**מחקר חולשה בפרוטוקול QUIC**

Based on the paper: 'QUIC Transport Protocol: Design and Internet-Scale Deployment'

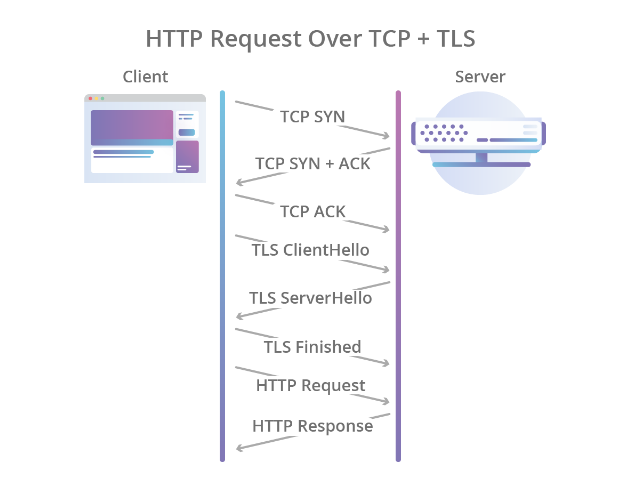
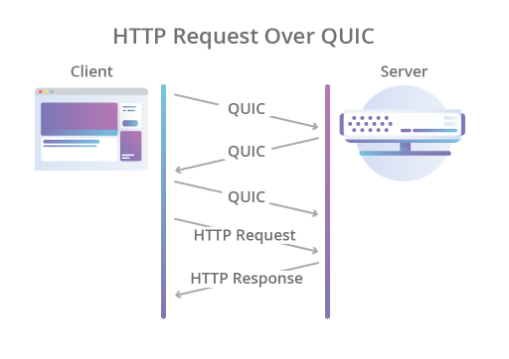
**יאיר בן חמו**, 204090930. **דניאל כהן**, 305316408.

הקדמה:

כיום, כשאנו מקישים את כתובת האינטרנט אליה אנו מעוניינים להגיע, דפדפן האינטרנט משתמש בפרוטוקול התקשורת הנפוץ בעולם, TCP, על מנת לאפשר לנו גישה לאתר. שרתיה של גוגל, המניעים את מנוע החיפוש, מתמודדים עם התעבורה מבין העמוסות בעולם, ולשם התמודדות עם עומס זה הכריזה החברה על פרוטוקול תקשורת חדש המבוסס על הפרוטוקול הקיים UDP ונקרא QUIC (ר”ת Quick UDP Internet Connection).

**פרוטוקול ה-QUIC**, כמשתמע משמו, נועד לאפשר חיבור מאובטח ומהיר בין המשתמש לבין האתר אליו מעוניין להגיע. גוגל פיתחה את QUIC כך שבמידה והמשתמש יצר קשר מאובטח מול השרת בעבר, שרת האתר יוכל לשלוח את המידע גם ללא בדיקת האתר והמתנה להקמת חיבור מאובטח.  
מבנה הפרוטוקול ומימושו מפורט בגוף המאמר "QUIC Transport Protocol: Design and Internet-Scale Deployment" עליו התבססנו במחקר הנוכחי.

להלן המחשה המתרחשת ללא פרוטוקול QUIC ולאחריה המחשה המציגה את השימוש בפרוטוקול:



בגוף המאמר מוזכרת הערה (הערה מספר 5) שבה נאמרו הדברים הבאים:

“QUIC connections are susceptible to off-path third-party reset packets with spoofed source addresses that terminate the connection. IETF work will address this issue.”

כלומר, פרוטוקול QUIC הוא פגיע לפקטות איתחול מזוייפות הנשלחות על ידי גורם חיצוני צד שלישי שגרמות לניתוק הקשר בין צד לקוח לצד שרת.

לטענת כותבי המאמר, וממציאי הפרוטוקול, האירגון IETF, האמון על סטנדרטיזצייה של פרוטוקולים, צריך לטפל בפגיעות זו על מנת שפרוטוקול QUIC ידע להתמודד עם מצב כזה.

במחקר הנוכחי נבדוק כיצד מתמודד פרוטוקול QUIC עם מקרה שבו off-path third-party מנסה לשלוח  reset packets מזוייפות והאם כתוצאה מכך נחבל הקשר בין השרת ללקוח.

במהלך המחקר נחקה קשר בין לקוח לשרת וניצור הודעות reset מגורם צד שלישי על מנת לחבל בקשר הקיים. נסתכל האם וכיצד IETF התמודדו עם בעיה זו ומהי התגובה שנקבל כתוצאה מחיקוי זה.

עבודת קדם בנושא המחקר:

פרוטוקול QUIC פותח בשנת 2013. על אף היותו יחסית חדש, פותחו עבור הפרוטוקול מימושים רבים בשפות שונות ומגוונות – Python, Java, C++, GO וכו'

לרשימת המימושים עיין בתת פרק ביבליוגרפיה הערה מספר [2] .

המחקר שלנו מבוסס על המימוש בשפה GO של הפרוטוקול [3].

בחרנו במימוש זה מהסיבות הבאות:

1. מימוש זה נראה הכי פופולרי מבין המימושים הקודמים (מעל 2500 stars ו300 forks).
2. מימוש זה מאפשר שליחת reset packets לעומת רוב המימושים.
3. שפה GO היא שפת פיתוח שפותחה בגוגל, המייסדים של QUIC, ולכן בחרנו לשמור על רצף לטובת המחקר.

כמו כן, המחקר מבוסס על תיעוד מפורט שנכתב על ידי ארגון IETF (Internet Engineering Task Force). ארגון זה הוא הגוף הממליץ על תקנים שמשמשים להפעלת האינטרנט. תקנים אלו מתפרסמים במסמכי RFC (קיצור של Request For Comments).

ה-IETF יצר את התקנים הבסיסיים ביותר באינטרנט, שמאפשרים חיבור מחשבים אחד לשני (TCP/IP, PPP, DNS ועוד) וכן תקנים שמאפשרים שירותים שונים כגון דואר אלקטרוני (MIME, POP3, IMAP4, SMTP), צפייה במדיה חיה (RTP), הורדת קבצים ומולטימדיה (FTP, HTTP) ועוד.

במחקר נשענו על תיעוד IETF על פרוטוקול QUIC להבנת התנהגות הפרוטוקול ולדרך התמודדתו עם שאלת המחקר [4].

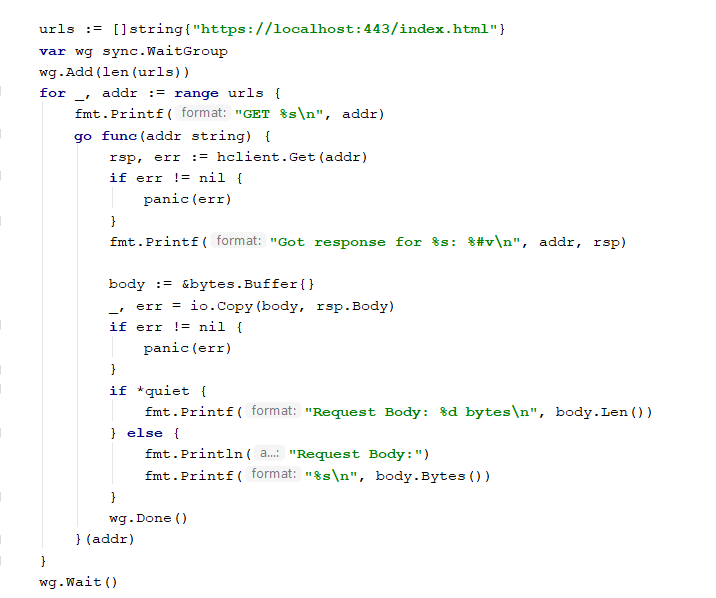
פרטים טכניים:

במחקר פיתחנו צד שרת וצד לקוח המדברים בפרוטוקול QUIC הממומש בשפת GO.

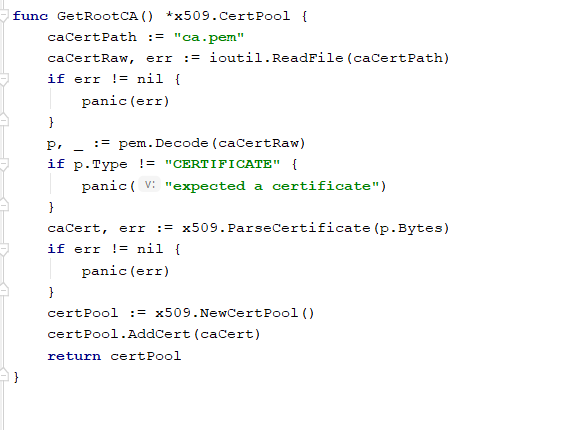
הIDE לפיתוח צד השרת והלקוח היה GoLand.

בתחילת המחקר ניסינו להריץ את הצדדים מעל מערכת ההפעלה Linux אך משראינו שלא ניתן לנתר אחר reset packets בwireshark מעל Linux עברנו למימוש צד שרת וצד לקוח מעל Windows.

להלן מתודת הקמת הקשר בין הלקוח לשרת:



כמו כן, להלן הקוד שדואג לקשר מאובטח בין שני הצדדים:

אבטחת הקשר תורחב בגוף המחקר ותוסבר מדוע הינה שלב מהותי בהתמודדות הפרוטוקול למול שאלת המחקר.

להלן ההסכם בין צד לקוח לשרת לשליחת reset packets:



כפי שניתן לראות, הלקוח קובע עם השרת שברגע שישלח האות "A" הוא צריך לצפות לreset packet לאחריו.

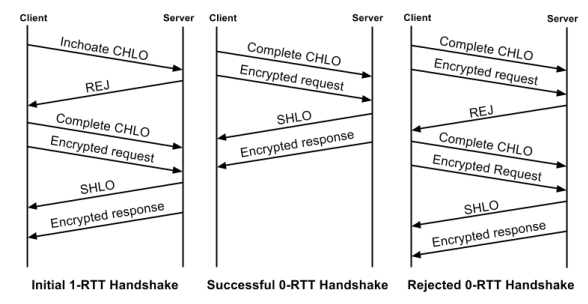
את כלל הפרוייקט העלנו לrepository בgithub [6].

כפי שנראה בהמשך, את כלל התקשורת בין צד הלקוח לשרת ניתרנו בעזרת Wireshark ומניתור זה נצליח להגיע בהמשך למסקנות בנוגע לשאלת המחקר.

גוף המחקר:

הקמת הConnection בין הלקוח לשרת מורכבת מהודעות שאינן מוצפנות (CHLO/SHLO - Client hello/Server hello) ומבקשות מוצפנות שבאות ביניהן.

להלן הדרך שבה מוקם הקשר בין הלקוח לשרת:



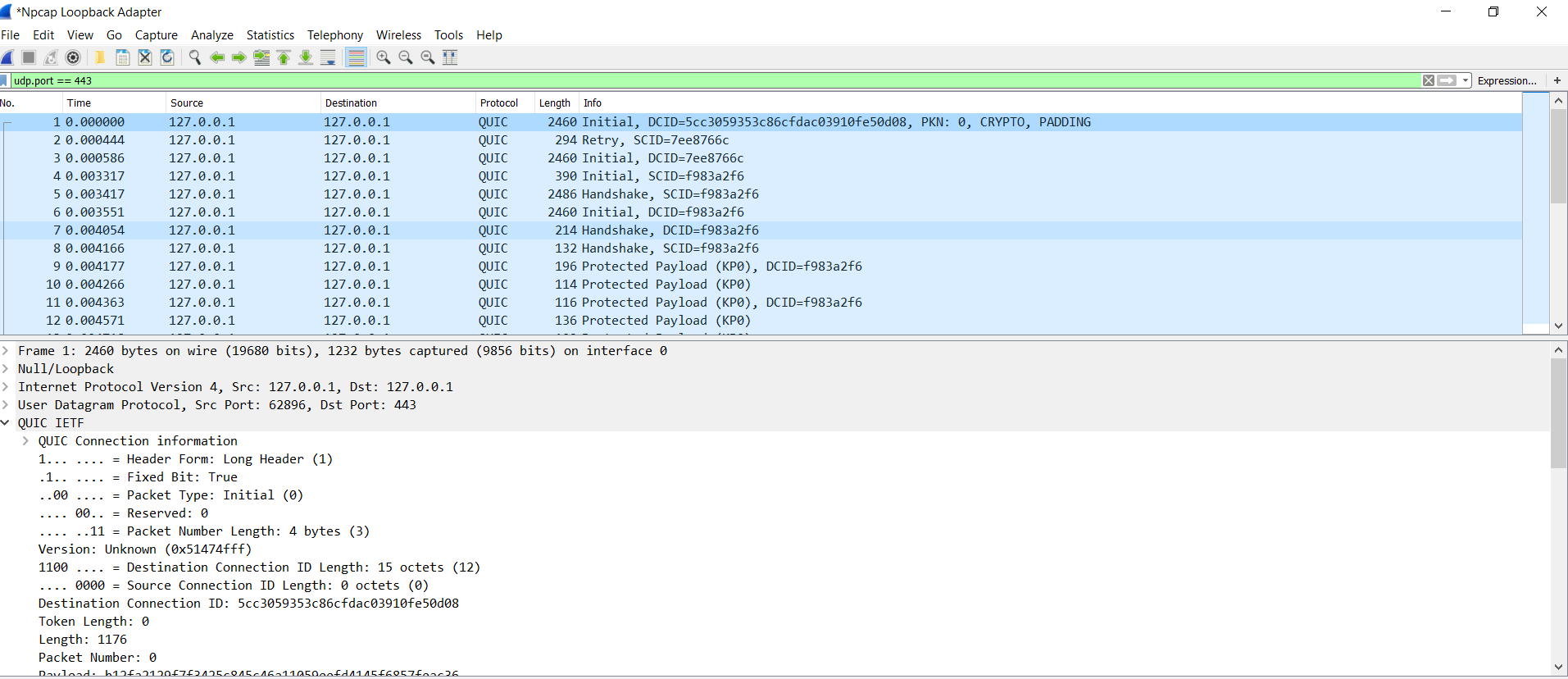
הפרוטוקול נשען על כך שבזמן הקמת הקשר, הקשר בין הלקוח והשרת מוקם כמוצפן ובשלב זה מוחלף ביניהם (בעזרת המפתחות הציבוריים שלהם) token שמשמש אותם להצפנת המשך הקשר. QUIC מייצר stream אמין לביצוע handshake קריפטוגרפי בין הלקוח לשרת.

ניתן לקרוא בהרחבה בתיעוד הפרוטוקול על כל אחד משלבי התקשורת בהקמת הקשר, ראינו לנכון שלא לפרט על שלבים אלו מכיוון שאינם עיקר המחקר שלנו.

לאחר שלב זה נשאלת השאלה, כיצד יתמודד הפרוטוקול עם reset packets המגיעות מגורם צד שלישי.

לצורך כך הקמנו צד שרת ולקוח המשתמשים בפרוטוקול.

להלן הקמת הקשר כפי שהיא נצפתה בwireshark:



ניתן לראות כי הקמת הקשר זהה למוגדר במאמר הרשמי של פרוטוקול QUIC.   
כמו כן, ניתן לראות כי שניהם מעבירים מידע בעזרת המפתחות הפרטיים שלהם.

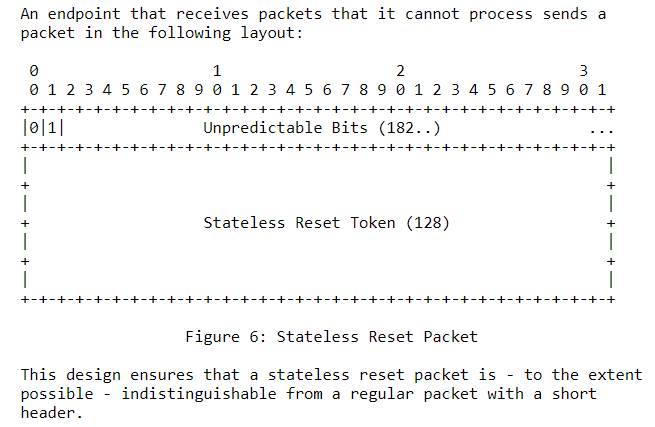
כעת החלטנו בגוף התקשורת בין הלקוח לשרת, להגדיר symbol ("A") שיכריז על reset packet, להלן הגדרת הsymbol בצד לקוח:



על ידי הגדרת symbol זה בשלב הקמת הקשר, IETF בעצם פתרו את הבעיה של גורם צד שלישי. שכן צד שלישי אינו יודע מה הsymbol שעליו לשלוח על מנת לחקות reset packets. בעצם לעולם לא יצליח צד שלישי לחבל בקשר בין הלקוח לשרת.

כמו כן, ניתן לראות שכאשר הלקוח שולח לצד שרת reset packets היא מוצפנת ואינה ניתנת לחיקוי. כלומר צד שלישי לא ידע מהי הreset packets ולכן לא יוכל לשלוח אותה לשרת/ללקוח.

לאחר שהרמנו בעצמנו ושמנו לב לכך שזה הנוהל שבוצע בפרוטוקול זה, מצאנו לכך אסמכתא במסמך תיעוד שכתב ארגון IETF לאחר הסטנדרטיזציה שעשה לפרוטוקול, להלן דבריו:



מסקנות המחקר:

במחקר הנ"ל יצרנו צד תקשורת בין לקוח לשרת בפרוטוקול QUIC, תוך חיקוי צד שלישי המנסה לשלוח reset packets.

במהלך המחקר למדנו כי לאחר העבודה שארגון IETF ביצעו על הפרוטוקול והפיכתו למוגן, הם הצליחו לטפל גם בחולשה המוזכרת בהקמת הפרוטוקול.

כלומר, IETF דאגו שגורם צד שלישי לא יצליח לחקות reset packets בין הלקוח לשרת ולחבל בקשר ביניהם.

זאת לאור העובדה, שreset packets יתקבלו על ידי אחד הצדדים רק לאחר שליחת symbol שתואם ביניהם בתחילת הקשר. כמו כן, reset packets יראו בדיוק כמו פקטה רגילה ולכן לא יוכל צד שלישי לדעת איזו פקטה זו ועל ידי כך להעתיק ולחבל בקשר בין הלקוח לשרת.

**לאור המחקר שביצענו, ניתן להראות כי החולשה שהייתה עלולה לפגוע באבטחתו ובאמינותו של פרוטוקול QUIC, אכן טופלה על ידי ארגון IETF.**

ביבליוגרפיה:

[1] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, Zhongyi Shi, **The QUIC Transport Protocol: Design and Internet-Scale Deployment** Sigcomm 2017

[2] <https://github.com/quicwg/base-drafts/wiki/Implementations>

[3] <https://github.com/lucas-clemente/quic-go>

[4] <https://tools.ietf.org/html/draft-ietf-quic-transport-06>

[5] https://datatracker.ietf.org/meeting/interim-2017-quic-02/materials/slides-interim-2017-quic-02-sessa-quic-stateless-reset

[6] https://github.com/sdanielco/QUIC-Project/tree/master