**TCP over ICMP**

*סדנה בתקשורת מחשבים בהנחיית דני כלפון*

*קיץ 2022*

*יובל זראד 318290780*

*יובל נויהאוז 204900229*

**תוכן עניינים**

מבוא3

למידת עומק4

TCP PROTOCOL4

UDP PROTOCOL4

ICMP PROTOCOL5

הבעיה והפתרון7

ארכיטקטורה7

אתגרים7

הקוד7

מדריך למשתמש10

מסמך התקנה והפעלה10

טסטים10

**מבוא**

תיק פרויקט זה מציג את המימוש שלנו לפרויקט TCP over ICMP. מטרת הפרויקט היא לאפשר תקשורת בפרוטוקולTCP לרכיב מרוחק, במחשבים בהם תקשורת TCP חסומה (ע"י הfirewall או רכיב תקשורתי אחר). באמצעות מימוש זה, יוכל המשתמש לבצע תקשורת TCP על שימושיה השונים (גלישה בדפדפן, שימוש ב emailוכו') גם במחשבים אלה. הדרישה היחידה היא שתקשורת ICMP לא תהיה חסומה מהמחשב היוצא.

בחרנו בפרויקט זה כיוון שראינו בו אתגר מימושי ועיצובי, בעל שימוש ריאלי - היכולת לייצר תקשורת באופן לא טריוויאלי. על מנת לממש פרויקט מסוג זה עלינו יהיה להעמיק את הידע שלנו בפרוטוקולים TCP, ICMP וההדרים שלהם, ללמוד ולהתנסות בשימוש בחבילות פייתון המאפשרות עבודה עם socket-ים, עבודה אל מול כרטיס הרשת ויצירה של פקטות – כל הללו הינם אבני בניין חשובות בעת כתיבת תוכנות רשתיות, נושא אשר אנו מעוניינים לעסוק בו בעתיד.

**למידת עומק**

TCP Protocol

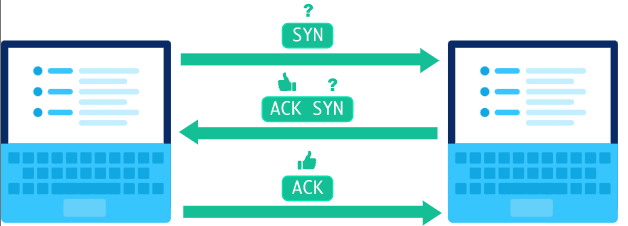
TCP הוא פרוטוקול תקשורת בשכבת התעבורה המבטיח העברה אמינה של נתונים בין שתי נקודות קצה. הפרוטוקול מתמודד עם בעיות נפוצות בתקשורת מבוססת פקטות, בין היתר פקטות אבודות, סדר פקטות, שכפול פקטות ופקטות משובשות.

תקשורת TCP מאופיינת בשלושה שלבים עיקריים.

שלב 1: הקמת הקשר

הקמת קשר מתבצעת בלחיצת יד הנקראת three-way handshake.

1. SYN – תחנת המקור שולחת הודעה לפתיחה הקשר.
2. ACK-SYN – תחנת היעד מקבלת את ההודעה ושולחת בתגובה הודעת אישור קבלה ופתיחת קשר מצידה.
3. ACK – תחנת המקור מיידעת את תחנת היעד על סיום מיסוד הקשר מצידה.



שלב 2: שליחת נתונים

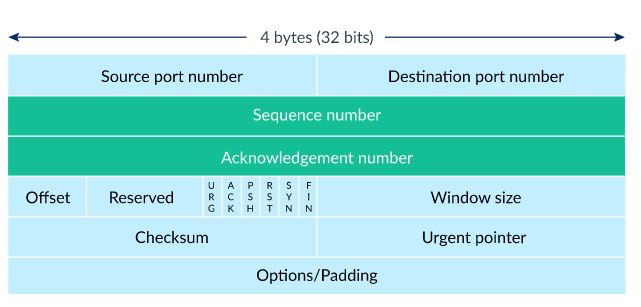
כאשר פקטה נשלחת על גבי TCP, תחנת היעד מיידעת את תחנת המקור בהודעת ACK על כך. בתחילה, תחנת המקור שולחת את המידע עם מספר סידורי של השליחה. לאחר מכן תחנת היעד שולחת הודעת ACK ומעלה את המספר הסידורי של הקבלה כאורך המידע שקיבלה. המספר הסידורי ומספר הACK הם חלק מההדרים של הפרוטוקול.

שלב 3: סגירת הקשר

כל אחת מהתחנות יכולה לסגור את החיבור כאשר הם לא מעוניינים בשליחת/קבלת מידע. סגירת הקשר, בדומה לפתיחה הקשר, מתבצעת ב3 שלבים.

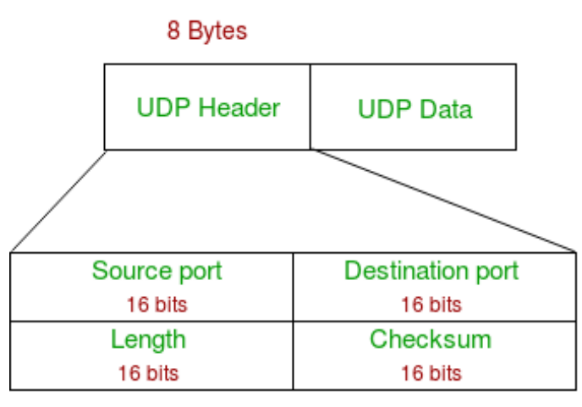
1. FIN – הודעה אשר נשלחת ע"י התחנה שמעוניינת לסגור את הקשר.
2. ACK FIN – תחנת היעד מיידעת את תחנת המקור שהיא קיבלה את ההודעה והיא סוגרת את הקשר.
3. ACK – תחנת המקור מיידעת על סגירת הקשר.
4. הדרים

* Source port number - פורט מקור.
* Destination port number – פורט יעד.
* Sequence number – מספר סידורי המזהה את המקטע בתוך זרם המידע הכולל.
* Acknowledgement number – מספר אשר נשלח ע"י תחנת היעד שמצביע על הבייט הבא שהיא מצפה לקבל.
* Reserved – מקטע ששמור עבור שימוש עתידי אופציונלי.
* Window size – גודל חלון השליחה של TCP, מציין כמה מידע השולח מוכן לקבל ברגע זה.
* Checksum – פונקציה המופעלת על המידע עצמו ולא רק הheader, משמש לווידא תקינות המידע.
* Pointer Urgent – מצביע למיקום המידע הדחוף בסגמנט, שהדגל URG מורה על קיומו.



UDP Protocol

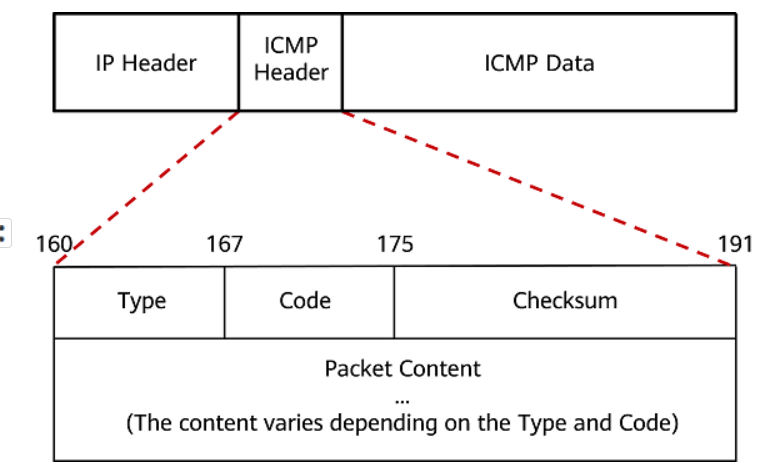
פרוטוקול UDP גם הוא פרוטוקול תקשורת בשכבת התעבורה. הפרוטוקול מאפשר העברת נתונים לא אמינה – הוא לא מספק שימור סדר והבטחה שכל הפקטות מגיעות, בשונה מTCP. בנוסף לכך, הוא פרוטוקול connectionless, כלומר לא מייסד קשר. ההדרים שלו צרים מאוד, והוא נראה כך:



ICMP Protocol

ICMP הוא פרוטוקול בשכבת הIP המשמש לתקשורת על שגיאות ועדכונים בין ראוטרים ומכשירים שונים ברשת. הודעות ICMP נשלחות בתרחישים שונים, למשל כאשר מכשיר שולח הודעה אשר גדולה מדי למכשיר היעד, מכשיר היעד יפיל את ההודעה וישלח הודעת ICMP בחזרה. שימוש נוסף הוא כאשר הnetwork gateway מוצא נתיב קצר יותר להודעה, הוא ישלח הודעת ICMP על מנת לבצע redirection לנתיב הקצר יותר. בשונה מפרוטוקולים בשכבות גבוהות יותר, הודעת ICMP מטופלת ע"י מודול הIP במחשב היעד ולא ע"י אפליקציה כלשהי.

הדרים

* Type - סוג ההודעה. למשל, הבייט 0 מסמן כי מדובר בתשובה הודעת ICMP, והבייט 8 מציין כי מדובר בבקשה.ישנם סוגים של הודעות ICMP:
* Destination Unreachable - לא ניתן היה להגיע אל כתובת היעד.
* Time Exceeded - הפקטה נזרקה מאחר ושדה ה TTL התאפס.
* Echo and Echo Reply - המידע שנשלח ליעד מוחזר כחלק מן התשובה
* Code – סוג של subtype עבור ה type הראשי. לדוגמה, בהודעה מסוג 3 ישנן 16 סוגי code שונים. Code 0 אומר שרשת היעד לא הייתה ניתנת להשגה ואילו code 1 פירושו הוא שה host אינו ניתן להשגה.
* Checksum - מורכב משני בתים ומשמש לבדיקות תקינות של ההודעה.
* Content - גוף ההודעה, משתנה בין סוגי ההודעות השונים.

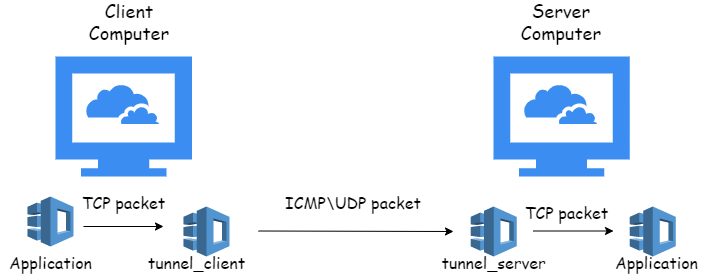
**הבעיה והפתרון**

ארכיטקטורה

מטרתנו היא לאפשר תקשורת בפרוטוקול TCP במצבים בהם פרוטוקול זה חסום ברשת, כפי שהוצג במבוא, באופן הבא:

1. תוכנת הלקוח רצה על מחשב הלקוח ומאזינה בפורט מסויים באמצעות raw socket.
2. במחשב הלקוח אפליקציה שולחת פקטת TCP ל loopback (127.0.0.1) בפורט שתוכנת הלקוח מאזינה עליו.
3. תוכנת הלקוח מורידה מהפקטה את שכבת הTCP, כולל ההדרים, ועוטפת אותה עם headerים של הפרוטוקול המאופשר ברשת בו משתמשים לביצוע הtunnel (UDP\ICMP).
4. תוכנת השרת רצה על מחשב השרת ומאזינה בפורט מסויים. היא מקבלת את הפקטה, ובונה פקטת IP הנשלחת לloopback על השרת עם ה payload שהתקבל מהלקוח.

באופן זה הלקוח והשרת מנהלים בפועל תקשורת TCP תקינה לחלוטין. פקטות הTCP עוברות בשלמותן והתוכנה לא עורכת את התוכן – רק את ההדרים על מנת לאפשר ניתוב מוצלח.

תיאור סכמטי של התוכנה:

אתגרים

* IPTables - כאשר משתמשים ב raw sockets, יש למצוא דרך למנוע ממערכת ההפעלה לטפל בעצמה בפקטה שהתקבלה. מאחר והתוכנה לא מבצעת socket bind על פורט מסויים בכרטיס הרשת, כאשר תכנס הודעת TCP, מערכת ההפעלה תענה ב RST-TCP כיוון שאף תוכנה לא מאזינה על הפורט המסויים, והחיבור ייכשל. על מנת לפתור בעיה זו השתמשנו בכלי הנקרא iptables. כלי זה קיים במערכות לינוקס ומאפשר למשתמש להגדיר חוקים המגדירים למערכת ההפעלה כיצד לטפל בתעבורת רשת. תחת INPUT נגדיר איזו תעבורה המערכת יכולה לקבל, ותחת OUTPUT איזו תעבורה היא יכולה להוציא. נתבסס על כך שקבלה או שליחה של פקטות בעזרת raw sockets אינם מוגבלים תחת חוקים אלה.

בתחילת ריצת התוכנית, צד שרת ולקוח, היא מגדירה חוק iptables אשר יגרום למערכת ההפעלה להתעלם מפקטות TCP נכנסות השייכות לפורט בו מקונפגת התוכנה להאזין. בסיום הריצה ימחק החוק, על מנת לא להשאיר עקבות.

* NAT – אנו מניחים כי צד הלקוח רץ על מחשב המחובר לאינטרנט באמצעות NAT. NAT הוא רכיב המסוגל לתרגם עשרות ומאות של כתובות פרטיות אל כתובת ציבורית אחת, מה שתורם לחיסכון גדול ברשתות מקומיות. בפועל, הוא יגרום לכך שכתובת הIP שהשרת יראה היא לא הכתובת בפועל של הלקוח אלא כתובת הNAT. כאשר התוכנה מקונפגת למצב UDP, הNAT יוסיף את צמד הפורטים לחוקים שלו ויאפשר את התעבורה ללא הפרעה. לעומת זאת, כאשר התוכנה מקונפגת לרוץ במצב ICMP אשר לא עושה שימוש בפורטים אלא בIDים, חוקי הNAT עובדים בצורה שונה. הNAT משתמש בIDים של ההודעות על מנת לאפשר קבלה וניתוב של בקשות ICMP. האתגר הוא שהמון NATים ימחקו את החוק מיד עם קבלת הודעות הICMP, מה שיגרום לכך שהשרת ינסה לענות בעצמו על בקשת הICMP, במקביל להודעה שנשלחת משרת הטאנל, ורק אחת מהתשובות תגיע ללקוח – מה שישבש את התקשורת על גבי ה-tunnel. הפתרון לבעיה זו היא שימוש בכלי iptables. כאשר צד השרת יקבל הודעת echo-request המיועדת אליו, לפי ה-ID שהוגדר מראש כפרמטרים לתוכנה, היא תוסיף חוק iptables המונע יציאה של תעבורת ICMP אל הכתובת ממנה הגיעה הבקשה, ובסיום ריצת התוכנית תמחק חוק זה.

הקוד

הקובץ tunnel\_client.py:

* המתודה main מקבלת את הארגומנטים הבאים בשורת הפקודה:
  + s – כתובת ip של השרת אליו רוצים להתחבר.
  + t – פורט (מעליו תעבור תעבורת ה-TCP) שרוצים להתחבר אליו בצד השרת.
  + u – דגל שמסמן האם לרוץ במצב UDP.
  + p – פורט שמשמש כאשר הtunnel מתבצע מעל פרוטוקול UDP.
  + I – דגל שמסמן לרוץ במצב ICMP.
  + d – ID של בקשות הICMP, כאשר הtunnel מתבצע מעל פרוטוקול UDP.
  + mtu – גודל פקטה מקסימלי, רלוונטי כאשר תנאי הרשת דורשים הקטנה של ערך זה.

אחראית לייצור שני socketים (במתודה create\_sockets) - socket לקוח על ה loopback ו-socket לקוח מעל UDP\ICMP. מאתחלת את הלולאה האינסופית של הלקוח (במתודה client\_main\_loop), בה מאזינים על הsocketים ומעבירים מידע לפי אופן פעולתם.

הקובץ tunnel\_server.py:

* המתודה main מקבלת את הארגומנטים הבאים בשורת הפקודה, באופן סימטרי לתוכנת הלקוח:
  + t – פורט להאזנה.
  + u – דגל שמסמן האם לרוץ במצב udp.
  + p – פורט שמשמש כאשר רצים במצב UDP.
  + I – דגל שמסמן לרוץ במצב ICMP.
  + d – ID של בקשות הICMP.
  + mtu – גודל פקטה מקסימלי.

אחראית לייצור שני socketים (במתודה create\_sockets) - socket שרת על ה loopback ו-socket שרת מעל UDP\ICMP. מאתחלת את הלולאה האינסופית של השרת (במתודה server\_main\_loop), בה מאזינים על הsocketים ומעבירים מידע לפי אופן פעולתם.

הקובץ sockets.py:

מייצא את מחלקות ה-socket של השרת והלקוח.

* המחלקה socket – מחלקה בסיסית המכילה חוקי iptables ומוחקת אותם בסיום השימוש ב-socket.
* המחלקה ICMPTunnelSocket – יורשת מ-socket. מגדירה כיצד התוכנה מקבלת ושולחת פקטות דרך ה-tunnel כאשר התוכנה רצה במצב ICMP. מסוגלת לשלוח פקטות ב-tunnel ע"י בניית פקטת ICMP כך שה-data הוא ה-payload ה-tcp-י.

במצב לקוח – המחלקה הרלוונטית היא ICMPTunnelClientSocket היורשת ממנה. הפקטות שיישלחו על ידי המחלקה הן פקטות echo-request. במצב שרת, המחלקה הרלוונטית היא ICMPTunnerlServer, והפקטות שיישלחו על ידי המחלקה הן מסוג echo-reply. כאשר מתקבלת פקטת ICMP עם ה-ID שהוגדר למחלקה, מוגדר חוק iptables המונע יציאת פקטות בפרוטוקול ICMP ממערכת ההפעלה של השרת אל כתובת הלקוח (מתמודד עם אתגר הnat).

* המחלקה UDPTunnelSocket – יורשת מ-socket, מגדירה כיצד התוכנה מקבלת ושולחת פקטות דרך ה-tunnel כאשר התוכנה רצה במצב UDP. במצב שרת, המחלקה הרלוונטית היא UDPTunnelServer, היא אחראית להאזנה (bind) על הפורט הייעודי שהוגדר בשורת ההרצה. מחלקה זן אחראית לקילוף פקטות מה-payload שלהן.
* המחלקה LoopbackSocket – מגדירה כיצד התוכנה מקבלת ושולחת פקטות על הכרטיס ה-loopback. נעשה שימוש באובייקט L3RawSocket של המודול scapy המאפשר קריאה-כתיבה מעל כרטיס זה. במצב לקוח (LoopbackClientSocket) התוכנה תקבל תעבורה המיועד לloopback של המחשב על הפורט הייעודי שהוגדר בשורת הריצה של התוכנה. במצב שרת (LoopbackServerSocket) התוכנה תקבל תעבורה המיועדת לכתובת ה loopback של המחשב שיצאה מהפורט הייעודי שהוגדר בשורת ההרצה של התוכנה, בעצם פקטות תשובה מהשרת אל הלקוח. אחראית לקילוף פקטות משכבת ה-TCP ושליחתן.

הקובץ iptables.py:

אחראי להתממשקות עם הכלי iptables.

* המחלקה IPTablesRule, מייצגת חוק iptables, ומאפשרת את הכנסתו ומחיקתו.
* המחלקה IPTablesLoopbackRule – יורשת מIPTablesRule. מגדירה חוקים המונעים קבלה של פקטות TCP לא רצויות על כרטיס ה-loopback של המחשב.
* המחלקה IPTablesICMPRule – יורשת מIPTablesRule. מגדירה חוקים המונעים יציאה של פקטות ICMP לא רצויות על כרטיס ה-loopback של המחשב. תהיה בשימוש רק כאשר התוכנה מקונפגת לריצה במצב ICMP.

**מדריך למשתמש**

מסמך התקנה והפעלה

התוכנה מתאימה למערכות לינוקס ונבדקה על Ubuntu server 20.04.   
לשם הרצתה, נדרש פייתון מגרסה 3. יש להתקין את המודולים הפייתונים באופן הבא:

* Sudo apt install python3-pip
* Sudo pip3 install scapy
* Sudo pip3 install python-iptables

הרצת השרת:

השרת יופעל ע"י הרצת הקובץ tunnerl\_server.py, יש למלא את הארגומנטים המסמלים את כתובת הלקוח והפורט והאם התוכנה רצה במצב UDP\ICMP כפי שמתואר בפרק מבנה הקוד.

דוגמה לשורת הרצה במצב UDP:

Sudo python3 tunnel\_server.py -t 15 -u -p 100 –mtu 1000

דוגמה לשורת הרצה במצב ICMP:

Sudo python3 tunnel\_server.py -t 15 -i -d 100 –mtu 1000

הלקוח יופעל ע"י הרצת הקובץ tunnel\_client.py, באופן דומה לשרת, יש למלא את הארגומנטים המסמלים את כתובת השרת והאם התוכנה רצה במצב ICMP\UDP כפי שמתואר בפרק מבנה הקוד.

דוגמה לשורת הרצה במצב UDP:

Sudo python3 tunnel\_client.py -s <server\_ip> -t 15 -u -p 100 –mtu 1000

דוגמה לשורת הרצה במצב ICMP:

Sudo python3 tunnel\_client.py -s <server\_ip> -t 15 -i -d 100 –mtu 1000