפקודות הבאות:

- **contains** - האם פרוטוקול או שדה מסוים בפרוטוקול מכיל מחרוזת מסוימת.
  - **matches** - שימוש ב-regex של פרל (preg) למציאת תבנית מסוימת.
- חשוב להדגיש כי contains ו-matches אינם יכולים להיות בחיפוש על שדות שהם אינם מחרוזות וטקסט, כך שלא ניתן להשתמש בהם על שדה של פורט למשל.

הדגמה ל-contains:

```
http contains "https://www.Wireshark.org"
```

הדגמה ל-matches:

```
wsp.user_agent matches "(?i)cldc"
```

חיפוש תחת פרוטוקול WSP את user agent לפי תבנית מסוימת, שבה הבדיקה היא גם לאותיות גדולות וקטנות (i?) ואז חיפוש התווים של cldc.  
בעבודה עם מחרוזות, ישנם 2 פונקציות:

- upper
- lower

הדגמה:

```
upper(http) contains "GOOGLE"
```

```
lower(mount.dump.hostname) == "angel"
```

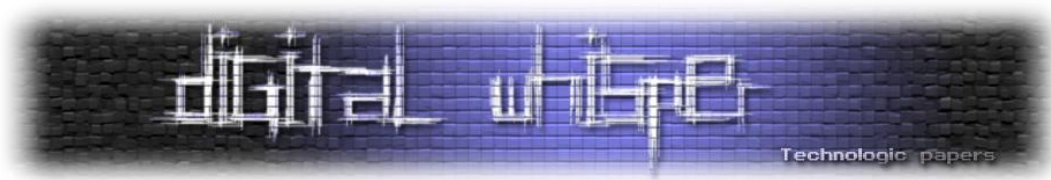
## חיפוש CIDR:

כאשר מחפשים כתובות IP, ניתן להשתמש בחיפוש של טווח על ידי CIDR:

```
ip.dst == 192.168.1.1/24
```

ההדגמה תחפש את כל היעד בטווח של 192.168.1.0 עד 192.168.1.254.





## סינון נכון

בשביל לקבל את כל פעולות ה-SYN אשר קיימים, יש להשתמש מערכת הסינון בצורה אחרת במקצת ממה שהצגתי בהתחלה. כידוע, ב-TCP יש מצב של "דגלים". עבור פעולת Three Way Handshake ישנם מספר דגלים. היות וכפי שהוסבר בהתחלה, האובייקטים של Wireshark מבוססי שכבות OSI, ניתן לגשת ב-TCP לדגלים ולבדוק האם הם "דולקים" או לא.

לשם כך, הסינון יראה כך:

```
tcp.flags.syn == 1
```

הסינון הנ"ל, יספק למעשה סינון אמיתי לפי דגל ה-syn:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	54	3268 → 7 [SYN] Seq=0 Win=16384 Len=0
2	0.000000000	127.0.0.1	127.0.0.1	TCP	54	7 → 3268 [SYN, ACK] Seq=0 Ack=0 Win=16384 Len=0
6	23.426000000	127.0.0.1	127.0.0.1	TCP	54	3280 → 9 [SYN] Seq=0 Win=16384 Len=0
7	23.426000000	127.0.0.1	127.0.0.1	TCP	54	9 → 3280 [SYN, ACK] Seq=0 Ack=0 Win=16384 Len=0
11	62.817000000	127.0.0.1	127.0.0.1	TCP	54	3302 → 443 [SYN] Seq=0 Win=16384 Len=0
12	62.817000000	127.0.0.1	127.0.0.1	TCP	54	443 → 3302 [SYN, ACK] Seq=0 Ack=0 Win=16384 Len=0
19	64.182000000	127.0.0.1	127.0.0.1	TCP	54	3304 → 443 [SYN] Seq=0 Win=16384 Len=0
20	64.182000000	127.0.0.1	127.0.0.1	TCP	54	443 → 3304 [SYN, ACK] Seq=0 Ack=0 Win=16384 Len=0
29	152.393000000	127.0.0.1	127.0.0.1	TCP	54	3454 → 5222 [SYN] Seq=0 Win=16384 Len=0
30	152.393000000	127.0.0.1	127.0.0.1	TCP	54	5222 → 3454 [SYN, ACK] Seq=0 Ack=0 Win=16384 Len=0

מה קורה אבל, כאשר יש צורך לראות רק אל הפקטות אשר קיבלו ACK אבל ללא SYN

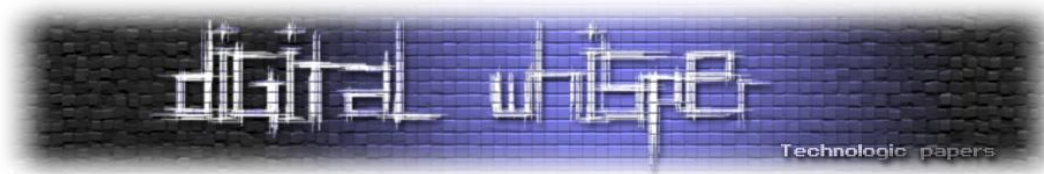
ובכן יש לרשום זאת כך:

```
tcp.flags.syn == 0 && tcp.flags.ack == 1
```

No.	Time	Source	Destination	Protocol	Length	Info
3	0.000000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3268 → 7 [ACK] Seq=0 Ack=0 Win=16384 Len=0
8	23.426000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3280 → 9 [ACK] Seq=0 Ack=0 Win=16384 Len=0
13	62.817000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3302 → 443 [ACK] Seq=0 Ack=0 Win=16384 Len=0
21	64.182000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3304 → 443 [ACK] Seq=0 Ack=0 Win=16384 Len=0
31	152.393000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3454 → 5222 [ACK] Seq=0 Ack=0 Win=16384 Len=0
42	183.124000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3539 → 17 [ACK] Seq=0 Ack=0 Win=16384 Len=0
46	254.660000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3645 → 21 [ACK] Seq=0 Ack=0 Win=16384 Len=0
52	273.731000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 20 → 3657 [ACK] Seq=0 Ack=0 Win=16384 Len=0
56	346.699000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3694 → 110 [ACK] Seq=0 Ack=0 Win=16384 Len=0
64	348.408000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3696 → 110 [ACK] Seq=0 Ack=0 Win=16384 Len=0

ולמעשה ניתן לסנן טוב יותר את התוצאות בהתאם לצורך.

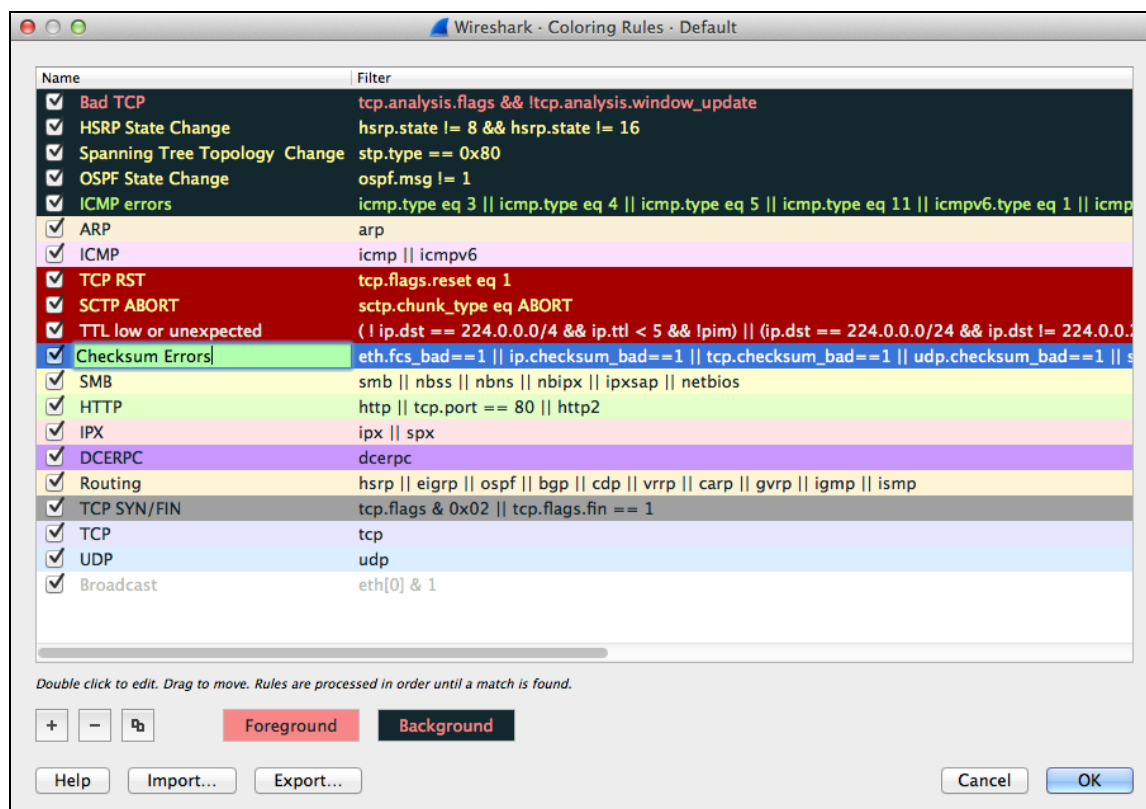




## ניתוח מידע

בחלק הקודם, סיימתי עם תמונה אשר מציגה מידע כאשר הרקע הוא בצבע שחור והטקסט הוא בצבע אדום. בברירת המחדל, במידה ולא שונו צבעי התוכנה, זה אומר כי ישנה בעיה בפקטות המוצגות כך.

צבעי ברירת המחדל מייצגים את המידע הבא:



[התמונה לקחה מהתיעוד הרשמי של Wireshark]

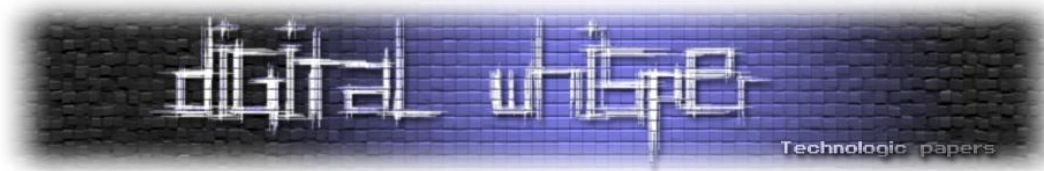
עכשיו כאשר יש צבעים אשר ברורים מה הם מייצגים, יש צורך לנתח מה קרה. היות ובתמונה הזו:

No.	Time	Source	Destination	Protocol	Length	Info
3	0.000000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3268 → 7 [ACK] Seq=0 Ack=0 Win=16384 Len=0
8	23.426000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3280 → 9 [ACK] Seq=0 Ack=0 Win=16384 Len=0
13	62.817000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3302 → 443 [ACK] Seq=0 Ack=0 Win=16384 Len=0
21	64.182000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3304 → 443 [ACK] Seq=0 Ack=0 Win=16384 Len=0
31	152.393000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3454 → 5222 [ACK] Seq=0 Ack=0 Win=16384 Len=0
42	183.124000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3539 → 17 [ACK] Seq=0 Ack=0 Win=16384 Len=0
46	254.660000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3645 → 21 [ACK] Seq=0 Ack=0 Win=16384 Len=0
52	273.731000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 20 → 3657 [ACK] Seq=0 Ack=0 Win=16384 Len=0
56	346.600000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3604 → 110 [ACK] Seq=0 Ack=0 Win=16384 Len=0

המידע בצבע אדום על רקע שחור, ניתן לדעת כי יש איזושהי בעיה, וצריך להתחיל ולנתח אותה.

עבודה עם Wireshark - חלק א'

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)



## הניתוח יתבצע בצורה פשוטה מאוד:

No.	Time	Source	Destination	Protocol	Length	Info
3	0.000000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3268 → 7 [ACK] Seq=0 Ack=0 Win=16384 Len=0
8	23.426000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3280 → 9 [ACK] Seq=0 Ack=0 Win=16384 Len=0
13	62.817000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3302 → 443 [ACK] Seq=0 Ack=0 Win=16384 Len=0
21	64.182000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3304 → 443 [ACK] Seq=0 Ack=0 Win=16384 Len=0
31	152.393000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3454 → 5222 [ACK] Seq=0 Ack=0 Win=16384 Len=0
42	183.124000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3539 → 17 [ACK] Seq=0 Ack=0 Win=16384 Len=0
46	254.660000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3645 → 21 [ACK] Seq=0 Ack=0 Win=16384 Len=0
52	273.731000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 20 → 3657 [ACK] Seq=0 Ack=0 Win=16384 Len=0
56	346.600000000	127.0.0.1	127.0.0.1	TCP	54	[TCP Keep-Alive] 3694 → 110 [ACK] Seq=0 Ack=0 Win=16384 Len=0

▶ Frame 3: 54 bytes on wire (432 bits), 54 bytes captured (432 bits)  
 ▶ Ethernet II, Src: 00:00:00\_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00\_00:00:02 (00:00:00:00:00:02)  
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 ▼ Transmission Control Protocol, Src Port: 3268 (3268), Dst Port: 7 (7), Seq: 0, Ack: 0, Len: 0

Source Port: 3268  
 Destination Port: 7  
 [Stream index: 0]  
 [TCP Segment Len: 0]  
 Sequence number: 0 (relative sequence number)  
 Acknowledgment number: 0 (relative ack number)  
 Header Length: 20 bytes  
 ▶ Flags: 0x010 (ACK)  
 Window size value: 16384  
 [Calculated window size: 16384]  
 [Window size scaling factor: -2 (no window scaling used)]  
 ▶ Checksum: 0x0000 [validation disabled]  
 Urgent pointer: 0

▼ [SEQ/ACK analysis]  
 ▼ [TCP Analysis Flags]  
 ▼ [Expert Info (Note/Sequence): TCP keep-alive segment]  
 [TCP keep-alive segment]  
 [Severity level: Note]  
 [Group: Sequence]

0000 00 00 00 00 00 02 00 00 00 00 00 01 08 00 45 00 .....E.  
 0010 00 28 00 00 00 00 40 06 7c ce 7f 00 00 01 7f 00 .(....@. |.....  
 0020 00 01 0c c4 00 07 00 00 00 00 00 00 00 50 10 .....P.  
 0030 40 00 00 00 00 00 @.....

המערכת של Wireshark מאפשרת להתכוון מה בעצם גרם לה לא לאהוב משהו, וניתן לראות כי מדובר בתוך ה-TCP, כאשר יש בעיה שהיא סוג של הערה למעשה לגבי מצב ה-SYN. מה שניתן לראות בניתוח, הוא שלמעשה התקבלה בקשה לגרום לפקטת ה-TCP להיות במצב של Keep-Alive.

איך זה מתבצע ברמה של TCP? ובכן, נשלחת בקשה עם דגל ACK בלבד עם sequence number קודם לבקשה זו. דבר אשר מבקש למעשה ליצור בקשת keep-alive ברמת ה-TCP.

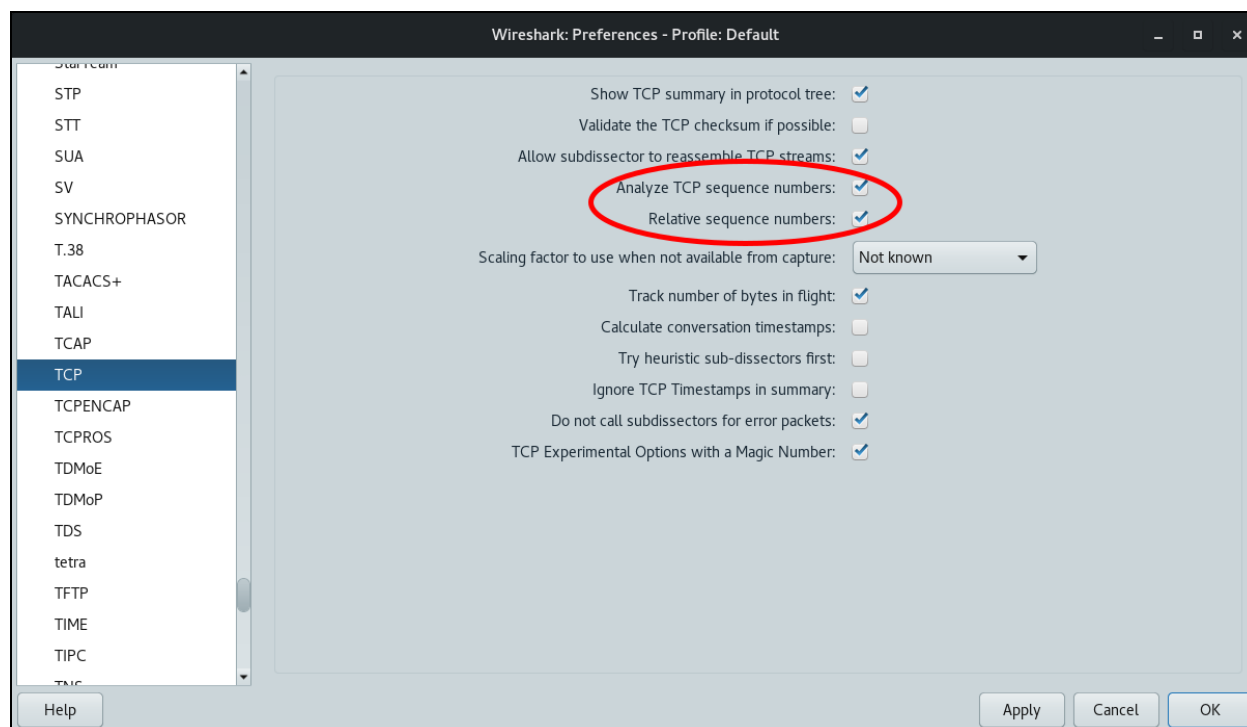
במידה ואין ב-Wireshark הנמצא ברשותכם את האפשרות לראות את נושא ה-analysis, ניתן לגשת לתפריט הבא:

Preferences -> Protocols -> TCP

עבודה עם Wireshark חלק א'

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)

שם יש את ה-checkbox הבא: Analyze TCP sequence numbers:



במקרה הזה, ש-Wireshark עוזר להבין בעיה של פקטה בודדת, זה קל יותר להבין מה קורה, וזאת כמובן, במידה וכמובן מבינים את הפרוטוקול וכיצד הוא עובד, אך לפעמים יש מצב בו יש צורך להבין דיאלוג שלם בשביל להבין את המידע ואולי אף להבין בעיה.

## ניתוח דיאלוג

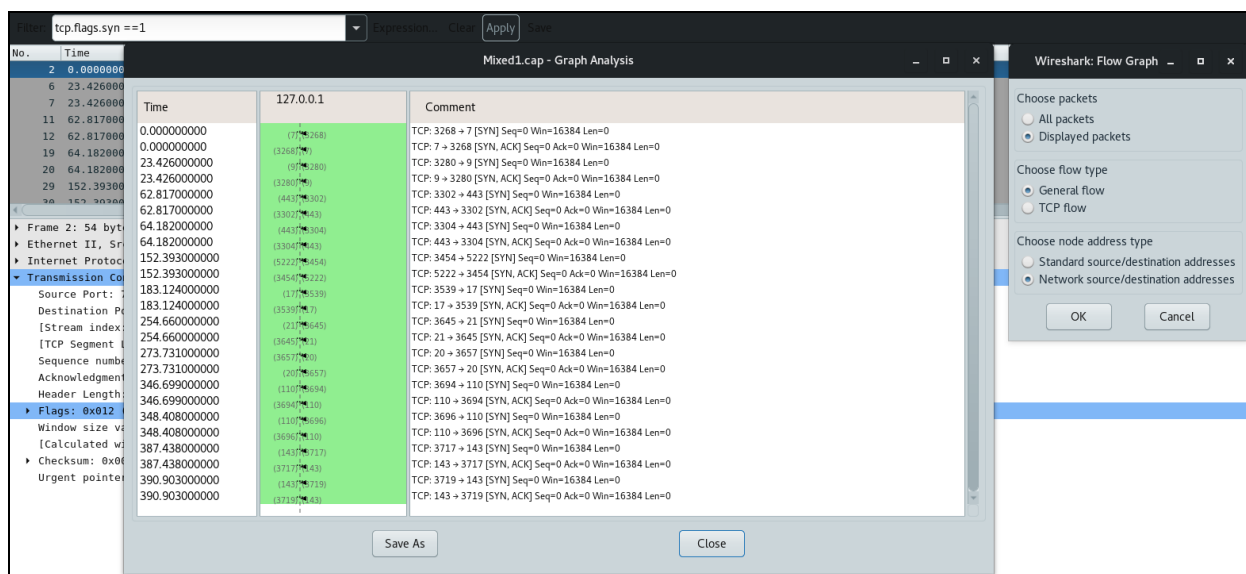
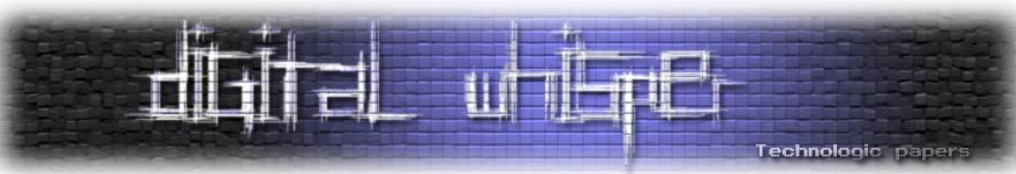
עד עכשיו הסברתי איך ניתן למצוא מידע, ולראות פקטה בודדת. לפעמים יש צורך ממש להבין מה קורה כתעבורה שלמה.

לשם כך, ישנם הרבה כלים ב-Wireshark, ובפרק זה אדבר על שניים מתוכם:

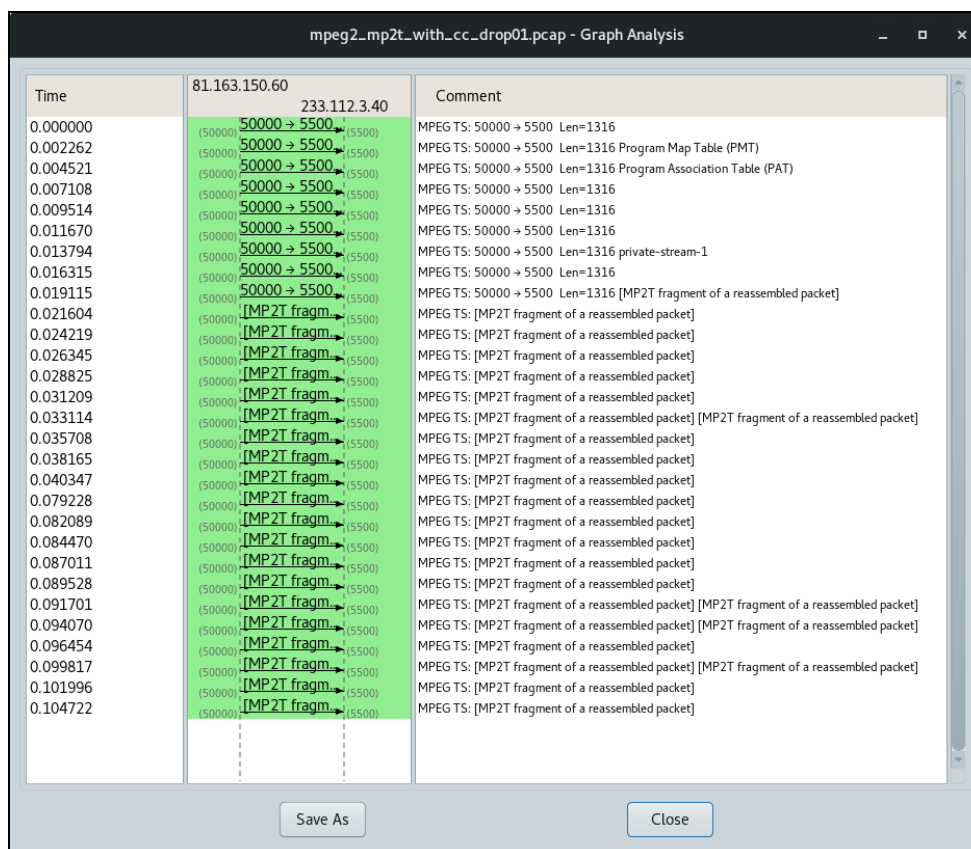
## :Flow Graph

כאשר יש צורך לראות איך המידע נשלח, ניתן להשתמש בכלי אשר נמצא תחת תפריט בשם:

Statistics -> Flow Chart



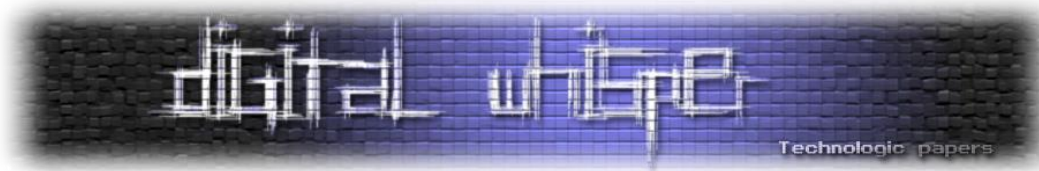
ניתן לצפות במספר דברים בגרף, בהתאם להגדרות בחלון הימני ביותר. בתמונה הזו, השתמשתי בסיוון אשר מביא את כל הפקטות מבוססות TCP עם פעולת SYN כלשהי. במידה והייתה תעבורה לעוד כתובת, היא הייתה מוצגת כך:



ניתן לראות את כתובת הבקשה 81.163.150.60 אשר מנסה לגשת לכתובת בידע של 233.112.3.40. במידה ושרת היעד היה עונה, היה ניתן לראות גם חץ חזרה אל כתובת המקור. לחיצה על כל שורה בגרף,

עבודה עם - Wireshark חלק א'

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)



מסמנת ברשימת הפקטות את הפקטה שעליה צופים, וכך ניתן למצוא טוב יותר פקטה סוררת כאשר ישנם עשרות פקטות לעבור בדיאלוג.

הכלי מסייע לראות לאן מידע מגיע ולאן מידע נשלח במשך זמן של דיאלוג. כאשר מדובר בפרוטוקול כדוגמת SIP, ממש ניתן לראות את כל הבקשות השונות, כולל יצירת דיאלוג, ואפילו תעבורת RTP, אך גם ב-HTTP ופרוטוקולים נוספים ניתן למצוא הרבה מידע מועיל בשמוש בגרף.

### מעקב אחר מידע:

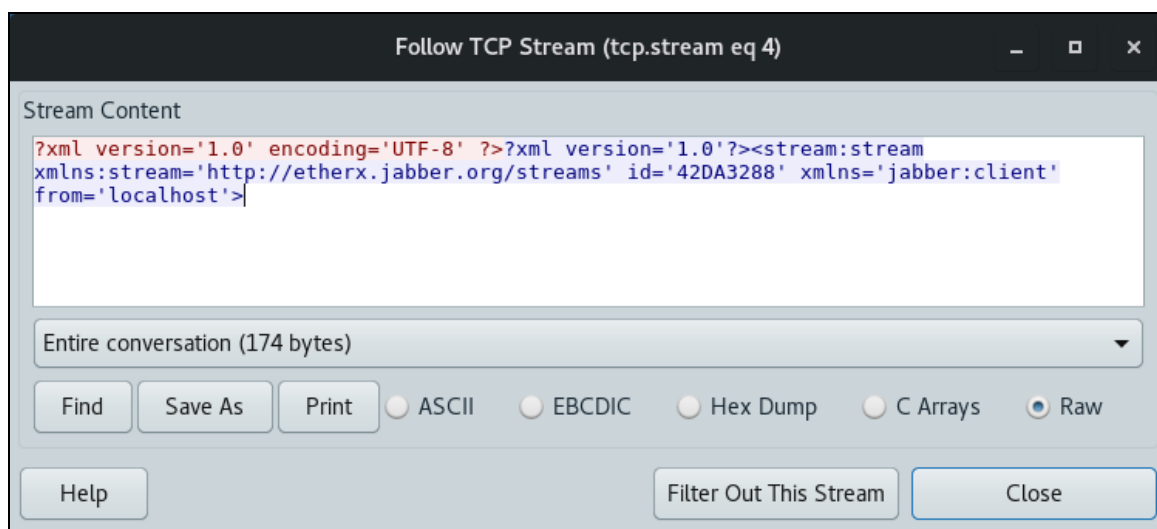
כלי נוסף ומאוד חשוב, שעוזר להבין הרבה מאוד ממצב התקשורת הוא כלי אשר נקרא Follow XXX Stream, כאשר XXX יכול לייצג אחד מתוך שלוש:

- Follow TCP Stream
- Follow SSL Stream
- Follow UDP Stream

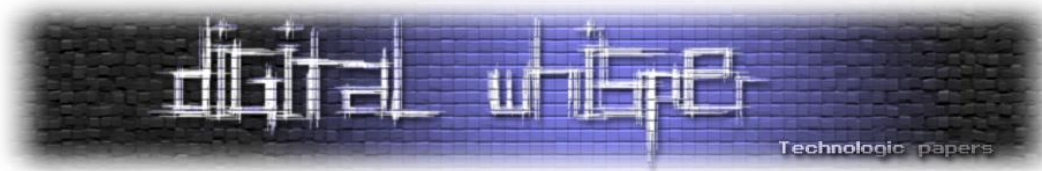
ניתן להגיע למצב זה בשני דרכים עיקריות:

- מקש ימני על פקטה מסוימת, ואז לבחור באופציה הרצויה של Follow XXX Stream.
- התפריט Analyze ושם לבחור את האופציה הרצויה של Follow XXX Stream

התוצאה תראה כך:



המסך אשר נפתח, מאפשר לצפות בכל מידע השייך ל-layer 7 (או נמוך יותר - תלוי בפרוטוקול) במגוון צורות, כאשר כאן בתמונה הוא במצב Raw, כלומר הבתים שנכנסו הם הבתים שרואים. בנוסף, מידע יוצא נצבע באדום על רקע ורוד, ומידע חוזר, נצבע בצבע כחול על רקע כחול.



ניתן לבצע חיפוש מחרוזות, ניתן לבחור רק צד מסוים בשיחה ועוד מספר פעולות. צפיה במצב של Hex Dump עזרה לי יותר מפעם או פעמיים לגלות מחסור בתווים לא מודפסים, כדוגמת UTF-8 או פרוטוקולים בינאריים.

בחלק הבא של המאמר, אסביר לעומק כיצד ניתן לצפות בתוכן כאשר הוא מוצפן. בנוסף, אדגים שימושים שונים בכלי לניתוח המידע.

## סיכום

בחלק זה הסברתי בקצרה יחסית, כיצד הממשק אשר נקרא Wireshark עובד. הסברתי כיצד ניתן לסנן מידע, והתחלתי להסביר על הכלים אשר מאפשרים לנתח בעיות שהם קצת מעבר לפקטה בודדת. המטרה של חלק זה, היה יותר להכניס לראש של Wireshark, בעוד שהחלק הבא, יתמקד בהתמודדות והבנה של כיצד מנתחים מידע, כולל עבודה עם מנהרות SSL/TLS.