

קישורים להורדה נמצאים בתוך ה README וכן גם הסבר איפה לשים כל קובץ כחלק מההכנה לריצה

המבנים והפורמטים של הקבצים והדאטה סט מצויינים ב README

חלק ראשון

1. הגורמים האפשריים לאיטיות העברת קבצים הם:

- רוחב פס נמוך : רוחב פס מייצג כמה נתונים ניתן להעביר בפרק זמן מסוים. במידה ורוחב הפס נמוך ההעברה תהיה איטית מכיוון שלא ניתן להעביר הרבה נתונים בזמן קצר. רוחב פס נמוך נגרם ממספר סיבות, לדוגמא, מגבלה של ספק האינטרנט, כאשר כמה מכשירים מחוברים באותו זמן, חיבור חלש וכו'...
- עיכובים ברשת : מרחק פיזי, עומס על הרשת ופרוטוקולים איטיים (כגון TCP) גורמים בין היתר לעיכובים ברשת.
- אובדן חבילות : בעיות ברשת, הפרעות רשת או חיבור אינטרנט לא יציב, יכולים לגרום למצב של איבוד חבילות ואי הגעתן ליעד.
- בעיות ברשת : בעיות כלליות שיכולות לנבוע מבעיות חומרה.

איך היינו מנתחים כל בעיה?

- בדיקת רוחב פס – ננסה להעביר קובץ קטן ונבדוק מהי המהירות המתקבלת בפועל, אם המהירות קטנה מאוד ביחס למהירות רוחב הפס המצופה אז ייתכן והתקלה ברוחב הפס.
- בדיקת עיכובים- שימוש בפקודת ping כדי לבדוק זמן תגובה, תוצאה עם זמן תגובה גבוה או פערים משמעותיים בזמנים מעידה על עיכובים ברשת.
- בדיקת אובדן חבילות – להשתמש בוואירשרק כדי לבדוק אובדן חבילות, לאחר לכידת תעבורה הוא מציג את כל החבילות שנלכדו ואז ניתן לחפש הודעות שגיהא המצביעות על חבילות שאבדו או לבדוק אם יש חור ברצף המספרים של ה "sequence numbers".
- אם אין הודעות שגיהא על חבילות שאבדו, אפשר לבדוק ב-Wireshark אם יש TCP Retransmissions, כלומר חבילות שנשלחו מחדש כי הן לא התקבלו. אם הפינג מראה זמן תגובה גבוה אבל אין אובדן חבילות, ייתכן שהבעיה היא עומס ברשת או רוחב פס נמוך.
- בעיות ברשת- בדיקת עומס על השרת (דרך מנהל משימות וכו'...)

2. ההשפעה של מנגנון בקרת הזרימה של TCP היא שהוא מוודא שהשולח לא ישלח יותר נתונים ממה שהמקבל יהיה מסוגל לעבד. מה שמאפשר זאת זה באמצעות חלון בקרת הזרימה שמוגדר על ידי המקבל. כאשר למעביר יש כוח עיבוד גבוה משמעותית מהמקבל אז השולח עשוי לשלוח נתונים בקצב גבוה יותר ממה שהמקבל מסוגל לעבד וזה יגרום להקטנת חלון בקרת הזרימה. ובסופו של דבר השולח יאלץ לחכות עד שהמקבל יוכל לעבד את הנתונים שנשלחו מה שיגרום להאטה בקצב ההעברה הכולל.
- אם רוצים לראות איך בקרת הזרימה משפיעה בפועל, אפשר לבדוק האם גודל החלון משתנה ב-Wireshark אם החלון מצטמצם מאוד אחרי מספר קטן של חבילות, זה סימן שהמקבל מתקשה לעבד את הנתונים, מה שגורם להאטה בקצב ההעברה.

3. תפקיד הניתוב ברשת הוא תהליך בחירת הנתוב שדרכו יעברו הנתונים מהרשת המקורית ליעד. בחירת הנתוב משפיעה על ביצועי הרשת בכך שלדוגמא בחירת נתוב קצר עשוי להקטין את זמן התגובה ולהגדיל את קצב ההעברה, בנוסף, נתוב עם פחות עומס עשוי להקטין את אובדן החבילות ולהגדיל את אמינות ההעברה.
- גורמים שיש לקחת בחשבון בהחלטות ניתוב:
- אורך הנתוב: מספר הקפיצות (hops) בין המקור ליעד.
 - עומס על הנתוב: כמות התעבורה שעוברת בנתוב.
 - אמינות הנתוב: שיעור אובדן החבילות בנתוב.
 - זמן תגובה: זמן התגובה הכולל של הנתוב.

אם רוצים לבדוק איזה חלק מהמסלול גורם להשהיות, אפשר להריץ פקודה שמראה את כל התחנות בדרך לשרת ולראות היכן מתרחשים עיכובים משמעותיים. אם אחת התחנות מציגה זמן תגובה גבוה מאוד, זה יכול להצביע על עומס באותו נתב.

4. MPTCP מאפשר שימוש במספר נתובים להעברת נתונים בין המקור ליעד.
- זה משפר את ביצועי הרשת על ידי ניצול טוב יותר של רוחב הפס הזמין והקטנת העומס על נתוב יחיד. הוא גם מגביר את האמינות על ידי מתן אפשרות להעברת נתונים בנתוב חלופי במקרה של כשל בנתוב הראשי.

השימוש ב-MPTCP שימושי במיוחד כשיש כמה חיבורים זמינים, למשל, כאשר מחשב מחובר גם לוויפי וגם לרשת סלולרית. זה מאפשר ניצול טוב יותר של רוחב הפס ומונע מצב שבו תעבורה מנותבת רק לנתיב אחד שיכול להיות עמוס.

5. סיבות אפשריות לאובדן חבילות:
 עומס יתר על הנתבים- נתבים עמוסים עשויים לאבד חבילות.
 בעיות חומרה- תקלות בצידוד הרשת.
 בעיות תוכנה- תקלות בתוכנת הניתוב.
 הגדרות שגויות- הגדרות ניתוב או QoS שגויות.
 צעדים לפתרון הבעיה:
 בדיקת עומס על הנתבים- צריך לנתר את עומס העבודה על הנתבים ואז להקטין אותו במידת הצורך.
 בדיקת חומרה- צריך לבדוק את ציוד הרשת ולהחליף רכיבים פגומים.
 עדכון תוכנה : לעדכן את תוכנת הניתוב לגרסה האחרונה.
 בדיקת הגדרות : לבדוק את הגדרות הניתוב ו-QoS-ולתקן שגיאות.

חלק שני

Early Traffic Classification With Encrypted ClientHello: A Multi-Country Study

תרומת המאמר:

המאמר מציג תרומה משמעותית בתחום סיווג התעבורה המוצפנת, במיוחד בתרחיש שבו נעשה שימוש בפרוטוקול Encrypted Client Hello (ECH). המחקר מתמקד בפיתוח אלגוריתם חדש הנקרא hybrid Random Forest Traffic Classifier (hRFTC) - מטרותו לשפר את היכולת לסווג תעבורה כבר בשלבים מוקדמים, תוך התמודדות עם האתגרים שמציב פרוטוקול ההצפנה TLS 1.3 וגרסאותיו המתקדמות.

האלגוריתם משלב בין תכונות בלתי מוצפנות מלחיצת הידיים (TLS handshake) לבין תכונות זרימתיות (flow-based features) - מה שמאפשר סיווג מדויק ומהיר יותר גם כאשר מטא-נתונים רגישים מוסתרים על ידי ECH. בנוסף, אחד ההישגים הבולטים של המאמר הוא יצירת מסד נתונים רחב ומגוון של למעלה מ-600,000 זרימות TLS, שנאספו ממדינות בצפון אמריקה, אירופה ואסיה, מה שמעניק למחקר בסיס אמין וייצוגי.

אילו תכונות תעבורה המאמר משתמש ואילו מהן חדשות:

המאמר עושה שימוש בתכונות תעבורה שונות, אותן הוא מחלק לשלוש קטגוריות עיקריות:

תכונות חבילה: (packet-based features)

נתונים בלתי מוצפנים מתוך הודעות ה "ClientHello" וה "ServerHello" - בפרוטוקול TLS כולל גרסת הפרוטוקול, קבוצות מפתחות (Groups), (Key Share) ורשימת צופן (Cipher Suites).

שימוש באלגוריתם "Random Forest" לניתוח סדר ומבנה של התוספים (extensions) הבלתי מוצפנים שמועברים במהלך ההידוק הידיים.

תכונות זרימתיות: (flow-based features)

גדלי פקטות: (Packet Sizes - PS) סטטיסטיקות כמו מינימום, מקסימום, ממוצע וסטיית תקן של גדלי המנות הנשלחות והמתקבלות.

זמני הגעה בין פקטות: (Inter-Packet Times - IPT) מדידת זמני הגעה בין מנות רציפות והפקת מדדים סטטיסטיים כמו חציון, אחוזון 25 ו-75, וממוצע.

תכונות היברידיות:

שילוב התכונות החבילות והזרימתיות לכדי וקטור תכונות אחד שמזן לאלגוריתם "Random Forest" שילוב זה מאפשר התחשבות הן במידע זמין מתוך ה TLS handshake והן בדפוסים זרימתיים שמאפיינים סוגי תעבורה שונים.

החידוש המרכזי בא לידי ביטוי בכך שהאלגוריתם hRFTC מרחיב את היכולות של האלגוריתם הקודם, "RB-RF", כך שיוכל להתמודד עם תעבורה של פרוטוקול "HTTP/3" המבוסס על "QUIC". בניגוד לשיטות אחרות, האלגוריתם החדש מתאים עצמו לתנאי ההצפנה המחמירים של ECH, תוך שמירה על מהירות ואמינות.

ממצאים מרכזיים ותובנות מהתוצאות:

שיפור משמעותי בדיוק הסיווג: האלגוריתם hRFTC הצליח להגיע לדיוק מרשים של 94.6% במדד ה F-score זאת בהשוואה לאלגוריתמים מבוססי TLS בלבד, שהשיגו דיוק של 38.4% בלבד. נתון זה מדגיש עד כמה חיוני לשלב בין תכונות זרימתיות ותכונות חבילות על מנת להשיג סיווג מדויק בתנאים של הצפנת ECH.

אי-יעילות של אלגוריתמים מבוססי TLS בלבד: התוצאות מצביעות על כך שהסתמכות בלעדית על תכונות TLS אינה מספיקה עוד לסיווג מדויק של תעבורה, בגלל העובדה שפרוטוקול ECH מצפין מטא-נתונים קריטיים כמו "Server Name Indication" (SNI) גם האלגוריתמים המתקדמים ביותר המתמקדים רק בנתונים החבילתיים התקשו להגיע לרמת דיוק גבוהה.

תלות גיאוגרפית: אחד הממצאים המרתקים של המחקר הוא ההשפעה המשמעותית של המיקום הגיאוגרפי על איכות הסיווג. האלגוריתמים שנאמדו על נתוני תעבורה שנאספו במדינה אחת התקשו לשמור על אותה רמת דיוק כאשר נוסו על נתוני תעבורה ממדינות אחרות. הדבר נובע מהבדלים במבנה הרשת, השימוש ב "Content Delivery Network" (CDN), ושינויים ברמת קיבוץ המנות.

גמישות בכמות נתונים: המחקר הדגים שיפור ניכר בגמישות האלגוריתם hRFTC גם כאשר נעשה שימוש ב-10% בלבד ממסד הנתונים לצורך אימון, רמת הדיוק נותרה כמעט זהה לזו שהתקבלה כאשר נעשה שימוש ב-70% מהנתונים. נתון זה מצביע על יכולת ההכללה (generalization) המרשימה של האלגוריתם, המאפשרת יישום מהיר ויעיל גם במקרים שבהם כמות הנתונים הזמינים מוגבלת.

FlowPic: Encrypted Internet Traffic Classification is as Easy as Image Recognition

במאמר מספרים לנו על פיתוח גישה חדשנית לסיווג תעבורת אינטרנט מוצפנת על ידי שימוש בטכניקות עיבוד תמונה.

במקום להסתמך על שיטות מסורתיות של ניתוח תכונות סטטיסטיות, החוקרים מציעים להמיר את נתוני התעבורה לתמונות flowpics ולאחר מכן להשתמש ברשתות עצביות (CNN) לסיווג התמונות.

הגישה מאפשרת להבחין בדפוסים מורכבים בתעבורה שקשה לזהות בשיטות אחרות, במיוחד שהתעבורה מוצפנת.

המאמר מתמקד בתכונות תעבורה שקשורות לגודל וזמן (שהן בדר"כ זמינות גם בתעבורה מוצפנת).

החידוש המרכזי הוא אופן הניתוח של התכונות האלה.

כשנמיר תעבורה לתמונות flowpics, נוכל במקום לחלץ תכונות סטטיסטיות באופן ידני, להמיר את נתוני הזרימה לתמונה דו-ממדית כך שציר ה-X של התמונה מייצג את זמן ההגעה של "המנות" וציר ה-Y מייצג את הגודל. (כל פיקסל בתמונה מייצג את מספר המנות שהגיעו בטווח זמן מסוים וגודל מסוים).

לאחר יצירת התמונות נשתמש ברשתות CNN לסיווג, הרשתות האלה (CNN) ידועות ביכולתן ללמוד את הדפסים המורכבים בתמונות והן מתאימות במיוחד לזיהוי דפוסים בתעבורת הרשת.

התוצאות המרכזיות במאמר מוצגות מרשימות על סטים שונים של נתונים, כולל תעבורה מוצפנת באמצעות VPN (Virtual Private Network) ו-Tor (The Onion Router).

בין היתר, דיוק גבוה בסיווג קטגוריות תעבורה (דיוק של 90%), יכולת טובה בזיהוי יישומים חדשים (יישומים שלא היו חלק מסט האימונים), ביצועים משופרים בהשוואה לעבודות קודמות (שהשתמשו בשיטות מסורתיות) ויכולת להתמודד עם בעיות סיווג מורכבות (כגון סיווג תעבורה מוצפנת וזיהוי יישומים חדשים).

השיטה הזו טובה במיוחד כי היא לא מסתמכת רק על תכונות סטטיסטיות שחולצו באופן ידני, אלא לוכדת דפוסים מורכבים בתבנית התעבורה על ידי יצירת ייצוג חזותי של זרמי תעבורה.

בנוסף היא יכולה לסווג תעבורה מוצפנת מבלי לפענח אותה (זה שימושי מאוד לניתוח אבטחת רשת).

מסקנות סופיות:

המרת תעבורה לתמונות מאפשרת ללכוד דפוסים מורכבים שקשה לזהות בשיטות אחרות.

רשתות CNN הן כלי יעיל לסיווג תעבורת רשת, במיוחד כאשר התעבורה מוצפנת.

הגישה המוצעת היא בעלת פוטנציאל גבוה לשימוש ביישומי אבטחת רשת וניתוח תעבורה.

התוצאות מראות שהשיטה החדשה משיגה דיוק גבוה יותר בהשוואה לשיטות מסורתיות של ניתוח תכונות סטטיסטיות. בפרט, היא מאפשרת לזהות תעבורה מוצפנת גם כאשר אין גישה למידע החבילות עצמן.

Analyzing HTTPS Encrypted Traffic to Identify User's Operating System, Browser and Application

במאמר מסבירים לנו על תוקף חיצוני שיכול לזהות את - מערכת ההפעלה, האפליקציה והדפדפן דרך תעבורת ה-HTTPS המוצפנת שלו מבלי לפענח ולנתח את התעבורה עצמה, בעצם התוקף יכול לעשות זאת דרך הבנת התבנית של התעבורה ובעזרת אלגוריתמים וטכניקות של machine learning.

לפי המאמר זו פריצת דרך ראשונית שבעזרתה אפשר להבין על כל השלושה (מערכת ההפעלה, דפדפן והאפליקציה) מתנועה מוצפנת.

המאמר מציג תכונות חדשות לניתוח התעבורה המוצפנת שמבוססות על התנהגות SSL (Secure Sockets Layer) שזה פרוטוקול הצפנה ועל דפוסי תעבורה בולטים של הדפדפנים.

בין היתר למרות שהנתונים מוצפנים ב-HTTPS עדיין ניתן להפיק מידע מהמטא-דאטה של הנתונים, בגלל שכל מערכת הפעלה ודפדפן משתמשים בקונפיגורציות שונות של SSL נוכל להבדיל ביניהם.

(למשל, Chrome ו-Firefox תומכים בצופן שונה מאשר Internet Explorer, ומערכת Windows עשויה להתנהל אחרת מ-Linux בנוגע לניהול סשן SSL).

החוקרים השתמשו בכלי אוטומציה בשם Selenium כדי להפעיל דפדפנים וליצור תעבורה אמיתית.

הכלי SplitCap שימש לפיצול התעבורה לקבצים נפרדים לפי "sessions" (כל session מוגדר כ-Tuple- <Protocol, IP source, IP destination, Port source, Port destination>).

המאמר מציג דיוק של 96.06% בזיהוי מערכת ההפעלה, הדפדפן והאפליקציה של המשתמש, על בסיס ניתוח תעבורת רשת מוצפנת (HTTPS).

המאמר מחלק את מאפייני התעבורה לשתי קטגוריות: מאפיינים בסיסיים (תכונות ממחקרים קודמים) ומאפיינים חדשים (תכונות שהוצגו במאמר זה ומשפרות את הדיוק).

המאפיינים הבסיסיים היו מספר חבילות נתונים (כמה נשלחו וכמה התקבלו), סך כל הנתונים (כמה מידע עבר בבתיים), הפרשי זמן בין חבילות, פרמטרים של TCP ועוד.

המאפיינים החדשים היו בעיקר קשורים ל-SSL ולדפוסי תעבורה בולטים, למשל מספר הרחבות (לכל דפדפן שיטות מימוש שונות), מספר שיטות הצפנה (סוגי האלג' להצפנה), אורך מזהה של Session של SSL (שיכול להעיד על ההגדרות והפרוטוקולים של מערכת ההפעלה או הדפדפן).

בין היתר המאפיינים החדשים חשובים כי הם משפרים את הדיוק מאוד! כלומר בעזרת השילוב המאפיינים הישנים והחדשים הגענו לדיוק כל כך גבוה.

תוצאות דיוק:

למאפיינים הבסיסיים: 93.51%, למאפיינים החדשים: ביצוע בר השווה, למאפיינים החדשים והישנים ביחד: 96.06%.

זיהוי כמעט מושלם של מערכת ההפעלה, זיהוי בדיוק גבוה של הדפדפן (לפעמים טעות נניח בין chrome ל-safari), וזיהוי אפליקציה שבגדול מדויק.

בדו"ח נצליח להבחין בין הדפדפנים בעזרת תבנית התעבורה שלהם.

מסקנות סופיות:

הצפנת HTTPS היא לא מוגנת במלואה ולא שומרת לחלוטין על פרטיות המשתמש.

המאפיינים החדשים הנ"ל (Bursty SSL) עוזרים מאד לשפר את הדיוק.

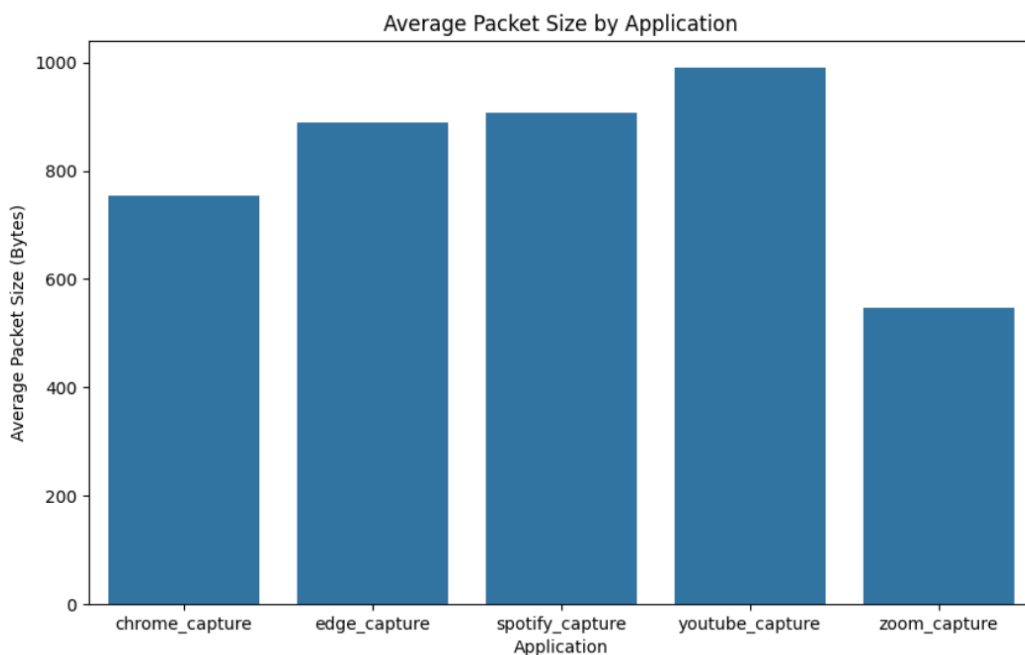
תוקפים יכול למנף את הדבר למעקב, איתור ובחירת מטרה למתקפות סייבר.

חלק שלישי:

1. הקלטות(נמצא בתוך התיקיה question_3 תחת התיקיה recordings) כדי להפחית רעש בתעבורה בזמן ההקלטות, נעשה כמה צעדים לפני תחילת ההקלטות: נכבה עדכונים אוטומטיים, נסגור אפליקציות שלא קשורות למה שאנחנו בודקים ונכבה התראות במערכת.
*כמובן שיש בנוסף עוד הרבה צעדים שניתן לעשות כדי להפחית רעש
כאשר אנחנו גולשים באינטרנט/משתתפים בשיחת וידאו/וכו', התעבורה מוצפנת באמצעות TLS- דבר שיקשה על ניתוח המאפיינים שלה. TLS מצפין את התוכן של ההודעות, מה שגורם לכך שה-wireshark מציג את הנתונים תחת "Encrypted Application Data" בלי לחשוף את התוכן שבפנים.
כדי לנתח את התעבורה ולהשוות בין מאפייני האפליקציות השונות (גלישה, סטרימינג ושיחות וידאו), אנחנו צריכים לפענח את ההצפנה של TLS ולחלץ את הנתונים. הדרך לעשות זאת היא באמצעות שמירת מפתחות ההצפנה (TLS Keys). המפתחות האלה הם בעצם הסיסמאות שבעזרתן אפשר לפתוח את ההצפנה ולראות את התוכן המקורי.
נגדיר משתנה סביבה וקובץ לוג שלשם יתווספו המפתחות בכל פעם שנרצה להקליט(הקובץ נמצא בתיקיה sslkeys_all-recordings).
- 2+3. לאחר שיש לנו את ההקלטות ואת המפתחות נרצה לפענח אותם- נכנס דרך הווירשארק להגדרת המסלול של המפתחות(אתם תצטרכו לעשות זאת גם אצלכם כי זה לא נשמר דרך הקלטות אלא במחשב עצמו- כמו צביעת חבילות).
במהלך הניתוח Wireshark הצליח לפענח חלק מהחבילות, אך לא את כולן. הסיבה לכך היא ש TLS 1.3 משתמש במפתחות זמניים. כמו כן, תעבורה המשתמשת בפרוטוקול QUIC לא ניתנת לפענוח, מכיוון שהיא לא משתמשת באותם מפתחות TLS הרגילים. לכן, למרות שהקלטנו מראש את מפתחות TLS חלק מהתעבורה עדיין נשארה מוצפנת ולא הייתה ניתנת לפענוח.
החלטנו **לא לסנן את החבילות שנשארו מוצפנות**, מכיוון שזה היה משפיע על הדיוק של הגרפים. חבילות מוצפנות עדיין מכילות מידע חשוב כמו גודל החבילה וזמן ההגעה שלה, ולכן הן חשובות לניתוח סטטיסטי אמיתי. אם היינו מסננים רק את הנתונים הקריאים, היינו משנים את תבנית התעבורה של האפליקציות וגורמים להטיה בתוצאות. לכן, כדי להבטיח שהגרפים ישקפו את ההתנהגות האמיתית של התעבורה, השארנו גם את החבילות המוצפנות בניתוח שלנו.
הגרפים:
- כדי להריץ את הקוד מצורף README עם מה שצריך להתקין.
לפני שנגיע לתוצאות של הגרפים נרצה לחשוב בעצמנו מה אנחנו מצפים לראות בכל גרף, נראה את הגרף ונסביר את התוצאה ומה שונה ממה שציפינו.

נפח תנועה ממוצע לחבילה לכל אפליקציה:

בגרף זה ניתן לראות את גודל החבילה הממוצע בכל אחת מהאפליקציות שנבדקו. ציפינו ששירותי סטרימינג כמו YouTube ו-Spotify יכילו חבילות גדולות, מכיוון שהם מעבירים תוכן מדיה. בנוסף, דפדפנים כמו Chrome ו-Edge אמורים להציג חבילות בגודל בינוני, מאחר שהם טוענים גם דפי אינטרנט וגם קבצי מדיה קטנים Zoom. היה צפוי להציג חבילות בגודל דומה ל YouTube כי שיחות וידאו אמורות לכלול חבילות גדולות.

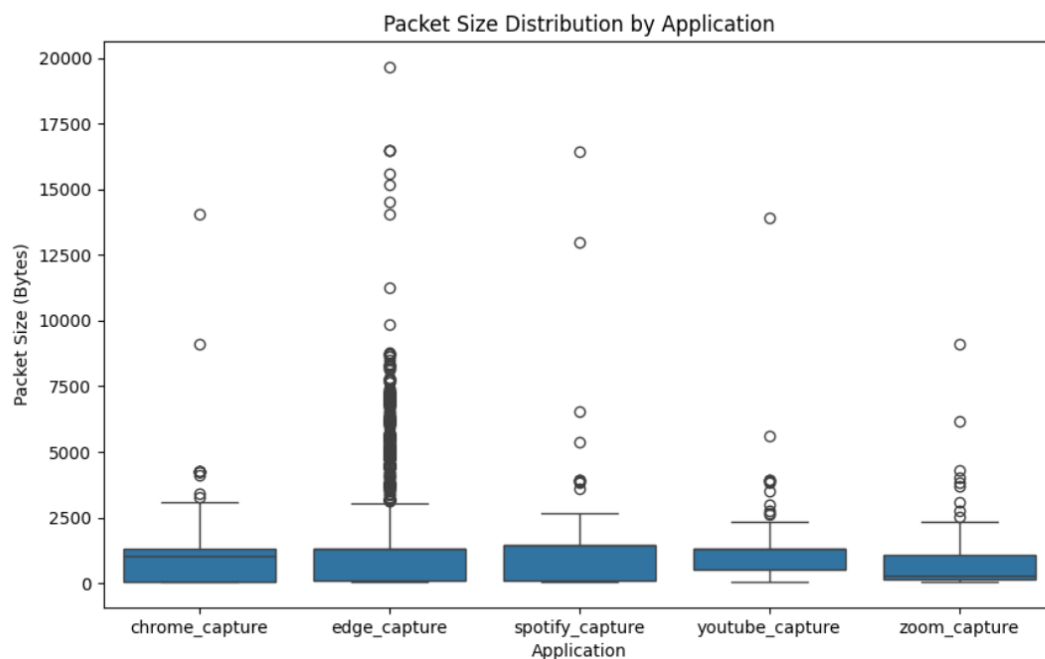


מה יצא בפועל:

התוצאות בפועל מראות ש YouTube אכן מכיל את החבילות הגדולות ביותר Spotify ו Edge מציגים חבילות בגודל בינוני, כפי שציפינו. לעומת זאת Zoom מציג חבילות קטנות יותר מהצפוי, עם גודל ממוצע של כ-500 בייט, מה שלא תואם את ההנחה הראשונית שלנו.

גודל חבילות לפי אפליקציה:

הגרף מציג את ההתפלגות של גדלי החבילות עבור כל אפליקציה שנבדקה. ציפינו ששירותי סטרימינג כמו YouTube ו Spotify יציגו חבילות גדולות יותר בממוצע, מכיוון שהם מעבירים מדיה. Zoom היה צפוי להציג גם כן חבילות גדולות, מאחר ששיחות וידאו דורשות העברת נתונים כבדה. Chrome ו Edge היו צפויים להראות טווח גדול של גדלי חבילות, מכיוון שגלישה כוללת בקשות קטנות (כמו טעינת דפי אינטרנט) וגם הורדות גדולות יותר של תוכן.



מה יצא בפועל:

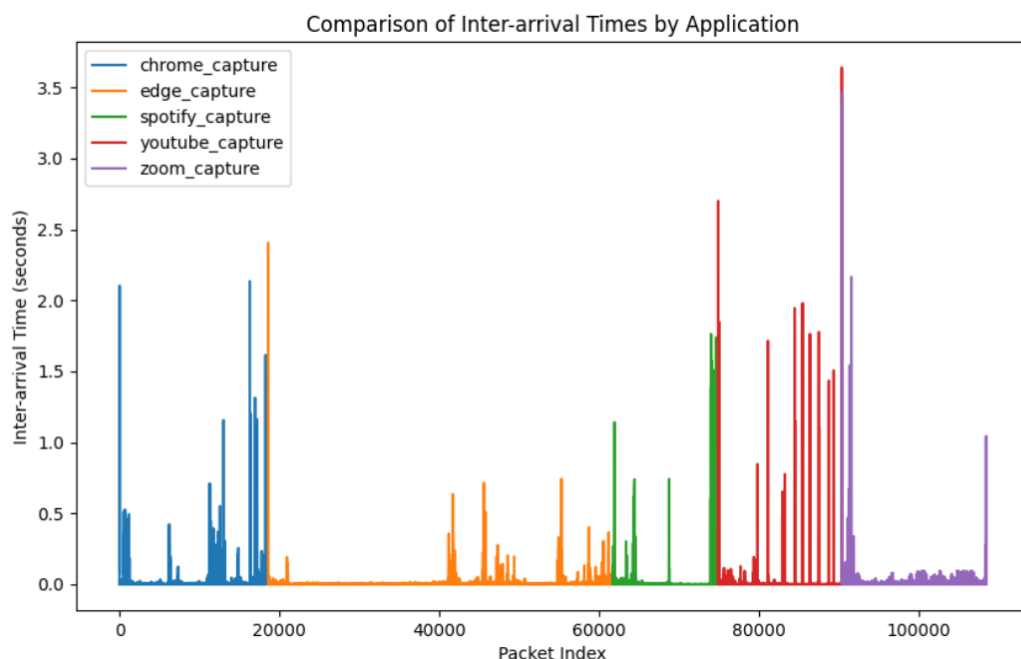
בפועל, ניתן לראות שהתפלגות גדלי החבילות דומה מאוד בין כל האפליקציות, כאשר רוב החבילות נמצאות בטווחים הנמוכים (מתחת ל-2000 בייט). Edge מציג חריגות רבות של חבילות גדולות בהשוואה לשאר האפליקציות.

מה שונה ממה שציפינו:

השוני העיקרי מהציפיות הוא בכך ששירותי סטרימינג כמו YouTube ו Spotify לא מציגים חבילות גדולות באופן קבוע, אלא בעיקר חבילות קטנות עם מספר קטן של חריגות גדולות. Edge לעומת זאת, מציג הרבה חריגות של חבילות גדולות.

השוואת זמני הגעה בין חבילות:

גרף זה מציג את זמני ההגעה בין חבילות עבור כל אחת מהאפליקציות. ציפינו לראות ערכים נמוכים וקבועים בשירותים שמזרימים תוכן באופן רציף כמו YouTube ו-Spotify מכיוון שהם שולחים חבילות באופן עקבי. לעומת זאת, בדפדפנים כמו Chrome ו-Edge ציפינו לראות זמני הגעה משתנים יותר, בגלל שחבילות נשלחות בהתאם לטעינת דפים ולבקשות משתמש. Zoom היה צפוי להציג זמני הגעה נמוכים וקבועים יחסית, מכיוון ששיחות וידאו מחייבות זרימה רציפה של נתונים.



מה יצא בפועל:

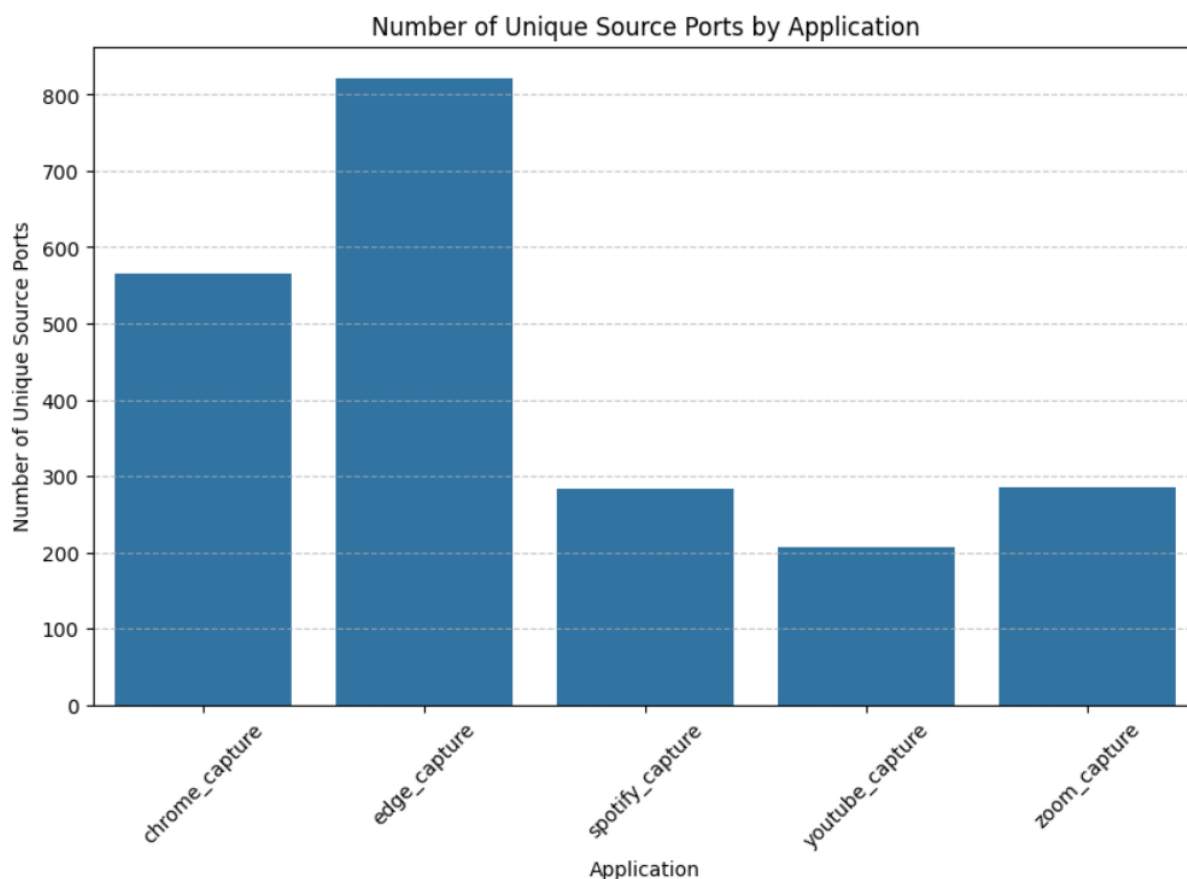
בפועל Chrome (כחול) ו-Edge (כתום) מציגים זמני הגעה לא אחידים עם קפיצות חדות, בדיוק כפי שציפינו, משום שהתעבורה בדפדפנים תלויה בפעולות המשתמש. Spotify (ירוק) מציג זמני הגעה יחסית יציבים, אך עם עליות מסוימות שמראות אולי שהתעבורה לא זורמת באופן מושלם. YouTube (אדום) מציג קפיצות משמעותיות בזמני ההגעה, דבר שמצביע על כך שהוא אינו שולח חבילות ברצף אחיד, אלא מקבץ אותן במרווחים משתנים (יכול להיות בגלל buffer). Zoom (סגול) מציג דפוס שבו רוב הזמן זמני ההגעה נמוכים ויציבים.

מה שונה ממה שציפינו:

השוני העיקרי מהציפיות הוא בכך ש-YouTube לא מציג זרימה חלקה כמו שחשבנו, אלא שולח חבילות בקבוצות עם הפסקות, כנראה בגלל טעינה מראש כפי שרשמנו למעלה. Spotify שומר על קצב יציב יותר, אך עדיין לא אחיד לחלוטין. Zoom מציג קפיצות מפתיעות בהתחלה ובסוף שזה יכול להיות בגלל שזה לא היה בזמן השיחה עצמה אלא בקבלת הקישור וכניסה לשיחה וסיום השיחה.

מספר פורטי מקור ייחודיים לפי אפליקציה:

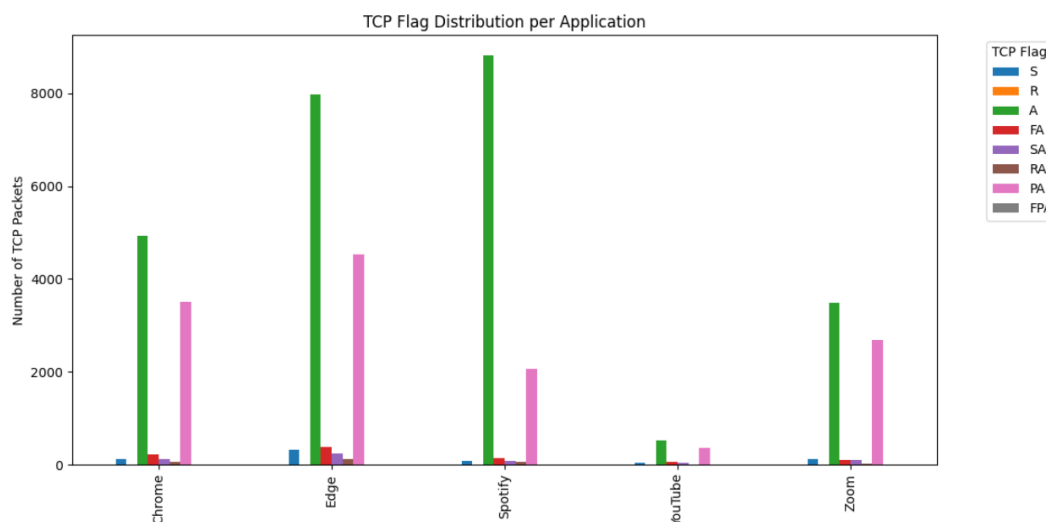
הגרף מציג את מספר הפורטים הייחודיים ששימשו כ Source Port עבור כל אפליקציה. ציפינו שדפדפנים כמו Chrome ו Edge יציגו מספר גבוה של פורטים ייחודיים, בגלל שכל חיבור חדש לאתר עשוי להשתמש בפורט מקור שונה. YouTube ו Spotify צפויים להציג מספר נמוך יותר של פורטים, כי הם משתמשים לרוב בחיבורים מתמשכים (Persistent)

**מה יצא בפועל:**

בפועל, ניתן לראות כי Edge מכיל את מספר הפורטים הייחודיים הגבוה ביותר, מה שמצביע על כך שהוא פותח חיבורים רבים עם פורטים משתנים. Chrome מציג גם מספר כי דפדפנים יוצרים חיבורים נפרדים לכל דף או שירות. YouTube ו Spotify מציגים מספר נמוך יחסית של פורטים ייחודיים, מה שמעיד על כך שהתקשורת שלהם מבוססת על פחות חיבורים מתחלפים. Zoom מציג מספר בינוני של פורטים ייחודיים.

התפלגות דגלי TCP לפי אפליקציה:

נראה אילו דגלי TCP נפוצים בכל אפליקציה. למשל, אם אפליקציה מבצעת הרבה חיבורי TCP חדשים, ייתכן שנראה יותר דגל SYN אם יש הרבה סגירות חיבור, ייתכן שנראה יותר דגלי FIN



מה יצא בפועל:

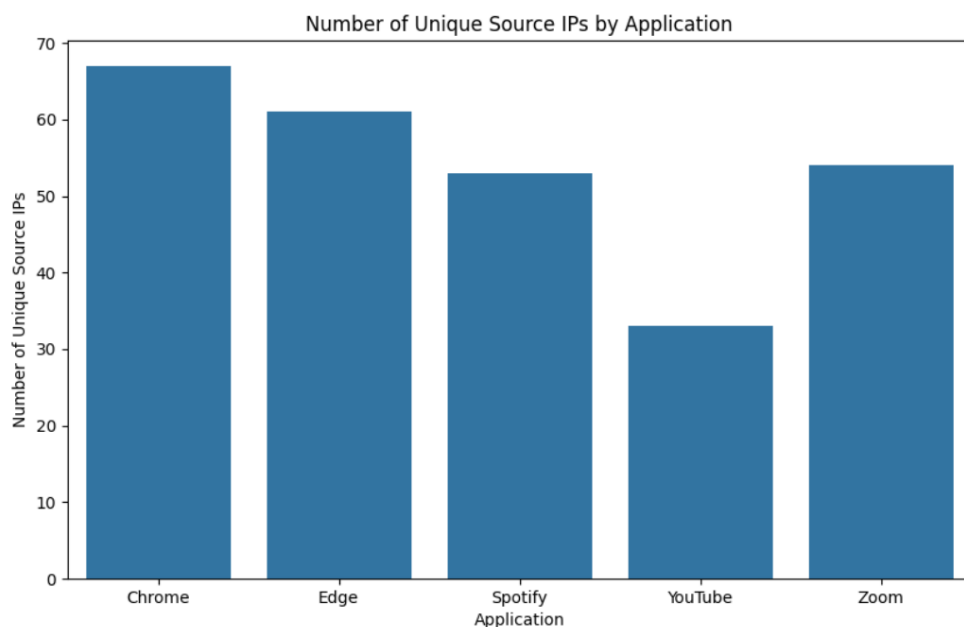
בגרף רואים שבכל האפליקציות יש בעיקר שתי קבוצות עיקריות של דגלי TCP שמופיעות בתדירות גבוהה (ACK: המסומן בגרף A, ו PSH/ACK מסומן PA).

מה שונה ממה שציפינו:

ציפינו למצוא הרבה SYN ו-FIN בחלק מהאפליקציות (במיוחד דפדפנים שפותחים וסוגרים חיבורים לעיתים קרובות), אך בפועל עיקר התעבורה מורכבת מחבילות ACK, ו PSH/ACK-כנראה מכיוון שרוב החיבורים נשארו פעילים לאורך זמן ולא נפתחו ונסגרו שוב ושוב. אפליקציות כמו Zoom, Spotify ו-Edge מציגות נפח גדול של ACK כי הן מזרימות נתונים רציפים, מה שיוצר צורך רב באישורים (ACK). YouTube מציג מעט מאוד חבילות מכל סוג.

מספר כתובות IP ייחודיות לפי אפליקציה:

נראה כמה כתובות IP שונות כל אפליקציה מתקשרת איתן. לדוגמה, דפדפנים כמו Chrome ו-Edge נצפה שיציגו מספר גבוה של כתובות IP ייחודיות כי הם ניגשים לאתרים רבים, בעוד אפליקציות כמו Zoom או Spotify עשויות לתקשר עם מספר קטן יחסית של שרתים קבועים.

**מה יצא בפועל:**

Chrome מוביל עם הכי הרבה כתובות מקור ייחודיות (כ-70), אחריו, Edge ואז Spotify ו-Zoom. במספרים מעט נמוכים יותר YouTube. מציג את המספר הנמוך ביותר של כתובות מקור ייחודיות.

4. המתקיף יכול לנסות לזהות את האפליקציות שהמשתמש השתמש בהן על ידי ניתוח של תבניות התעבורה ברשת. התהליך הזה יקרה גם אם התעבורה מוצפנת או כאשר אין לו גישה ישירה לתוכן של התעבורה, אבל הוא יכול ללמוד על דפוסי השימוש של האפליקציות על פי תכונות כמו גודל החבילות וחומות הזמן.

Web surfing:

כשמדובר בגלישה באתרים באמצעות דפדפנים כמו Google Chrome או MS Edge המתקיף יכול להבחין בכך שהתעבורה כוללת חבילות קטנות שמגיעות בפרקי זמן קצרים יחסית. חבילות קטנות עם הפסקות בין כל חבילה יכולות להעיד על גלישה באתרים שונים. לדוגמה, כל טעינת דף אינטרנט או תמונה תיצור חבילה קטנה מאוד. מכיוון שהדפדפנים השונים עשויים לשלוח בקשות HTTP/HTTPS עם תכנים שונים- תמונות, קבצי CSS, סקריפטים וכו' - המתקיף עשוי לזהות את התעבורה הזו כגלישה באתרים, אך יהיה קשה לו להבחין אם מדובר ב-Chrome או MS Edge כי חבילות התעבורה יהיו דומות מאוד.

Spotify:

בשירותים כמו Spotify המתקיף יכול לזהות דפוסי תעבורה שמרמזים על סטרימינג של אודיו, מכיוון שהתעבורה תהיה רציפה ואחידה יותר לעומת גלישה רגילה באתרים. כל חבילה שתשלח תהיה בגודל בינוני או גדול יחסית, ותהיה תעבורה מתמשכת, כלומר חבילות ישלחו בצורה רציפה ללא הפסקות ארוכות. המתקיף יכול לזהות כפעולה של סטרימינג אודיו, כי השירותים האלה שולחים מידע בצורה קבועה.

Youtube:

שירותים כמו YouTube דורשים כמות רבה מאוד של תעבורה כדי לשלוח וידאו באיכות טובה. המתקיף יבחין בכך שהתעבורה כוללת חבילות גדולות מאוד שמגיעות בתבנית רציפה, כלומר חבילה אחרי חבילה. אם הוא יראה תעבורה שבה יש חבילות גדולות יחסית, בפרקי זמן קבועים, הוא יוכל להניח שמדובר בסטרימינג של וידאו, לדוגמה YouTube. כל שיחה או תנועה של סרטון תדרוש כמות רבה של נתונים, כך שהמתקיף יוכל להבין שמדובר בתעבורה שמיועדת להעברת וידאו. בנוסף, אם ישנן חבילות גדולות עם זמן קצר ביניהם, יתכן שזה יהיה סטרימינג וידאו.

Zoom:

אפליקציות כמו Zoom גם כן ידרשו תעבורה רציפה, אך היא תהיה שונה בסוגה. המתקיף יכול להבחין בכך שהתעבורה לא תהיה יציבה כמו סטרימינג של וידאו, אלא תשתנה לפי מספר המשתמשים והאיכות של השיחה. חבילות גדולות מאוד עשויות להופיע כשיש שיחה עם וידאו באיכות גבוהה, בעוד שבשיחה עם מספר מועט של אנשים ואיכות וידאו נמוכה יהיו חבילות קטנות יותר. המתקיף יוכל להבחין כי מדובר בתקשורת וידאו כאשר יש דפוסים של חבילות שמתפצלות למספר מקטעים בתוך פרקי זמן קצרים. הוא יכול גם לזהות תעבורה של אפליקציות כמו Zoom על פי החבילות הגדולות שנשלחות בצורה רציפה, ובסופו של דבר להבין שמדובר בשיחת וידאו.

כאשר משתמשים בשירותי אנונימיזציה כמו VPN או Tor התעבורה מוצפנת כך שהתוקף לא יכול לדעת לאילו אתרים או אפליקציות המשתמש מתחבר ישירות. ב-vpn כל החבילות עוברות דרך שרת מרכזי, כך שהתוקף רואה רק חיבור כללי ל-VPN אך לא יודע מה היעד הסופי. ב-TOR החבילות מועברות דרך כמה שרתים מוצפנים, מה שמסתיר את המקור והיעד, אך עדיין משאיר דפוסי תעבורה שניתן לנתח. למרות שהתוקף לא רואה את כתובת ה-IP של השרתים אליהם המשתמש מתחבר, הוא יכול לנסות לזהות את האפליקציה לפי דפוסי השידור. אם הוא רואה חבילות גדולות שנשלחות ברצף, הוא יכול להניח שמדובר בשירות סטרימינג כמו YouTube. אם החבילות קטנות ומופיעות עם הפסקות קצרות, זה עשוי לרמוז על גלישה באינטרנט. אם התעבורה קבועה לאורך זמן, ייתכן שמדובר בהזרמת אודיו כמו Spotify או שיחת וידאו zoom. מאחר שהתוקף עדיין יכול ללמוד מהמאפיינים הללו, ניתן להקשות עליו עוד יותר באמצעות טכניקות כמו ריפוד חבילות, טשטוש זמנים ושימוש

בפרוטוקולים כמו HTTP/2 שמערבבים זרמי תעבורה יחד. כך, גם אם התעבורה מוצפנת או אנונימית, יהיה קשה הרבה יותר לזהות איזו אפליקציה נמצאת בשימוש.

לסיכום, למרות שהתוקף לא יכול לקרוא את התוכן עצמו, הוא עדיין יכול ללמוד על סוג האפליקציה שבה נעשה שימוש על סמך גודל החבילות, זמני ההגעה ורציפות השידור. שימוש בהצפנה ובאנונימיות מקשה עליו מאוד, אבל לא תמיד מונע לחלוטין את הזיהוי.

נבנה מודל:

כפי שנאמר בפורום ניתן לקחת דאטה סט קיים ולעשות על זה מודל למידה.

ניקח דאטה שאושר לשימוש בפורום אשר מוזכר במאמר:

"FlowPic_Encrypted_Internet_Traffic_Classification_is_as_Easy_as_Image_Recognition" שבו מצוין הדאטה - "NonVPN-PCAPs"

מכיוון שהקלטות המקוריות (PCAP/PCAPNG) תופסות מקום רב (כ-800 מ"ב), החלטנו לבצע את כל שלבי ההכנה המקדימים אצלנו ולספק לכם ישירות את הדאטה סט הסופי.

במקום להמיר מחדש את ההקלטות, תוכלו להריץ את הקוד הסופי ques_4.py על הדאטה סט המוכן (combined_dataset.csv) ולהגיע ישירות לשלב של אימון המודל ולחיזוי האפליקציות.

לצורך שקיפות, צירפנו צילומי מסך שמראים את תהליך ההמרה.

בשלב הראשון של העבודה, לקחנו את ההקלטות מהאתר שנאמר לנו במאמר. ההקלטות שנלקחו היו בקבצי pcap/pcapng הכוללים את תעבורת הרשת. כל קובץ כזה מכיל נתונים על פרמטרים שונים של חיבורי רשת. בעזרת הקוד שלנו "from_rec_to_dataset" המרנו כל אחת מההקלטות לקובץ csv. בכל קובץ ששמרנו, הוספנו עמודה חדשה בשם "Classification", שבה שומרים את קטגוריית האפליקציה של אותה הקלטה. הקטגוריות הוגדרו מראש, והשמות של ההקלטות שימשו כקטגוריות. כך, כל קובץ הומר לפורמט שניתן לעבוד איתו ולהתאימו לצרכים של ניתוח ולמידת מכונה. בנוסף, הוספנו עמודה חדשה בשם "Inter-Arrival Time" המחושבת כהפרש הזמן בין חבילה אחת לחבילה הבאה אחריה בתוך אותו זרם תעבורה. החלטנו להוסיף נתון זה, משום שבשאלה 4 התוקף יודע את חותמות הזמן (Timestamp) של החבילות, ולכן הוא יכול לחשב את ההפרשים בעצמו. מאחר שלכל אפליקציה יש דפוסי שליחת חבילות ייחודיים, מידע זה יכול לעזור בניתוח ולסייע לזיהוי האפליקציה גם כאשר התעבורה מוצפנת.

מצורף צילום של הקוד:

from_rec_to_dataset.py x

```

1  import os
2  import pandas as pd
3  from scapy.all import rdpcap
4  import hashlib
5
6  # Function to extract the classification (file name up to the underscore)
7  def extract_classification(file_name): 1usage
8      return file_name.split('_')[0]
9
10 # Function to create a hash of the 4-tuple
11 def hash_4tuple(source_ip, dest_ip, source_port, dest_port): 1usage
12     tuple_string = f"{source_ip}_{dest_ip}_{source_port}_{dest_port}"
13     return hashlib.sha256(tuple_string.encode()).hexdigest()
14 # Function to convert a pcap file into a dataframe
15 def pcap_to_dataframe(pcap_file): 1usage
16     packets = rdpcap(pcap_file)
17     data = []
18     for packet in packets:
19         if packet.haslayer('IP'):
20             source_ip = packet['IP'].src
21             dest_ip = packet['IP'].dst
22             size = len(packet)
23             timestamp = packet.time
24             # Support both TCP and UDP
25             if packet.haslayer('TCP'):
26                 source_port = packet['TCP'].sport
27                 dest_port = packet['TCP'].dport
28             elif packet.haslayer('UDP'):
29                 source_port = packet['UDP'].sport
30                 dest_port = packet['UDP'].dport
31             else:
32                 source_port = None
33                 dest_port = None
34             # Create the hash of the 4-tuple (for scenario 1)
35             flow_hash = hash_4tuple(source_ip, dest_ip, source_port, dest_port)
36             # Add the data to the dataframe
37             data.append([source_ip, dest_ip, size, timestamp, source_port, dest_port, flow_hash])
38     df = pd.DataFrame(data,
39                       columns=["Source IP", "Dest IP", "Size", "Timestamp", "Source Port", "Dest Port", "Flow Hash"])
40
41     # Add classification category (file name up to the underscore)
42     classification = extract_classification(os.path.basename(pcap_file))
43     df['Classification'] = classification
44     # Compute inter-arrival time (since attacker can infer it)
45     df['Inter-Arrival Time'] = df.groupby('Flow Hash')['Timestamp'].diff().fillna(0)
46
47     return df
48
49 # Function to process all pcap and pcapng files in the directory and save the dataset in the output directory
50 def process_pcap_files(input_dir, output_dir): 1usage
51     # Create the output directory if it doesn't exist
52     if not os.path.exists(output_dir):
53         os.makedirs(output_dir)
54     all_files = os.listdir(input_dir) # Get all files in the input directory
55     pcap_files = [f for f in all_files if f.endswith((".pcap", ".pcapng"))] # Filter only pcap and pcapng files
56     # Print files that are not processed
57     for filename in all_files:
58         if not filename.endswith((".pcap", ".pcapng")):
59             print(f"Skipping file (not pcap or pcapng): {filename}")
60     # Process each pcap or pcapng file
61     for filename in pcap_files:
62         print(f"Processing file: {filename}") # Print the current file name
63         file_path = os.path.join(input_dir, filename)
64         try:
65             # Create the dataframe from the pcap file

```

```

62         # Create the dataframe from the pcap file
63         df = pcap_to_dataframe(file_path)
64         # Save the dataframe (single file for both scenarios)
65         output_path = os.path.join(output_dir, f"{filename}.csv")
66         df.to_csv(output_path, index=False)
67         print(f"Saved dataset for {filename} to {output_path}")
68     except Exception as e:
69         print(f"Error processing {filename}: {e}")
70
71 # Call the function
72 input_directory = './NonVPN-PCAPs-01' # Directory with pcap files
73 output_directory = './dataset' # Directory to save the datasets
74
75 process_pcap_files(input_directory, output_directory)

```

לאחר המרת כל ההקלטות, בשלב השני השתמשנו בקוד "datasets_to_dataset" לאיחוד כל הקבצים שהומרו לדאטה סט אחד מאוחד. כל קובץ CSV נטען כ DataFrame נפרד, ולאחר מכן איחדנו את כל ה DataFrames לדאטה סט אחד גדול שמכיל את כל הנתונים. בשלב זה, ביצענו המרה של הערכים בעמודת ה "Classification"-למספרים על פי מילון שהוגדר מראש בקוד. המרת הקטגוריה למספרים מאפשרת לנו לעבוד עם תכונה זו בצורה נוחה יותר לצורך אימון המודל. לדוגמה, קטגוריה כמו audio הומרה למספר 1, video למספר 2, וכן הלאה. כל קובץ שימש כקלט עם התכונות הרלוונטיות וקטגוריית הסיווג המומרת.

```

datasets_to_dataset.py x
1  import os
2  import pandas as pd
3  # Define the dictionary to map categories to numbers
4  category_map = {
5      'aim': 1,
6      'audio': 2,
7      'chat': 3,
8      'email': 4,
9      'video': 5
10 }
11 # Function to process all CSV files in the dataset directory and combine them into one large dataframe
12 def combine_datasets(input_dir): 1usage
13     all_data = []
14     # List all CSV files in the directory
15     for filename in os.listdir(input_dir):
16         if filename.endswith(".csv"):
17             print(f"Processing file: {filename}") # Print the current file being processed
18             file_path = os.path.join(input_dir, filename)
19             # Read the CSV file into a DataFrame
20             df = pd.read_csv(file_path)
21             # Skip empty files
22             if df.empty:
23                 print(f"Skipping empty file: {filename}")

```



```

24         continue
25         # Check if the Classification column exists
26         if 'Classification' not in df.columns:
27             print(f"Skipping file {filename} (missing Classification column)")
28             continue
29         # Convert classification category to a numeric value
30         classification = df['Classification'].iloc[0].lower()
31         category_number = category_map.get(classification, -1) # Default to -1 if not found
32         df['Classification'] = category_number
33         # Append the dataframe to the list
34         all_data.append(df)
35     # If no valid data found, return None
36     if not all_data:
37         print("No valid data found in the dataset directory.")
38         return None
39     # Concatenate all dataframes into one large dataframe
40     combined_df = pd.concat(all_data, ignore_index=True)
41     return combined_df
42 # Function to save the combined dataframe as a single CSV
43 def save_combined_dataset(output_dir, combined_df): # 1 usage
44     if combined_df is None or combined_df.empty:
45         print("No data to save. Exiting.")
46     return
47     # Save full dataset in the same directory as this script
48     output_path = os.path.join(output_dir, 'combined_dataset.csv')
49     combined_df.to_csv(output_path, index=False)
50     print(f"Saved the combined dataset to {output_path}")
51
52 # Path to the directory containing the individual dataset files
53 input_directory = './dataset' # The directory where the individual CSV files are stored
54 output_directory = './' # Save the final dataset in the same directory as this script
55 # Combine the datasets and save the result
56 combined_df = combine_datasets(input_directory)
57 save_combined_dataset(output_directory, combined_df)

```

לאחר שיצרנו דאטה סט אחד מאוחד עם קטגוריות מומרות למספרים, בשלב השלישי השתמשנו בקוד `ques_4` לאימון המודל ולחיזוי הקטגוריות. המודל שבחרנו לאימון היה Random Forest משום שהוא מסוגל להתמודד בצורה טובה עם בעיות סיווג לא לינאריות, כמו אלו שמצאנו בתכונות שהיו לנו. בעזרת Random Forest, האימון על הדאטה אפשר למודל ללמוד את הדפוסים בין התכונות השונות ולחזות את קטגוריית האפליקציה על פי תעבורת הרשת. המודל הופעל על דאטה של אימון ובדיקה, ולאחר האימון ביצענו חיזוי עבור קבוצת הבדיקה והערכנו את ביצועי המודל באמצעות מדדי דיוק.

Random Forest מבוסס על יצירת מספר עצי החלטה של כל אחד מהם יש גישה עצמאית ומגוון החלטות שיכולות לזהות דפוסים לא לינאריים בצורה טובה. כל עץ בודק את הנתונים באופן אקראי ומספק תחזית שונה, ובסופו של דבר אנחנו מאחדים את תוצאות כל העצים כדי לקבל תוצאה מדויקת יותר.

הפשטות והגמישות של Random Forest מאפשרות לנו להשתמש בו בקלות יחסית מבלי צורך בהתאמות מורכבות. כל אלו יחד הופכים את המודל לאופציה הטובה ביותר לפרויקט הזה, שבו אנחנו מנסים לחזות קטגוריות של אפליקציות או סוגי תעבורה ברשת.

התוקף רואה את הזמנים המוחלטים של החבילות, אבל הוא לא יודע מראש את הסיווג שלהם בזמנים אחרים. אם המודל היה מתאמן עם זמן מוחלט, הוא היה מנצל קשרים מלאכותיים בין זמן מוחלט לסיווג – קשרים שהתוקף לא יכול להשתמש בהם במציאות. לכן, במקום זאת, אנחנו משתמשים בהפרשי זמנים (Inter-Arrival Time) כדי לדמות טוב יותר את הידע שהתוקף באמת יכול לנצל.

תוצאות:

```
Warning: Dataset contains missing values.  
Source Port      3916  
Dest Port        3916  
dtype: int64  
✅ Dataset validation passed. Proceeding with processing...  
Accuracy for Case 1 (Flow ID included): 0.9514  
Accuracy for Case 2 (Without Flow ID): 0.8719
```