האוניברסיטה הפתוחה
המחלקה למתמטיקה ולמדעי המחשב
החטיבה למדעי המחשב

<u>פרויקט מתקדם במדעי המחשב:</u>

סיווג הודעות HTTP באמצעות TFIDF

מנחה: פרופ' אהוד גודס

מגיש:
אופיר חכמוף
ת.ז. 311602296

סמסטר ב' 2020

פרויקט מתקדם זה הוגש כחלק מהדרישות לקבלת תואר "מוסמך (M.Sc.) במדעי המחשב"
באוניברסיטה הפתוחה

יוני 2020

1. תוכן עניינים

פרויקט מתקדם במדעי המחשב: 1

סיווג הודעות HTTP באמצעות TFIDF 1

2. תקציר - 4 -

3. מבוא - 4 -

3.1. מבנה המסמך - 4 -

3.2. כללי - 5 -

3.3. תיאור הבעיה - 5 -

3.4. מטרת הפרויקט - 6 -

3.5. חשיבות הפרויקט - 7 -

4. רקע – מונחים עיקריים - 8 -

4.1. פרוטוקול HTTP ([7]) - 8 -

4.2.XSS – Cross Site Scripting ([3],[4],[5],[9],[10]) - 9 -

4.2.1. סוגי XSS - 10 -

4.3.Sql Injection ([5],[11]) - 10 -

4.4.LDAP Injection ([12]) - 12 -

4.5.XPath Injection ([13]) - 12 -

4.6.Path Traversal ([14]) - 14 -

4.7.OS Commanding ([15]) - 15 -

4.8.SSI – Server Side Include ([16], [18]) - 16 -

4.9.Vector Space Model ([8]) - 18 -

5. ניתוח המאמר ([1]) - 19 -

5.1. הקדמה - 19 -

5.2. מטרות - 21 -

5.2.1. מבנה מאגר הנתונים - 21 -

5.3. מודל המסווג - 22 -

5.3.1. שלב הלמידה - 22 -

5.3.2. הפעלת המסווג - 27 -

5.3.3. סיבוכיות זמן - 29 -

5.4. תוצאות המסווג - 30 -

6. הפרויקט - 34 -

6.1. תכולות העבודה - 35 -

6.2. סביבת פיתוח וכלי עבודה - 35 -

6.2.1. הכנת הסביבה להרצה - 36 -

6.2.2. הרצת הפרויקט - 37 -

6.2.3. תוצאת ההרצה .... - 38 -
6.3. אלגוריתם .... - 39 -
6.4. ארכיטקטורה ומבנה הספריות.. - 40 -
Main.6.4.1 .... - 40 -
6.4.2. DatasetHandler .... - 40 -
7. תוצאות ומסקנות .... - 46 -
7.1. תוצאות סיווג על כלל הDataset .... - 49 -
7.2. תוצאות סיווג על 10-Fold Cross Validation .... - 50 -
7.3. תוצאות סיווג על 10-Fold עם shuffling .... - 52 -
7.4. המונחים המשמעותיים ביותר לסיווג .... - 54 -
7.5. בדיקת המסווג על דוגמאות חיצוניות .... - 55 -
7.6. זמן ואופן ההרצה .... - 58 -
8. סיכום והצעות לכיווני מחקר נוספים .... - 59 -
9. רשימת מקורות .... - 61 -

רשימת איורים

איור 1: שימוש ב HTTP לטובת טעינת אתר... - 8 -
איור 2: דף פגיע להזרקת SSI .... - 17 -
איור 3: תוצאת הרצת הSSI הזדוני.. - 17 -
איור 4: הצגת מסמכים ושאילתות כווקטורים במרחב .... - 18 -
איור 5: תקיפת Sql Injection בבקשת HTTP.. - 20 -
איור 6: הסיווגים השונים והתפלגותם במאגרי המידע .... - 21 -
איור 7: בקשת HTTP מקורית (XSS) .... - 24 -
איור 8: בקשת XSS) HTTP) לאחר ביצוע Tokenization .... - 25 -
איור 9: הצגה חלקית של מילון tfidf עבור XSS .... - 26 -
איור 10: חישוב דמיון לבקשה מתוך הרצת הפרויקט.. - 28 -
איור 11: הצגת המונחים בעלי המשמעות הגבוהה לסיווג בבקשה.............................. - 29 -
איור 12: תוצאות המסווג במאמר .... - 30 -
איור 13:תוצאת האימון על סט הלמידה .... - 31 -
איור 14:תוצאות המסווגים על סט המבחן .... - 31 -
איור 15:תוצאת המסווג עם וללא גישה לקונטקסט .... - 31 -
איור 16:תוצאת המסווג לפי סוג ההתקפה.. - 32 -
איור 17:תוצאת המסווג עם מילון מצומצם .... - 32 -
איור 18:מונחים מובילים לכל סוג תקיפה, ממוינים לפי TF-IDF.................................. - 33 -
איור 19: ווידוא גרסת הפייתון שבשימוש.. - 36 -
איור 20: פלט ההרצה של התוכנית לקבלת עזרה... - 37 -
איור 21: זרימת התוכנית הראשית.. - 40 -
איור 22:תהליך העבודה בשלב הpreProcessDataSet.. - 41 -
איור 23:תהליך העבודה בעת חישוב ערכי הTFIDF .... - 42 -
איור 24:תהליך חישוב ה threshold בעבודה עם fold-cross .... - 43 -
איור 25:תהליך חישוב ה threshold בעבודה עם threads .... - 45 -
איור 26:תהליך חישוב תוצאת המסווג .... - 45 -
איור 27:הגדרות תוצאת הסיווג... - 46 -
איור 28:תוצאות המסווג במאמר .... - 47 -
איור 29:תוצאת המסווג שלנו על כלל ה dataset... - 47 -
איור 30:תקריב גרף תוצאת המסווג .... - 48 -
איור 31:תקריב עמוק יותר על גרף תוצאת המסווג.. - 48 -
איור 32:תוצאות המסווג בעבודה על כלל הDataset .... - 49 -
איור 33:תוצאות המסווג על כלל ה Dataset לזדוני/תקין .... - 50 -
איור 34:תוצאות המסווג בהרצת 10-Fold ללא shuffle.. - 50 -
איור 35:תוצאות המסווג בהרצת 10-Fold ללא shuffle לזדוני/תקין .... - 51 -
איור 36:תוצאות המסווג בהרצת 10-Fold ללא shuffle על XSS.................................. - 51 -
איור 37:תוצאות המסווג בהרצת 10-Fold עם shuffle .... - 52 -
איור 38:תוצאות המסווג בהרצת 10-Fold עם shuffle על XSS .... - 53 -
איור 39:תוצאות המסווג בהרצת 10-Fold עם shuffle לזדוני/תקין .... - 53 -
איור 40:המונחים המשמעותיים ביותר של כל סוג תקיפה.. - 54 -
איור 41:בקשה זדונית (XSS) שלא מתוך ה Dataset .... - 55 -
איור 42:הבקשה לאחר ביצוע טוקניזציה... - 55 -
איור 43:תוצאת המסווג על הבקשה החדשה.. - 56 -
איור 44:בקשת SQLi שאינה מהDataset.. - 56 -
איור 45:בקשת SQLi שאינה מהDataset לאחר טוקניזציה.. - 57 -
איור 46:תוצאת המסווג על בקשת הSQLi החדשה.. - 57 -

2. תקציר

פרויקט זה מממש אלגוריתם לסיווג הודעות HTTP לסוגי תקיפות שונות/תקין המוצג כחלק ממאמר [1]. מאגר המידע לסיווג הינו ECML/PKDD 2007 Discovery Challenge [2] אשר מכיל קצת יותר מ 50,000 דוגמאות.

המאגר הינו סט של בקשות מהעולם האמיתי אשר נאספו לטובת אתגר שנתי לData Mining בתחום סיווג הודעות HTTP. ההודעות מחולקות ל 7 סוגי תקיפה שונים ולסיווג תקין (Valid)

מסמך זה מציג את אופן המימוש, סוגיות אשר עלו במהלך כתיבת הפרויקט, תוצאות המסווג המפותח, וכן נקודות לעבודת המשך.

3. מבוא

3.1. מבנה המסמך

המסמך נותן רקע לעולם התוכן של הגנת מערכות ווב, וביניהם רקע לקושי ביצירת ההגנה על מערכות ווב.

המסמך מכיל רקע תיאורטי רלוונטי מעולם המושגים של הפרויקט, לטובת הגדרת מטרת הפרויקט והאתגר אותו הוא בא לפתור. (רמת פירוט בינונית – פירוט רב קיים בעבודה המסכמת)

המסמך מכיל תיאור של המאמר [1], האלגוריתם המוצע, וכן פירוט על מאגר המידע עליו המאמר מבוסס.

לאחר מכן, המסמך סוקר את הפרויקט המימושי, אפיון המימוש, סוגיות שעלו בעת הביצוע, תוצאות הפרויקט, מסקנות, ונקודות להמשך.

3.2. כללי

הגנת מערכות וביות עוסק בטכניקות אנליזת תקשורת, כאשר המטרה היא זיהוי ניסיונות חדירה, השבתה, תקיפה, זיוף נתונים או גניבתם.

מחקרים אחרונים שבוצעו הציגו כי 75% מתקיפות הסייבר מבוצעות ברמת שכבת האפליקציה!

פגיעויות במערכות וביות גדלות בצורה משמעותית, מתוך הדרישה הגוברת של חברות לשחרר גרסאות חדשות מהר יותר – על מנת לתת מענה לדרישות הלקוחות.

התפתחותה של שכבת האפליקציה הוובית אפשרה עושר אפליקטיבי ופונקציונלי היישר מתוך הדפדפן, ובכך סרוויסים רשתיים מחליפים אפליקציות "לגסי", ובכך משטח התקיפה כלפי התוקפים מתרחב.

תקיפות וביות מצליחות לנצל חולשות באפליקציה, כך שבאמצעות בקשות "לגיטימיות", ניתן להשיג מידע רגיש, להשבית את המערכת, לשבש נתונים ועוד. הבקשות לגיטימיות בכך שמבחינת מערכות הגנה סטנדרטיות כגון FW, IDS/IPS הודעות אלו עומדות בקטגורית האבטחה שהן הגדירו – הבקשות מגיעות לפורט האפליקציה הרצוי (443/80 לדוגמה), ממקור לגיטימי, ולא נראות חשודות ברמת הפקטה שעוברת (פקטה לא תקינה תקשורתית כדוגמה).

דוגמה לפגיעות נפוצה – SQL Injection – הזרקת שאילתת SQL לבקשה הנשלחת לשרת, על מנת שיריץ את שאילתא זו בשם התוקף – למטרת השגת נתונים רגישים, מחיקת נתונים ועוד. במידה והאפליקציה חשופה לפגיעות זו, מערכות הגנה סטנדרטיות לא יזהו תקיפה זו, כיוון שהתוקף פונה לשרת "בערוצים המותרים", אך לדאבונו של מפתח האפליקציה – השרת יבצע דברים שהוא לא תכנן בעת כתיבת השירות.

3.3. תיאור הבעיה

בהמשך לרקע המתאר את הבעיה בפתרונות המבוססים בחינה "per packet", הוצאו לשוק פתרונות Web Application Firewall – WAF שמטרתם לתת מענה ייעודי לבעיית ההגנה על אפליקציות וביות.

פתרונות המבוססים חוקות "סטטיות" ו/או חתימות (בדומה לאנטיוירוס) לא מספקים מענה הרמטי למגוון התקיפות והפגיעויות הקיים, ולאופן המגוון שתוקפים מוצאים לביצוע אותה תקיפה כך שהחתימות המוכרות לא יזהו זאת.

קיים פרויקט בשם OWASP [6] המפרסם בכל כמה שנים את 10 הפגיעויות שנוצלו הכי הרבה ובעלי פוטנציאל נזק גבוה על ידי תוקפי סייבר.

מתוך הדו״ח אפשר ללמוד הן על חומרת הפגיעויות והן על השוני בין התקיפות, המורכבות להגן מכל אחת ואחת מהתקיפות, וההבנה העמוקה שצריכה להיות למפתחים ולמגני הרשת על מנת להתמודד עם משטח התקיפה הנ״ל.

נקח לדוגמה פגיעות ל״הזרקה״:

הפגיעות לא רלוונטית רק למתארי SQL, אלא גם ל LDAP, ניהול מערכות קבצים, ואפילו קבלת הודעות שגיאה מפורטות (שעלולות לתת מידע שימושי להמשך התקיפה) שנובעות מתוך קלט זדוני של משתמש שלא טופל כהלכה על ידי האפליקציה.

ניצול החולשה יכול להיות שונה בין אפליקציה לאפליקציה, ולכן גם ההגנה מחולשה זו יכולה להיות שונה על מנת להיות אפקטיבית. כמו כן, צורת ההגנה הנכונה להיום יכולה להיות כבר לא נכונה מחר, כאשר מתגלות עוד חולשות או אפשרויות חדשות לניצול הפגיעויות המוכרות.

3.4. מטרת הפרויקט

תקיפות מערכות ווב הופכות נפוצות יותר, מתוחכמות יותר וגמישות יותר אל מול מערכות הגנה קיימות. יצירת חוקים סטטיים מתוך מקרי עבר לא מספק מענה מספק אל מול תוקפים אשר משנים את דרכי התקיפה שלהם, ובכך מתקיים מצב של ״חתול ועכבר״.

בעקבות כך עולה הצורך ביכולת למידה של חוקיות לגבי הודעות זדוניות והודעות לגיטימיות, כך שניתן יהיה למנוע את התקיפה הבאה שעדיין לא מוכרת (zero day attack). פתרונות כאלו לרוב ממומשות באמצעות למידת מכונה, אך לא תמיד ברורה שיטת הלמידה או הפיצ׳רים שנמצאו ברשת לטובת הסיווג.

מטרת הפרויקט הינה מימוש אחת מהשיטות שתוארה באחד מהמאמרים שנבחרו לטובת העבודה המסכמת. השיטה מתבססת על עקרונות למידה פשוטים מתוך סט הנתונים, כך שברור למנתח כיצד הנתונים הוסקו על ידי השיטה.

המאמר בוחן את הצלחת השיטה המוצעת אל מול סט מידע מוכר וידוע בשם ECML/PKDD 2007 Discovery Challenge [2], אשר מכיל הודעות לדוגמה המסווגות כזדוניות או לגיטימיות בהתאם.

3.5. חשיבות הפרויקט

עבודה זו מאפשרת כניסה לעולמות למידת המכונה והגנה על מערכות וביות. באמצעות העבודה, ניתן להבין את מורכבות עולם ההגנה, את הכלים איתם מנסים כלל החוקרים לביצוע הגנות יעילות וגמישות, ואת עתיד הפיתוחים בתחום זה.

בסיום עבודה זו, ניתן לחשוב על שילוב של טכניקות שונות משכבות הגנה שונות על מנת לתת מענה מקיף יותר, והן ללמוד את נקודות החוזקה והחולשה של כל טכניקה, על מנת להבין לאילו פערים יש לתת מענה במסגרת מחקרים מתקדמים בתחום.

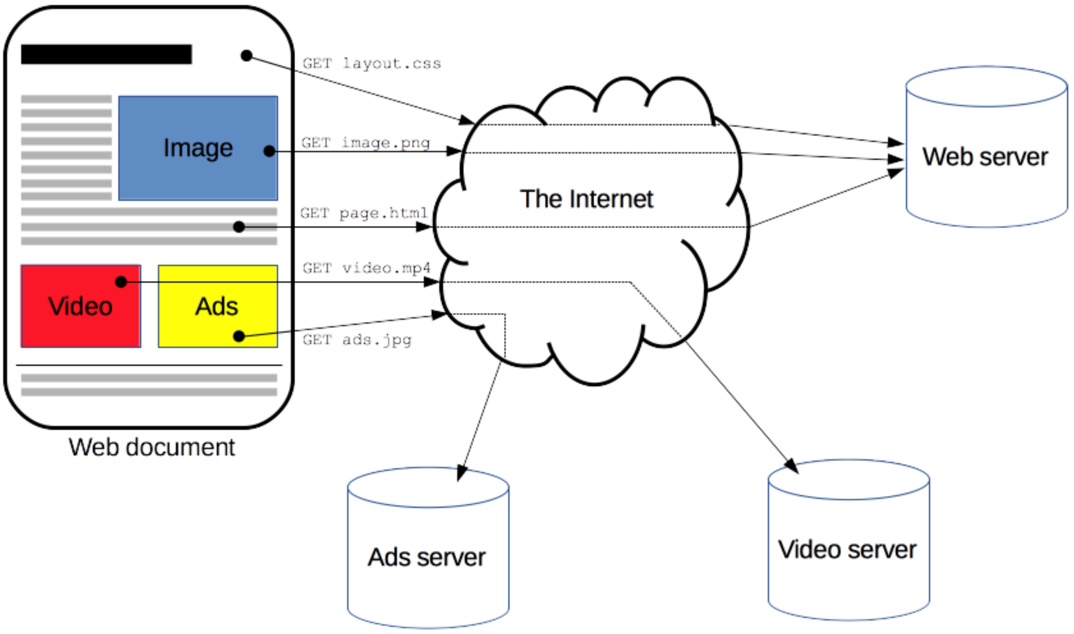
4. רקע – מונחים עיקריים

4.1. פרוטוקול HTTP ([7])

פרוטוקול HTTP- HyperText Transfer Protocol נוצר לטובת העברת היפר-מדיה (hypermedia) – כגון תמונות, וידאו, טקסט וקבצי HTML (HyperText Markup Language), בין מערכות מבוזרות על גבי רשת האינטרנט.

הפרוטוקול עובד בתצורת client-server, כלומר הבקשות נוצרות על ידי הלקוח – לרוב דפדפן אינטרנט.

אתר אינטרנט שלם מורכב מכמה תתי מסמכים – לדוגמה טקסט, עיצוב, תמונות, וידאו, סקריפטים וכו'.



איור 1: שימוש ב HTTP לטובת טעינת אתר

ברמה התקשורתית, פרוטוקול הHTTP ממומש בשכבת האפליקציה, ורוכב על TCP כ Transport Layer.

הפרוטוקול בנוי מכמה חלקים עיקריים : (גרסת 1.0 ומעלה של הפרוטוקול)

- מתודה לפעולה (Method) – הפעולה אותה מבקש הלקוח (הדפדפן) מהשרת לבצע. בין המתודות האפשריות : Get – קבלת משאב כלשהו, POST – העלאת משאב לשרת, DELETE – מחיקת משאב, ועוד
- URL – הנתיב אליו פונים בשרת הווה לטובת הפעולה. בשילוב עם המתודה מתקבלת בקשה מוגדרת, כגון קבלת המשאב הנמצא בנתיב X, או העלאת משאב לשרת לנתיב X בשרת.

- Query – חלק מהURL, שתפקידו הצבת ערכים במשתנים לטובת הבקשה שעוברת.
הQuery מופרד מהנתיב עצמו באמצעות סימן שאלה
לדוגמה **https://example.com/over/there?name=ferret** , הquery דואג להציב את הערך ferret במשתנה name.
- Headers – מאפשרים ללקוח/שרת להוסיף מידע נלווה לבקשה/תשובה. Header מורכב משם (case-insensitive) המופרד באמצעות נקודתיים (:) מהערך שלו.
ניתן לחלק את ההאדרים לכמה קבוצות לפי הקונטקסט שלהם:
  - האדרים כללים – מתאימים הן לבקשות והן לתשובות, ללא תלות בתוכן ההודעה שעובר
  - האדרים לבקשות – מכילים אינפורמציה נוספת לגבי המשאב המבוקש, או לגבי הלקוח המבקש את הבקשה
  - האדרים לתשובות – מכילים אינפורמציה נוספת לגבי המשאב החוזר כגון המיקום שלו, או אינפורמציה על השרת המחזיר את התשובה
  - האדרים לתוכן (Entity Headers) – מכילים אינפורמציה לגבי התוכן (הData) שעובר – כגון אורכו (Content-Length) , סוגו (Content-Type) ועוד.
- Body – תוכן ההודעה – הData אותו רוצים לשלוח לשרת / השרת מעוניין להחזיר ללקוח.
הBody מופרד באמצעות 2*CRLF מהHeader Section, ובכך הלקוח והשרת יודעים לפרסר אותו.
תוכן הBody אינו תלוי בHTTP והוא יכול להיות מגוון – דף HTML, תמונה, וידאו, קובץ טקסט, מלל חופשי ועוד.

4.2. XSS – Cross Site Scripting ([3],[4],[5],[9],[10])

תקיפת XSS הינה תקיפה מבוססת הזרקה (Injection), שבה סקריפט זדוני מוזרק אל אתר לגיטימי ואמין. התקיפה מתרחשת כאשר תוקף משתמש באפליקציית ווב על מנת לשלוח קוד זדוני, לרוב סקריפט צד לקוח, למשתמש קצה אחר – הקורבן.

חולשות המאפשרות תקיפות כאלו הינן נרחבות ומגוונות, ויכולות לקרות בכל מקום בו האפליקציה מקבלת קלט משתמש כחלק מהתוצר אותה היא מעבדת, ללא וואלידציה על גבי הקלט / ביצוע קידוד מקדים.

תוקף יכול להשתמש בתקיפת XSS על מנת לשלוח סקריפט זדוני למשתמש תמים. דפדפן המשתמש אינו יכול לדעת שהסקריפט אותו הוא עומד להריץ אינו מהימן, ולכן הוא יריץ את הסקריפט בשם הקורבן. כיוון שהסקריפט מורץ על ידי הדפדפן של הלקוח, יש לו גישה למרכיבים רגישים כגון הcookies, session tokens או אינפורמציה רגישה שנגישה לדפדפן עבור אותו אתר.

בעבר, כאשר רק התחילו להשתמש בשפת JavaScript, לקוד הרץ הייתה גישה <u>לכלל החלונות הפתוחים של הדפדפן</u>, ולכן אתר/קוד זדוני יכול לגנוב מידע של משתמש בהקשר לאתרים אחרים – כגון בנקים, מיילים וכד'. על מנת לתקן בעיה זו, פותחה מדיניות "אותו המקור" (Same Origin Policy), שבמסגרתה קוד שנשלח על ידי שרת ווב יכול לגשת רק למידע שמקורו באתרים מאותו שם תחום (Domain) ועל גבי אותו פרוטוקול (HTTP/S, Websocket/S)

הסקריפטים יכולים אפילו לשכתב את התוכן של עמוד הHTML, ובכך להפוך למנגנון יעיל לתקיפות phishing, או לבצע את הרצת הקוד כחלק מעבודת הדפדפן על הHTML המתקבל.

4.2.1. סוגי XSS

ניתן לחלק את תקיפות ה XSS ל2 קטגוריות: Stored (קבוע) ו reflected (זמנית).

קיים סוג שלישי פחות מוכר בשם DOM Based XSS שלא נכנס אליו במסגרת הפרויקט.

4.2.1.1. Stored XSS (XSS קבוע)

תקיפות מסוג זה הינן תקיפות שהסקריפט המוזרק נשמר באופן קבוע בשרת היעד, לדוגמה בDB שלו, בפורום הודעות, בלוג המבקרים, בהערות ועוד. הקורבן מקבל את הסקריפט הזדוני כאשר הוא מבקש את המשאב בשרת המכיל את הסקריפט הזדוני – לדוגמה כאשר הוא ניגש לכתבה בה הסקריפט נשמר כתגובה לכתבה.

תקיפת Stored XSS נקראת גם Persistent או Type-I XSS.

4.2.1.2. Reflected XSS (XSS זמני)

בתקיפות מסוג זה, הסקריפט המוזרק משתקף/מוחזר מהשרת – לרוב כהודעת שגיאה, תוצאת חיפוש, או כל תשובה שמכילה את קלט המשתמש (או חלקו) שהגיע בבקשה.

תקיפות אלו מגיעות לקורבן לרוב במסלול חיצוני – באמצעות אימייל, תגובה באתר אחר וכו'. כאשר הקורבן לוחץ על הלינק הזדוני, הוא מבצע בקשה מיוחדת לשרת שנבנתה על ידי התוקף, או מבצע גלישה לאתר זדוני ברשות התוקף. הקוד המוזרק חוזר בתשובה ללקוח, ודפדפן הלקוח מריץ את הקוד הזדוני – כיוון שהתקבל מהשרת, הקוד נחשב מהימן ולכן ירוץ.

תקיפת Reflected XSS נקראת גם Non-Persistent או Type-II XSS

4.3. Sql Injection ([5],[11])

תקיפות SQLi הינן תקיפות המזריקות שאילתת SQL כחלק מקלט משתמש לאפליקציה. תקיפת SQL מוצלחת יכולה לקרוא מידע רגיש מהDatabase, לשנות מידע (Insert/Update/Delete), לבצע פעולות אדמיניסטרטיביות על הDatabase (כגון כיבוי השירות), ולפעמים אף להריץ פקודות על מערכת ההפעלה המריצה את השירות.

הזרקת SQL קורית כאשר מידע נכנס לאפליקציה ממקור לא מהימן (כגון קלט משתמש), והמידע משמש ליצירת שאילתת SQL בצורה דינמית.

השפעות תקיפת SQL:

- **סודיות** – מאגר נתונים מסוג SQL לרוב מכיל מידע רגיש. הרצת שאילתות זדוניות על גבי המאגר פוגעת בסודיות המידע השמור במאגר.

- **הזדהות** – אם שאילתות SQL לא אבטחתיות משמשות לטובת בדיקת שם משתמש וסיסמה כהזדהות לאפליקציה, תקיפה מוצלחת יכולה לאפשר לתוקף גישה מורשית ללא ידע מקדים של הסיסמה.
- **הרשאות** – אם ניהול ההרשאות מנוהל כחלק ממאגר הנתונים, ניתן באמצעות תקיפה מוצלחת לשנות את המידע השמור ובכך להשפיע על הרשאות המשתמשים השמורים.
- **אמינות** – כפי שניתן לקרוא מידע רגיש מהמאגר, באמצעות תקיפה מוצלחת ניתן לשנות או למחוק מידע מהמאגר.

בדוגמה שלפנינו מוגדר קוד C# אשר יוצר בצורה דינמית שאילתת SQL ומריץ אותה לטובת חיפוש פריטים השייכים לאדם מסוים. השאילתא רוצה להגביל את התוצאות לפריטים השייכים לאדם המסויים.

```
...
string userName = ctx.getAuthenticatedUserName();
string query = "SELECT * FROM items WHERE owner = "'"
                + userName + "' AND itemname = '"
                + ItemName.Text + "'";
sda = new SqlDataAdapter(query, conn);
DataTable dt = new DataTable();
sda.Fill(dt);
...
```

השאילתא אותה הקוד מנסה להריץ:

```
SELECT * FROM items
WHERE owner =
AND itemname = ;
```

נשים לב שהשאילתא נוצרת בצורה דינמית באמצעות שילוב מחרוזת קבועה יחד עם קלט משתמש. השאילתא עובדת כנדרש רק כאשר itemName לא מכיל את התו גרש ('). אם תוקף עם משתמש בשם wiley מכניס את הקלט `itemName` for `"name' OR 'a'='a"` נקבל את השאילתא הבאה:

```
SELECT * FROM items
WHERE owner = 'wiley'
AND itemname = 'name' OR 'a'='a';
```

התוספת של OR 'a'='a' לשאילתא מייצרת בדיקת תנאי כחלק מההרצה. כיוון שהתנאי הנ"ל תמיד נכון, והתנאי מוכל כחלק מכל הגבלת ה Where מתקבלת שאילתא מקבילה פשוטה מאוד:

```
SELECT * FROM items;
```

התוקף הצליח במצב זה לקבל את כל מאגר הפריטים, ולא רק של המשתמש המחובר.

4.4. LDAP Injection ([12])

הזרקת LDAP הינה הזרקת קוד שמטרתה לנצל חולשת אבטחה באפליקציה באמצעות קלט פרמטרים שעובר לטובת חיפוש פנימי, או בפעולות הוספה/שינוי באפליקציה. כאשר האפליקציה לא מבצעת סניטציה או ווידוא טוב של קלט המשתמש, תוקף יכול לנצל זאת לטובת עריכת ה LDAP Statement.

ההזרקה מבוצעת כאשר קלט המשתמש מוכנס ע״י האפליקציה כחלק מיצירת פילטר LDAP בצורה דינמית, וקלט המשתמש לא עובר ואלידציה או סניטציה ראויה. התוצאה היא מניפולציה על הLDAP Statement שמבוצע על ידי שרת ה LDAP, כך שהתוקף יכול לראות, לערוך ואף לעקוף מנגנוני הזדהות.

בדוגמה הבאה מוגדרת שאילתא שמטרתה לבדוק את פרטי ההזדהות של המשתמש למטרת התחברות למערכת:

```
String filter = "(&(USER = " + user_name + ") (PASSWORD = " + 
user_password + "))";
```

בשימוש תמים של השאילתא, המשתמש מכניס את הפרטים שלו – שם המשתמש והסיסמה, ואכן השאילתא תבצע את מטרתה. עם זאת, תוקף יכול להכניס קלט משתמש זדוני עבור הפרמטר user_name כגון `johnDoe)(&)` ועם כל ערך סיסמה שרירותי שיכניס, השאילתא הסופית תהיה:

`(&(USER = johnDoe)(&))(PASSWORD = pass))`

כיוון שרק החלק הראשון בשאילתא `(&(USER = johnDoe)(&)` מעובד על ידי שרת ה LDAP, ושאילתא זו תמיד מחזירה "אמת", תוצאת השאילתא תאפשר לתוקף לקבל אישור התחברות לאפליקציה ללא ידיעת הסיסמה.

4.5. XPath Injection ([13])

בדומה להזרקת SQL, הזרקת XPath קורית כאשר אפליקציית ווב משתמש בקלט משתמש על מנת לייצר שאילתת XPath עבור מידע מסוג XML. ע״י שליחת הודעות זדוניות, התוקף יכול לקבל מידע לגבי מבנה הXML, או להשיג גישה למידע שהוא לא אמור לגשת אליו. במידה ומשתמשים בXML לטובת שמירת פרטי הזדהות, תקיפה מוצלחת יכולה לאפשר לתוקף גישה בלתי מורשת לאפליקציה.

תשאול מידע בXML מבוצע באמצעות XPath – יצירת הגדרות פשוטות המאפשרות לתשאל את הXML לקבלת חלקי מידע ממנו. בדומה לSQL, ניתן להגדיר פרמטרים ספציפיים לתשאול, הגדרת תבניות למציאה וכו׳.

XPATH זו שפה סטנדרטית, כך שהנוטציות והסינטקס שלה אינם תלויים במימוש – מה שמאפשר לתקיפות להיות אוטומטיות. אין תלות במימוש ספציפי, בניגוד לשאילתות SQL. בנוסף, אין הגדרת רמות גישה כמו שקיימים בSQL, ולכן ניתן להשיג את כל המסמך בתקיפה מוצלחת, כפי שנראה בדוגמה הבאה:

```
<?xml version="1.0" encoding="utf-8"?>
<Employees>
   <Employee ID="1">
      <FirstName>Arnold</FirstName>
      <LastName>Baker</LastName>
      <UserName>ABaker</UserName>
      <Password>SoSecret</Password>
      <Type>Admin</Type>
   </Employee>
   <Employee ID="2">
      <FirstName>Peter</FirstName>
      <LastName>Pan</LastName>
      <UserName>PPan</UserName>
      <Password>NotTelling</Password>
      <Type>User</Type>
   </Employee>
</Employees>
```

נניח שלאתר יש מערכת התחברות משתמשים שמשתמשת בסוג זה של קובץ לטובת ניהול המשתמשים. כאשר שם משתמש וסיסמה מוזנים במערכת, התוכנה יכולה להשתמש בXPATH בתצורה הבאה לטובת חיפוש המשתמש:

```
VB:
Dim FindUserXPath as String
FindUserXPath = "//Employee[UserName/text()='" &
Request("Username") & "' And
        Password/text()='" & Request("Password") & "']"

C#:
String FindUserXPath;
FindUserXPath = "//Employee[UserName/text()='" +
Request("Username") + "' And
        Password/text()='" + Request("Password") + "']";
```

כאשר מזינים שם משתמש וסיסמה בצורה תמימה, המערכת אכן תעבוד. אך תוקף יכול לנצל את הקוד הנ"ל באמצעות שליחת שם משתמש וסיסמה זדוניים לקבלת מידע מהXML שהוא לא אמור לקבל:

```
Username: blah' or 1=1 or 'a'='a
Password: blah

FindUserXPath becomes //Employee[UserName/text()='blah' or 1=1 or
        'a'='a' And Password/text()='blah']

Logically this is equivalent to:
        //Employee[(UserName/text()='blah' or 1=1) or
        ('a'='a' And Password/text()='blah')]
```

במקרה זה, מספיק שהחלק הראשון של השאילתא יהיה נכון, ולכן הסיסמה שהוזנה הופכת ללא רלוונטית. בנוסף, החלק של UserName מקבל גם "אמת" בגלל החלק של 1=1, ולכן השאילתא תתפוס לגבי כל המשתמשים במערכת.

התקיפה אפשרה לתוקף להשיג את כלל המשתמשים והסיסמאות השמורים במערכת.

כאשר משתמשים בXML באתר, נפוץ להשתמש באיזשהו קלט משתמש לתשאול, כדי לזהות איזה חלק יש להציג על האתר. על מנת להבטיח שלא מנצלים את הבקשות לרעה, יש לבצע סניטציה לקלט המשתמש, כך שלא ניתן יהיה להשיג את המידע שהמפתח לא התכוון אליו.

4.6. Path Traversal ([14])

תקיפה זו ידועה גם בשמות "dot-dot-slash", "directory traversal", "directory climbing", "backtracking".

מטרת תקיפה זו היא השגת גישה לקבצים ותיקיות השמורים מחוץ לנתיב הבסיס של שרת הווב. באמצעות מניפולציה על המשתנים המפנים לקבצים, עם רצפים כמו "נקודה-נקודה-סלאש" (../) ודומיו, או באמצעות שימוש בנתיב אבסלוטי קיימת אפשרות להשיג גישה לקבצים ותיקיות שרירותיים השמורים על מערכת הקבצים – כולל קבצי מקור, קונפיגורציה או קבצי מערכת קריטיים.

- חשוב להגדיש כי הגישה לקבצים אלו מוגבלת על ידי מערכת הרשאות הקבצים המנוהלת על ידי מערכת ההפעלה, ופעולות אלו ניתנות במסגרת ההגבלות הקיימות במערכת ההפעלה לקבצים המיועדים על ידי התוקף

כלל אפליקציות הווב נדרשות לכלול משאבים מקומיים כחלק מריצתם, כגון – תמונות, סקריפטים, עיצובים וכו'. בכל פעם שהאפליקציה נדרשת לטעון משאב או קובץ, קיים חשש שתוקף ינסה לטעון קובץ או משאב מרוחק שכותב האפליקציה לא התכוון שייחשף.

ישנם מספר ואריאציות ל"בריחה" מהScope של התיקייה הנוכחית, ולכן החשיבות לטיפול בקלט משתמש בצורה נכונה הינה קריטית, ופעולות blacklist אינן מספיקות. דוגמאות לקידוד וקידוד כפול:

`../` represents `%2e%2e%2f`

`../` represents `%2e%2e/`

`../` represents `..%2f`

`..\` represents `%2e%2e%5c`

`..\` represents `%2e%2e\`

`..\` represents `..%5c`

`..\` represents `%252e%252e%255c`

`..\` represents `..%255c`

כמו כן יש קידודי URL (Url Encoding):

`..%c0%af` represents `../`

`..%c1%9c` represents `..\`

ישנה גם חשיבות למערכת ההפעלה שמטפלת בבקשות, שכן המפריד בין התיקיות, ותיקיית השורש מטופלות שונה:

```
UNIX
Root directory:  " / "
Directory separator: " / "

WINDOWS
Root directory: "  <partition letter> : \ "
Directory separator: " / " or " \ "
Note that windows allows filenames to be followed by extra . \ /
characters.
```

ישנם מערכות הפעלה שניתן להכניס %00 null bytes לטובת טרמינצית שם הקובץ. ניקח לדוגמה שליחת פרמטר שתפקידו להכיל את שם הקובץ המבוקש:

`?file=secret.doc%00.pdf`

התוצאה משליחה כזו היא שאפליקציית JAVA רואה את המחרוזת הזו מסתיימת ב pdf. בעוד שמערכת ההפעלה רואה קובץ שמסתיים ב doc. תוקפים יכולים להשתמש בטריק זה על מנת לעקוף מנגנוני אכיפה, לדוגמה כאלו שמגדירים העלאת קבצים מסוג מסוים.

4.7. OS Commanding ([15])

ידוע גם בשם Command Injection. מטרת תקיפה זו היא הרצת פקודות על מערכת ההפעלה המארחת של שרת הווב, באמצעות ניצול חולשה באפליקציה. הזרקת פקודות אפשרית כאשר האפליקציה מעבירה קלט משתמש שלא עבר ווידוא לShell של מערכת ההפעלה. המידע יכול להגיע מטפסים שנשלחים על ידי המשתמש, cookies, האדרים בHTTP ועוד.

הפקודות המוזרקות רצות על ידי השרת הפגיע, ולכן הרשאות הריצה של הפקודות הן כשל השרת עצמו.

בניגוד להזרקת קוד, בו הקוד המוזרק מורץ על ידי האפליקציה, הזרקת פקודות משתמשות בפונקציונליות קיימת של האפליקציה – הפעלת פקודות מערכת ההפעלה, למטרות שונות מכוונת המפתח.

הקוד הבא רץ בסביבת UNIX, והוא עוטף את הפקודה cat שמטרתה הדפסת התוכן של קובץ לstandard output.

```
#include <stdio.h>
```

```
#include <unistd.h>

int main(int argc, char **argv) {
 char cat[] = "cat ";
 char *command;
 size_t commandLength;

 commandLength = strlen(cat) + strlen(argv[1]) + 1;
 command = (char *) malloc(commandLength);
 strncpy(command, cat, commandLength);
 strncat(command, argv[1], (commandLength - strlen(cat)) );

 system(command);
 return (0);
}
```

כאשר משתמשים בקוד בצורה נאיבית, אכן רק מודפס התוכן של הקובץ המבוקש:

```
$ ./catWrapper Story.txt
When last we left our heroes...
```

אם נוסיף semicolon (;) ולאחריו עוד פקודה, הפקודה תבוצע על ידי התוכנית ללא בעיה:

```
$ ./catWrapper "Story.txt; ls"
When last we left our heroes...
Story.txt               doubFree.c              nullpointer.c
unstosig.c              www*                    a.out*
format.c                strlen.c                useFree*
catWrapper*             misnull.c               strlength.c
useFree.c
commandinjection.c      nodefault.c             trunc.c
writeWhatWhere.c
```

במקרה הזה ביקשנו לפרוט את הקבצים הקיימים בתיקייה. במידה ולאפליקציה הרשאות גבוהות, תוקף יכול להשתמש בחולשה זו להרצת פקודות בהרשאות גבוהות על השרת.

4.8. Server Side Include – SSI ([16], [18])

SSI הינם דירקטיבות הקיימות באפליקציית ווב שמטרתן הזנת דף HTML עם תוכן דינמי. SSI דומה לCGI (Common Gateway Interface), אך משתמשים בSSI בשביל להריץ מספר פעולות לפני שהדף הנוכחי נטען, או בזמן שהדף מוצג ללקוח. על מנת לאפשר זאת, שרת הווב מעבד את הSSI לפני שהוא מספק את הדף ללקוח.

תקיפות SSI מבוצעות על ידי הזרקת סקריפט בדפי הHTML או בהרצת קוד מרוחק. הניצול יכול להיות מבוצע על ידי מניפולציה על אופן השימוש בSSI על ידי האפליקציה, או הכרחת האפליקציה להשתמש בSSI באמצעות קלט משתמש זדוני.

התקיפה תצליח אם השרת מאפשר הרצת SSI מבלי לבצע וואלידציה מספקת על הקלט. התקיפה יכולה להוביל למניפולציה על מערכת הקבצים שהשרת רץ בה, תחת ההרשאות הקיימות לשרת עצמו במערכת ההפעלה.

בעת ניצול החולשה, תוקף יכול לקרוא מידע רגיש כגון סיסמאות, ואף להריץ פקודות Shell. הנחיות הSSI מוזרקות כחלק מקלט המשתמש, והן נשלחות לשרת. השרת מפרסר ומבצע את ההנחיות לפני החזרת התשובה ללקוח. תוצאות התקיפה נגישות בפעם הבאה שהעמוד נטען בדפדפן הלקוח.

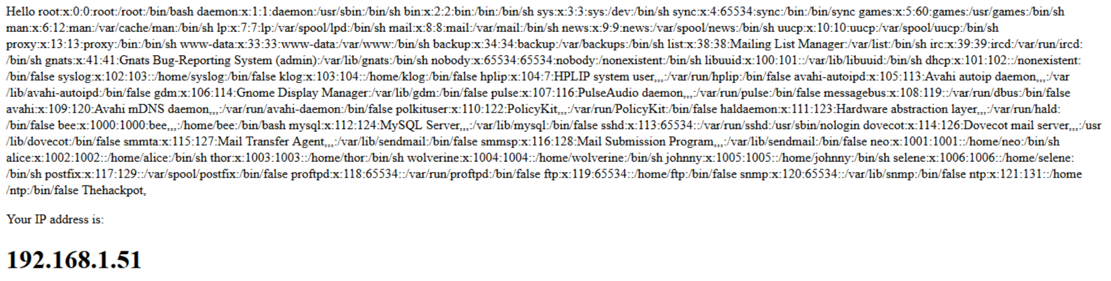
SSI מפורסר כחלק מהHTML בדף, ולכן תנאי מקדים לתקיפה היא יכולת הזרקת HTML לדף המיוצר בשרת באמצעות קלט המשתמש (הזרקת ה SSI בקלט המשתמש).

ניקח לדוגמה עמוד בשרת המקבל כקלט משתמש שם פרטי ושם משפחה, ומחזיר בתשובה דף עם השם שהתקבל + כתובת הIP איתה המשתמש התחבר:

איור 2: דף פגיע להזרקת SSI

השרת מכניס את קלט המשתמש לHTML שחוזר בדף התשובה השרת אינו מבצע בדיקות קלט, ובדוגמה זו, השרת תומך בSSI. לכן אם במקום להזין את השם, נזריק את הקלט הבא:

`<!--#exec cmd=”cat /etc/passwd -->`, השרת יריץ את הפקודה הכתובה, ובעצם יחשוף את את קובץ הסיסמאות השמורות בשרת:



איור 3: תוצאת הרצת הSSI הזדוני

התווים שאיתם משתמשים לרוב בהגדרת SSI הינם:

```
< ! # = / . " - > and [a-zA-Z0-9]
```

כאשר מבצעים סניטציה לקלט המשתמש יש לשים לב שמבצעים בדיקות על בעלי המשמעות המיוחדת בעולם הSSI.

‎4.9. Vector Space Model ‏([8])

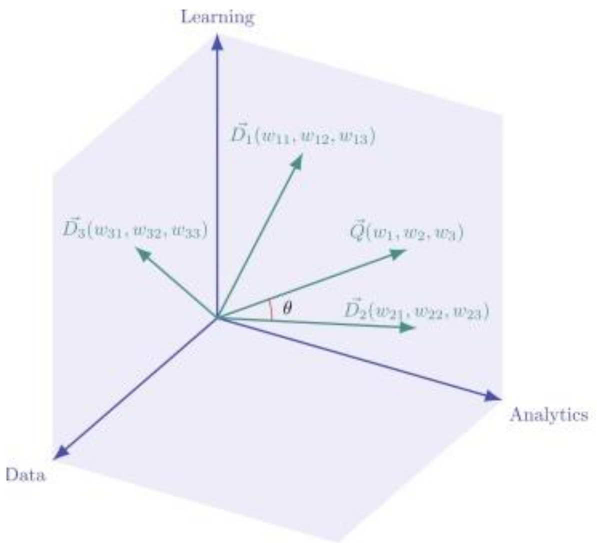
המודל המתמטי Vector Space Model (ידוע גם כ Term Vector Model) הינו מודל לייצוג מסמכים טקסטואליים כווקטורים מייצגים.

המודל מתבסס על הנוטציה של "דמיון" (similarity). המודל מניח שהרלוונטיות של מסמך לשאילתא היא יחסית זהה לדמיון בין המסמך לשאילתא.

המסמכים והשאילתות מיוצגים באמצעות מודל "bag-of-words". עבור אוסף מסמכים (Corpus) מגדירים קודם סט מונחים (המילון) ומייצרים סדר למונחים.

המסמכים מיוצגים על ידי ווקטור n-מימדי, כאשר כל מימד מקושר למונח. המונחים מקבלים משקלים באמצעות שיטות tf-idf (*מבוצע במאמר) או BM25.

באיור [4] ניתן לראות ווקטור 3-מימדי המורכב מ3 מונחים : Data, Analytics, and Learning.



איור 4: הצגת מסמכים ושאילתות כווקטורים במרחב

מוגדרים 3 מסמכים – D_1,D_2,D_3 ולכל אחד מהם מוגדר ווקטור $\vec{D}_1, \vec{D}_2, \vec{D}_3$. משקלי המונחים עבור $\vec{D}_1$ הינם (w_{11},w_{12},w_{13}). משקלי המונחים קובעים את מיקום/אוריינטציית המסמך במרחב הווקטורי. שאילתות מיוצגות גם כן באמצעות ווקטור 3-מימדי $\vec{Q}(w_1, w_2, w_3)$, ומשקלי השאילתא מקבלים משקל גם כן.

ווקטורי המסמך והשאילתות עוברים נרמול לווקטורי יחידה. הווקטור עובר נרמול זה באמצעות חלוקת הווקטור על ידי הגודל הווקטורי שלו : $\vec{d}_1 = \dfrac{\vec{D}_1}{\sqrt{w_{11}^2 + w_{12}^2 + w_{13}^2}}$

הדמיון (similarity) בין מסמך לשאילתא ניתן על ידי ביצוע ה "dot product" בין ווקטורי היחידה שלהם: $sim(D_2,Q)=\vec{d}_2 \bullet \vec{q}$. תוצר הנקודה הוא הסכום של $w_{21}\cdot w_1+w_{22}\cdot w_2+w_{23}\cdot w_3$.

ניתן לחשב את $sim(D_2,Q)$ גם באמצעות הcosine של הזווית בין $\vec{d}_2$ ל $\vec{q}$.

בשביל לדרג את המסמכים באוסף ביחס לשאילתא, מחשבים את קוסינוס הזווית בין השאילתא ובין כל אחד מהמסמכים – ובכך מקבלים Relevance Status Value (RSV) לכל מסמך-שאילתא.

VSM הוא בעצם framework שכן הוא לא מגדיר כיצד מבצעים כל אחד ואחד מהשלבים והחישובים – ביצוע tf-idf עם cosine similarity זו דוגמה אחת, והיא זו שמבוצעת במאמר.

5. ניתוח המאמר ([1])

5.1. הקדמה

הפרויקט מתבסס בעיקרו על מאמר [1], אשר פותח בהצגת המורכבות הגדלה באפליקציות ווב, ובכך בהגדלת משטח התקיפה עליהם מעצם השימוש בעוד טכנולוגיות, יכולות וממשקי משתמש.

תקיפת ווב הם מטרות מועדפות על ידי תוקפי סייבר, שכן באמצעותן ניתן להשיג פרטי משתמש, כרטיסי אשראי, ביצוע הונאות, השבתת מערכות ועוד, וכלל האירועים הללו מתורגמים בסוף היום לסכומי כסף גבוהים, מטרות עסקיות / פוליטיות / אישיות ועוד.

על מנת לבצע את אותן תקיפות, התוקפים מזריקים קוד זדוני לבקשות הHTTP על מנת להריץ קוד על השרת. הקוד מטרתו לבצע מגוון פעולות על השרת ,בהתאם לצורך התוקף - קריאת מידע רגיש, הריסת מאגרי מידע, ביצוע הונאות וכו׳.

הדרך שבה המאמר מנסה למנוע את התקיפות היא זיהוי הקוד הזדוני בבקשת הHTTP לפני שהיא מגיעה לשרת, ולהסיר את הבקשות הזדוניות, כך שהשרת לא יטפל בהן.

התצורה שבה המאמר מבצע את הזיהוי הוא על ידי בניית מסווג (Classifier) המשתמש בכלים מעולם הMachine Learning על מנת לסווג את ההודעות ל״תקיפה״ (Attack) או ״תקין״ (Valid).

המסווג מתבסס על Vector Space Model הנפוץ בעולם כריית המידע, לטובת סיווג הבקשות – סיווג לפי סוג התקיפה ממש (כפי שמוגדר ב Dataset), ולא רק סיווג תקין/זדוני.

המאמר מבסס את תוצאות המסווג על מאגר מידע ECML/PKDD 2007 Discovery Challenge [2] המכיל כ 50K הודעות, מסווגות לפי סוגי תקיפות/תקין.

המאמר מציג דוגמה לבקשת HTTP המכילה תקיפת SQL Injection – ראה איור [5]

איור 5: תקיפת Sql Injection בבקשת HTTP

ההאדר "Cookie" מכיל שאילתת SQL מוחבאת כחלק מהתוכן הלגיטימי של ההאדר. בעת ביצוע השאילתא על ידי שרת הSQL, השרת יקרא מידע מהDB, יכתוב את תוכנו לקובץ שנגיש ציבורית מהרשת, כך שהתוקף יוכל לגשת אליו לאחר מכן.

5.2. מטרות

מאגר המידע ECML/PKDD 2007 Discovery Challenge הינו מאגר שפורסם כחלק מתחרות כריית מידע.

תחרות כריית המידע הגדירה 2 מטרות בהקשר לזיהוי תקיפות HTTP :

1. **סיווג בקשות הHTTP** – סיווג ההודעות לסוגי התקיפות. יש חשיבות הן לתוצאות המסווג והן לתוצאות זמן הריצה של המסווג.

2.**אבחון מזהה התקיפה** – זיהוי המחרוזת הקצרה ביותר (בתוך התקיפה) שמגדירה את סוג התקיפה.

מאמר זה ממקד את העשייה במטרה הראשונה, אך בהסתכלות על משקלי המסווג ניתן להשיג את האינפורמציה המגדירה את התקיפה הכללית עצמה (ולא פר בקשה)

5.2.1. מבנה מאגר הנתונים

באתגר ניתנו 2 מאגרי עבודה - סט ללמידה, וסט למבחן.

כל מאגר מכיל את הטקסט המלא של בקשות הHTTP, מחולקות לקטגוריות הבאות : method, protocol, uri, query, headers, body. בנוסף, כל בקשת HTTP מכילה אינפורמציה נוספת לגבי הקונטקסטים הבאים :

- מערכת ההפעלה שעליה רץ שרת הווב – UNIX, WINDOWS, UNKOWN
- סוג שרת הווב שמיועד לקבל את הבקשה –APACHE,MISS,UNKOWN
- האם טכנולוגית הXPATH מובנת על ידי השרת – TRUE,FALSE,UNKOWN
- האם קיים DB של LDAP בשרת - TRUE,FALSE,UNKOWN
- האם קיים DB של SQL בשרת - TRUE,FALSE,UNKOWN

לבסוף, כל שאילתא קיבלה סיווג לאחד מ8 סיווגים : תקין או תקיפה ספציפית.

	Training Set	Test Set
Total Requests	50,116	70,143
Valid Requests	35,006 (70%)	42,006 (60%)
Attacks	15,110 (30%)	28,137 (40%)
Cross-Site Scripting	12%	11%
SQL Injection	17%	18%
LDAP Injection	15%	16%
XPATH Injecction	15%	16%
Path traversal	20%	18%
Command execution	23%	23%
SSI attacks	13%	12%

איור 6: הסיווגים השונים והתפלגותם במאגרי המידע

5.3. מודל המסווג

המאמר משתמש בVector Space Model (VSM) כדרך אפקטיבית לאנדקס מידע ולהשוות מסמכי טקסט לטובת כריית מידע. במודל זה, מסמך טקסט מיוצג על ידי ווקטור מונחים בעל משקל, כאשר מונח עם משקל גבוה יותר מייצג חשיבות גבוהה יותר למסמך.

למרות שבקשות HTTP בעלות מבנה להודעה – ניתן לחלק לשורת הבקשה (Method + URL + Protocol), האדרים ו Body, הרעיון המרכזי בגישת המאמר היא להתעלם מהמבנה, ולהתייחס לכל בקשה כטקסט חסר מבנה. גישה זו מאפשרת להתייחס לבקשת HTTP כבעיית כריית מידע ממסמך טקסט.

5.3.1. שלב הלמידה

אלגוריתם הלמידה מוגדר בתצורה הבאה:

1. עיבוד מקדים על ה data-set (המתקבל כ xml גדול):
   a. חלוקת ה data-set ל 8 קבצים שונים, לפי הסיווג של הsample – כלומר קובץ לכל סוג תקיפה (כולל valid)
   b. מכל קובץ מגדירים "מסמך" ב"אוסף המסמכים" – מסמך עבור כל סוג תקיפה (כולל Valid). כל מסמך מכיל את כלל הבקשות השייכות לסוג התקיפה. הבקשה שמורה כטקסט בלבד של ה HTTP Request, כלומר ללא המטה-דאטה של הבקשה כגון הקונטקסט שלו.
   c. יצירת אוסף tokens עבור כל מסמך, כך שכל token נוצר/הופרד באמצעות – רווח (whitespace), התו '+', ותווים שעברו url encode (כגון %20). כל token הינו term לשלב הבא.
2. חישוב ערכי tfidf:
   a. עבור כל מסמך, מחשבים את ערך ה tfidf של כל term (token משלב 1.c) במסמך שלו.
   b. את הערך המתקבל שומרים במילון (dictionary) עבור כל מסמך, כך שימפה בין הterm/token, לבין ערך ה tfidf שלו במסמך.
   c. משקל tf-idf מורכב מ2 חלקים:
      i. Tf – term frequency – מתגמל את הערך כאשר הוא נפוץ בתוך אותו מסמך
      ii. Idf – inverse document frequency – מעניש ערך כאשר הוא נפוץ גם במסמכים אחרים
   d. עבור ערך (term) t ומסמך d באוסף מסמכים D האלגוריתם מוגדר כך:

$$tfidf(t,d) = tf(t,d) \cdot idf(t) \qquad (1)$$

$$tf(t,d) = \frac{count(t,d)}{\sum_{v \in d} count(v,d)} \qquad (2)$$

$$idf(t) = log\frac{|D|}{|\{d \in D: t \in d\}|} \qquad (3)$$

כאשר count(t,d) מחזיר את מספר המופעים של הערך t במסמך d

3. חישוב ערך הThreshold :
   a. נחזיק 2 רשימות – predicted, target בגודל l training set n=l
   b. עבור כל בקשה i בtraining set, נחשב את ערך ה similarity שלו לסיווג Valid, כפי שמפורט בשלב הפעלת המסווג, סעיף 1.b.ii. ונשמור ערך זה ב predicted[i]
   c. אם בקשה i אכן שייכת לvalid נשמור בtarget[i] את הערך 1, אחרת 1-.
   d. נמצא את ה threshold האופטימלי, כך שנקבל מקסימום TP ומינימום FP ביחס לערכים ברשימות predicted & target– כלומר מקסום ה - AUC = Area Under RoC בעת בחירת ערך t.

בסוף שלב הלמידה מתקבלים הנתונים הבאים :

- ערך tfidf עבור כל term בכל סוג מסמך לטובת חישוב similarity בשלב הפעלת המסווג
- ערך Threshold שאיתו המסווג מחליט לסווג בקשה חדשה לValid, במידה והsimilarity של הבקשה החדשה ל Valid גבוהה מהThreshold.

<u>דוגמה לתוצאת שלב הלמידה (מתוך הרצת הפרויקט):</u>

הבקשה המקורית :

```
GET
/RACu9rM/Acmd_2j9@/ixybX3.34d8/o1eoerie1oethEn8Evs/iTJU/8Imzn/t1LPxgbPw9
HyKe-
VZeUB/or6Mstoto5ieae/7Bs@Kz8ojo3V.WF95gMV/nRT8LXBgkXajF2/eairentaa/mT
P4@J6Tw.swfu5io=%3Cxml+id++%3D+%22X+++%22+++%3E%3Ca%3E%3Cb++%
3E%26lt%3Bscript%3E%5Bwindow.open%28%27http%3A%2F%2F155.236.12.33%2Far
ng.asp%27%2Bdocument.cookie%29%3B%5D%26lt%3B%2Fscript%3E%3B%3C%2Fb
++%3E%3C%2Fa+%3E%3C%2Fxml++%3E HTTP/1.0

Host: www.cb9HgselA.it
```

```
Connection: keep-alive

Accept: video/mpeg; q=0.8, audio/*

Accept-Charset: euc-jp; q=0.6, iso-10646-ucs-2; q=0.9, iso-8859-6; q=0.0, iso-8859-8-
i; q=0.2, iso-8859-4

Accept-Encoding: deflate; q=0.6, compress; q=0.6, identity; q=0.8

Accept-Language: *; q=0.0

Cache-Control: min-fresh=80

Client-ip: 50.84.88.200

Cookie: tr2eixtciyd=626116851; z74fnatusaAucA8=hsssystemstdinst7n4i9wp-
isPdocument%26nwindow.open; yn=t+%3F; zatehntri=7jtou8oaeUofa5

Cookie2: $Version="936"

Date: Sat, 05 Jul 08 07:28:46 GMT

ETag: W/"eK9JG65XxM2Bk-Ko"

Expect: 100-continue

From: dhMEe9le@RpqtlrdwLb.be

Via: 0.0 240.221.188.123, HTTP/0.4 www.eYEtzha.jpg:8, 4.3 www.ivo4he.css

Transfer-Encoding: deflate

Upgrade: vihit/6.1, e9om/9.5, eo8qd/9.3

Warning: 630 69.99.208.229 "aypepysetwq" "Mon, 12 Apr 04 06:01:50 CET"

X-Forwarded-For: 131.225.250.219

X-Serial-Number: 71254341199
```

איור 7: בקשת HTTP מקורית (XSS)

הבקשה לאחר ביצוע Tokenization : (כל טוקן מופרד בפסיק)

```
GET ,
/RACu9rM/Acmd_2j9@/ixybX3.34d8/o1eoerie1oethEn8Evs/iTJU/8Imzn/t1LPxgbPw9
HyKe-
VZeUB/or6Mstoto5ieae/7Bs@Kz8ojo3V.WF95gMV/nRT8LXBgkXajF2/eairentaa/mT
P4@J6Tw.swfu5io= , xml , id , X , a , b , lt , script , window.open , http , 155.236.12.33 ,
arng.asp , document.cookie , lt , script , b , a , xml , HTTP/1.0
```

Host: , www.cb9HgselA.it

Connection: , keep-alive

Accept: , video/mpeg; q=0.8, , audio/*

Accept-Charset: , euc-jp; q=0.6, , iso-10646-ucs-2; q=0.9, , iso-8859-6; q=0.0, , iso-8859-8-i; q=0.2, , iso-8859-4

Accept-Encoding: , deflate; q=0.6, , compress; q=0.6, , identity; q=0.8

Accept-Language: , *; q=0.0

Cache-Control: , min-fresh=80

Client-ip: , 50.84.88.200

Cookie: , tr2eixtciyd=626116851; z74fnatusaAucA8=hsssystemstdinst7n4i9wp-isPdocument , nwindow.open; yn=t , ; zatehntri=7jtou8oaeUofa5

Cookie2: , $Version="936"

Date: , Sat, , 05 , Jul , 08 , 07: 28: 46 , GMT

ETag: , W/"eK9JG65XxM2Bk-Ko"

Expect: , 100-continue

From: , dhMEe9le@RpqtlrdwLb.be

Via: , 0.0 , 240.221.188.123, , HTTP/0.4 , www.eYEtzha.jpg: 8, , 4.3 , www.ivo4he.css

Transfer-Encoding: , deflate

Upgrade: , vihit/6.1, , e9om/9.5, , eo8qd/9.3

Warning: , 630 , 69.99.208.229 , "aypepysetwq" , "Mon, , 12 , Apr , 04 , 06: 01: 50 , CET"

X-Forwarded-For: , 131.225.250.219

X-Serial-Number: , 71254341199

איור 8: בקשת HTTP (XSS) לאחר ביצוע Tokenization

כמו כן, מתקבלים מילוני ה tfidf עבור כל סוג תקיפה. לדוגמה עבור XSS : (הצגה חלקית כי המילון המלא גדול. הקובץ המלא נמצא בתוצאת ההרצה בקובץ XSS_Sorted_TFIDF.txt)


```
{
   "GET": 0.0,
   "Host: ": 0.0,
   "Connection: ": 0.0,
   "close": 0.0,
   "Accept: ": 0.0,
   "audio/*,": 0.0,
   "application/*; q=0.1": 0.0,
…….
   "script": 0.00045813185434644615,
   "onmouseover": 0.0005020776302852718,
   "window.open": 0.0005057693775667812,
   "url": 0.0007494246981463983,
   "img": 0.0007875727533886617,
   "src": 0.0009401649743577149,
   "cgi-bin": 0.0010484562279486557,
   "http": 0.001201048448917709,
   "document.location.replace": 0.0015726843419229836,
   "javascript": 0.001628060551145624,
   "alert": 0.002071070224926746,
   "document.cookie": 0.003152752178408986
}
```


איור 9: הצגה חלקית של מילון tfidf עבור XSS

5.3.2. הפעלת המסווג

לאחר אימון המסווג, משתמשים במסווג לטובת טיפול בבקשת HTTP חדשה. אופן השימוש:

1. טיפול בהודעה חדשה:

   a. הבקשה החדשה הופכת ל term vector כפי שבוצע בשלב הלמידה

   b. מוסיפים את הבקשה לטובת הסיווג כמסמך חדש באוסף המסמכים (ה corpus) כך שניתן יהיה לחשב עבורו את tfidf(t,R).

   c. הבקשה מסווגת לאחת מ8 האפשרויות לפי אחוז ההתאמה לכל אחד מהאפשרויות:

      i. מחשבים את הסתברות החלוקה בתצורה הבאה

$$P(A = a|R) = \alpha \cdot sim_{cos}(a, R)$$

כאשר A הינו משתנה מקרי של התקיפות (Attacks) האפשריות,

a היא השמה ספציפית של תקיפה

α הוא משתנה מנורמל המבטיח $\sum_a P(A = a|R) = 1,$ כאשר R זו הבקשה לסיווג,
ו simcos זו פונקציית קוסינוס של דמיון

      ii. פונקציית הדמיון מוגדרת בצורה הבאה:

$$sim_{cos}(a, R) = \frac{\vec{a} \cdot \vec{R}}{\|\vec{a}\| \cdot \|\vec{R}\|}$$

$$= \frac{\sum_{t \in a \cap R} tfidf(t, a) \cdot tfidf(t, R)}{\sqrt{\sum_{t \in a} tfidf(t, a)^2} \cdot \sqrt{\sum_{t \in R} tfidf(t, R)^2}}$$

      iii. בהתבסס על תוצאות חישוב ההסתברות (שלב i לעיל), הסיווג לבקשה ניתן לפי החוקיות הבאה:

$$class(R) = \begin{cases} Valid & \text{if } P(A = Valid) > T \\ \underset{a}{\text{argmax}}\, P(A = a|R) & \text{otherwise} \end{cases}$$

כאשר ה T=Threshold להחלטה מחושב בהתאם לתוצאות סט הבדיקות וההרצות השונות ב -10 fold cross-validation

<u>מתוך ההרצה בפרויקט:</u>

כאשר נבצע סיווג לבקשה מאיור [7], נקבל את הסיווג שלו על בסיס ערכי ה similarity שמתקבלים עבורו:

Request similarity for assignment Valid is: 0.001663292596371107З

Request similarity for assignment XSS is: 0.16455117396380006

Request similarity for assignment SqlInjection is: 0.00014065725214734332

Request similarity for assignment LdapInjection is: 0.00028799407497458375

Request similarity for assignment XPathInjection is: 9.880186438392996e-05

Request similarity for assignment PathTransversal is: 1.3678506364179187e-05

Request similarity for assignment OsCommanding is: 0.0005497608952418514

Request similarity for assignment SSI is: 0.0005740622762820426

Normalized probability after multiplication in alpha(5.956656220783774):
0.9801747740287426

Best assignment: XSS

איור 10: חישוב דמיון לבקשה מתוך הרצת הפרויקט

אנו מקבלים ערכים שונים לsimilarity עבור כל אחד מסוגי התקיפות. הalpha שחושבה עבור בקשה זו היא ..5.9566 ותפקידה לנרמל את סכום ההסתברויות ל1.

הsimilarity הכי גבוהה התקבל ל"XSS", ולאחר ההכפלה בalpha נקבל סבירות התאמה של 98.017% לפי המסווג (ואכן הבקשה שייכת לXSS).

ניתן לראות בהרצה את חמשת המונחים בעלי המשמעות הגבוהה ביותר בבקשה שאיתם המסווג קישר את הבקשה לסוג התקיפה (ההרצה ממיינת את חמשת המונחים עם ערכי tfidf הגבוהים ביותר עבור בקשה זו):

Top 5 terms in the request, by their tfidf value:

('document.cookie', 0.003152752178408986)

('http', 0.001201048448917709)

('window.open', 0.0005057693775667812)

('script', 0.00045813185434644615)

('xml', 0.00023333691757344198)

איור 11: הצגת המונחים בעלי המשמעות הגבוהה לסיווג בבקשה

ניתן לראות באיור [11] את חמשת המונחים (טוקנים) שנמצאו בבקשה ובעלי ערכי הtfidf שלהם במסמך הXSS.

5.3.3. סיבוכיות זמן

בחישוב סיבוכיות הזמן, מנתחים הן את זמן הריצה של הפעלת המסווג, והן של שלב הלמידה. עם זאת, שלב הלמידה יכול להתבצע בOffline ובפעולות לא רציפות, בעוד זמן ריצת המסווג חייב להיות בזמן אמת על הבקשות המתקבלות, ולכן חשיבות הזמן בפעולת הלמידה פחותה.

5.3.3.1. סיבוכיות הלמידה

לימוד המודל כולל חישוב משקלי tf-idf עבור כל מונח בכל אחד מהבקשות במאגר הלמידה. במעבר בודד על מאגר המידע, ניתן לחשב את $\sum_{v \in d} count(v,d)$, $count(t,d)$ ו $idf(t)$.

באמצעות המרכיבים הללו, ניתן לחשב את משקלי tf-idf במעבר נוסף על מאגר הלמידה.

לכן סיבוכיות שלב הלמידה הוא O(K) כאשר K הוא מספר הטוקנים הכולל במאגר הלמידה. במילים אחרות, הסיבוכיות היא $O(|D| \cdot L_d)$ כאשר L_d הוא האורך הממוצע של המסמכים ב D.

5.3.3.2. סיבוכיות הסיווג

ניתוח סיבוכיות הסיווג דומה סיבוכיות הלמידה, אך יש שוני עיקרי ביניהם:

1. עובדים על בקשה יחידה q, ולא על סט אימון מלא
2. בנוסף לחישוב משקלי ה tf-idf, יש לחשב את sim_{cos}

כיוון שחישוב הדמיון תלוי במשקלי הtf-idf, אותם ניתן לחשב בזמן $O(|q|)$, הפעלת המסווג כולו עבור 8 סיווגים שונים תרוץ בזמן $O(8 \cdot |q|) = O(|q|)$.

נשים לב שכיוון שכל טוקן בבקשה חייב לעבור פרסור בשרת בכל מקרה (לטובת טיפול בבקשה), אנחנו לא יכולים לצפות לעיבוד הבקשה בזמן הקטן מזמן ליניארי.

5.4. תוצאות המסווג

לטובת חישוב תוצאות המסווג, המאמר מחלק את מאגר המידע ל10 חלקים לטובת ביצוע וואלידצית 10-fold cross-validation.

על מנת לשחזר את אופן העבודה שקיבלו המתמודדים באתגר (שכן המאמר נכתב מספר שנים לאחר קיום האירוע), שחזרו כותבי המאמר את מתאר הבדיקה – למידה על כל מאגר הלמידה, וביצוע מדידות הצלחה על סט הבדיקה המלא.

המאמר מציג את תוצאותיו אל מול 2 הצעות לפתרון אחרות – LM (Language Model), ו C4 (אלגוריתם ליצירת עץ החלטה)

המדדים איתם בוחנים את תוצאות המסווג באתגר הן : precision, recall, F1

המאמר מוסיף פרמטר נוסף של accuracy – סיווג תקיפה / זדוני, ו AUC (Area Under ROC).
לביצוע הסיווג על 8 האפשרויות, מבצעים AUC כמשקל משוכלל ממוצע של 1-vs-rest על כל 8 הסיווגים האפשריים.

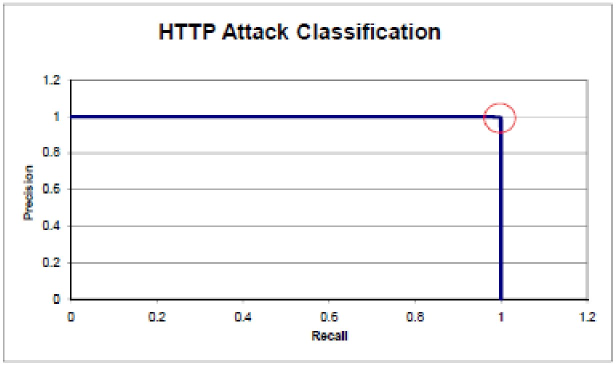
	Classifier		
	tf-idf	LM	C4
Training Set Only			
Accuracy	> 0.99	-	0.77
AUC	> 0.99	-	-
Precision	> 0.99	0.98	-
Recall	> 0.99	0.93	-
F1	> 0.99	0.96	-
Discovery Challenge (Train/Test)			
Accuracy	0.94	-	-
AUC	0.97	-	-
Precision	0.98	0.82	0.50
Recall	0.88	0.78	0.47
F1	0.93	0.80	0.48

איור 12: תוצאות המסווג במאמר

הטבלה באיור [12] מסכמת את תוצאות המסכם על כלל הפרמטרים, וכן השוואה אל מול 2 המסווגים שהוצגו בתוצאות האתגר.

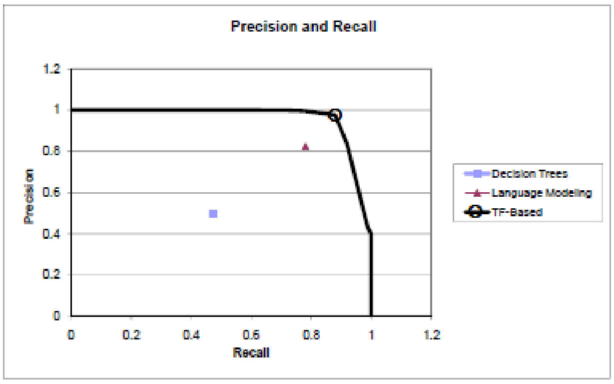
המסווג משיג תוצאות כמעט מושלמות על מאגר האימון, וכן תוצאות גבוהות מאוד על מאגר המבחן – במיוחד אל מול שני המסווגים המתחרים.

ניתן לראות כי המסווג המבוסס על tf-idf אינו רגיש למצב בו הקונטקסט לא קיים עבורו, שכן האלגוריתם לא מתבסס על נקודות אלו. לעומת זאת, תוצאות שאר האלגוריתמים נפגעו משמעותית – עובדה המשפיעה על היכולת להשתמש במסווג בעולם האמיתי, שכן לא תמיד המידע



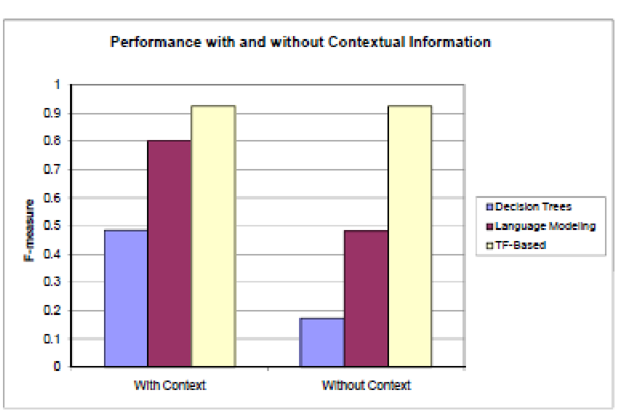
איור 13:תוצאת האימון על סט הלמידה

באיור [13] ניתן לראות את בחירת הThreshold, T עבור שלב הלמידה.



איור 14:תוצאות המסווגים על סט המבחן

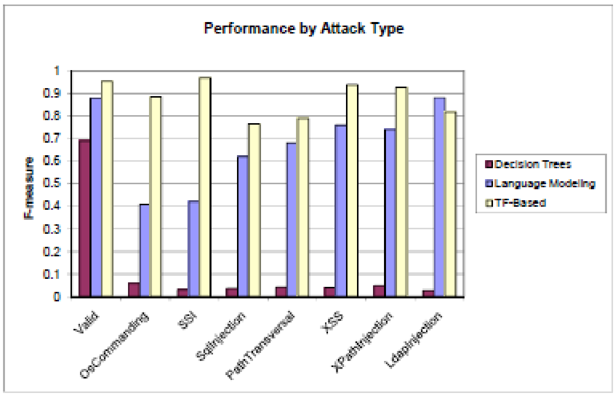
באיור [14] ניתן לראות את תוצאות המסווגים על מאגר המבחן. העיגול מסמן את הThreshold שנבחר מתוך שלב האימון. ניתן לראות שהPrecision וRecall של המסווג גבוהים משמעותית מעצי החלטה ו LM.



איור 15:תוצאת המסווג עם וללא גישה לקונטקסט

ניתן לראות מאיור [15] כי המסווג המבוסס על tf-idf אינו רגיש למצב בו הקונטקסט לא קיים עבורו, שכן האלגוריתם לא מתבסס על נקודות אלו. לעומת זאת, תוצאות שאר האלגוריתמים נפגעו משמעותית – עובדה המשפיעה על היכולת להשתמש במסווג בעולם האמיתי, שכן לא תמיד המידע

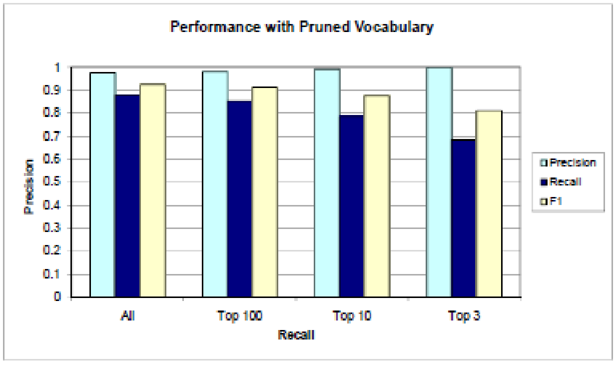
נגיש למערכת ההגנה, ובנוסף הגישה המועדפת היום הינה Plug & Play ללא התערבות והכנסת מידע ידנית למערכת ההגנה.



איור 16:תוצאת המסווג לפי סוג ההתקפה

המסווג מציג תוצאות גבוהות בצורה קונסיסטנטית – מעל 75% הצלחה עבור כל אחד מסוגי התקיפה. ניתן לראות שתוצאות שאר המודלים מאוד מבוזרים לאורך סוגי התקיפות, וכן בעלי ביצועים נמוכים יותר (למעט זיהוי Ldap Injection ב LM). התוצאות הממוצעות של שני המסווגים קטנים מ50% - כלומר יותר יעיל להטיל מטבע בסיווג מאשר להשתמש בהם.

על מנת להציג את אפקטיביות המודל, המאמר מציג את תוצאות המסווג עם מילון מצומצם – הקטנת כמות הtermsשל כל סיווג. Top 100 אומר שלוקחים רק את 100 ה terms בעלי המשמעות הגבוהה ביותר (ערך tfidf) לסיווג זה. זמן הריצה של המודל מושפע ממספר המילים במילון שעובדים איתו. אם המסווג מצליח לשמור על תוצאות גבוהות עם מילון מצומצם, ניתן לשפר את זמן הריצה של המסווג בלי לפגוע באיכות הסיווג.



איור 17:תוצאת המסווג עם מילון מצומצם

ניתן לראות מאיור [17] שניתן להשיג סיווג טוב עם צמצום המילון. ניתן לשים לב שעם צמצום גודל המילון מתרחשים 2 דברים במקביל – מקבלים עלייה באחוז ה precision בעוד יש ירידה בrecall. התנהגות זו צפויה מ2 סיבות:

1. הקיום של המונחים העיקריים בלבד בהודעה לסיווג מראה על אינדיקציה גבוהה לשייכות לאותה מחלקה, ולכן כמות הfalse-positive קטנה.

2. אנו מורידים מונחים שימושיים לסיווג, ולכן כמות ה false-negative גדלה.

יתרון נוסף של מודל ה tf-idf אל מול שאר המסווגים שהוא מייצר מודל קריא, מובן לבני אדם, המורכב ממילות מפתח לכל סוג תקיפה, ממוינים לפי ערך הtf-idf שלהם.

	Training Data		Test Data	
	Term	Tfidf Score	Term	Tfidf Score
LDAP Injection	had*	0.005844	had*	0.004065
	objectclass	0.003944	objectclass	0.003861
	*o	0.003872	*o	0.002828
	brien*	0.003872	brien*	0.002828
	netscaperoot	0.001978	displayname	0.002616
Command Execution	..	0.003871	..	0.003558
	dir	0.003546	/c	0.003229
	/c	0.003328	dir	0.002507
	–	0.001650	–	0.001733
	../winnt/system32/cmd.exe	0.001612	../winnt/system32/cmd.exe	0.001678
Path Traversal	..	0.016041	..	0.016415
	.	0.005513	.	0.008600
	virtual	0.002526	virtual	0.001427
	–	0.001713	–	0.000968
	include	0.001263	file	0.000927
SSI Attack	–	0.006241	–	0.006003
	virtual	0.003719	statement	0.002167
	include	0.001859	odbc	0.002167
	statement	0.001275	virtual	0.001967
	odbc	0.001275	progra	0.000810
SQL Injection	**	0.003874	**	0.005199
	select	0.000883	statement	0.001163
	statement	0.000832	odbc	0.001163
	odbc	0.000832	–	0.000807
	union	0.000805	union	0.000674
XPATH Injection	path	0.005394	path	0.005449
	count	0.005108	count	0.005072
	child	0.003756	text	0.002616
	text	0.002421	comment	0.002093
	position	0.002200	child	0.001065
Cross-Site Scripting	document.cookie	0.006498	document.cookie	0.006504
	alert	0.004298	alert	0.004287
	javascript	0.003463	javascript	0.003449
	document.location.replace	0.003209	document.location.replace	0.003208
	url	0.001644	http	0.002411
Valid (No Attack)	13224	0.000061	dddddd	0.002054
	213.191.153.150	0.000057	lkl	0.000969
	9055,045,32	0.000055	large–majorite*des_membres	0.000751
	27260320301	0.000054	ministre-de-l-enseignement-superieur	0.000265
	13.228.134.190	0.000054	tehgghgjty	0.000259

איור 18:מונחים מובילים לכל סוג תקיפה, ממוינים לפי TF-IDF

ניתן לראות באיור [18] את המונחים המובילים לכל סוג תקיפה, ואת ציון ה TF-IDF שלהם. מאגר המידע חולק ל Training/Test, וניתן לראות באיור [18] את המונחים המובילים בכל חלוקה. בחלוקה זו ניתן לראות כי לסיווגים שאינם בעלי מונחים "הדוקים" לסיווג, כגון Valid, מתקבלים ערכים שונים לחלוטין, בעוד בCommand Execution מתקבלים אותם מונחים (סדר קצת שונה).

ניתן לזהות דפוסי התנהגות מוכרים עבור שלל התקיפות – כגון document.cookie ב XSS – ניסיון השגת הCookie של המשתמש.

נשים לב שאין משמעות למונחים של Valid, שכן הם שרירותיים – הרי שמחלקת Valid הינה מחלקת "Catch All" עבור הודעות שלא נפלו על אף קטגוריה אחרת. ניתן גם לראות שהציון עבור ה Top Terms ממחלקה זו נמוכים משמעותית משאר המונחים בשאר הקטגוריות – אינדיקציה שמראה שהקיום של מונח זה לא מקשר בצורה חזקה למחלקה זו.

6. הפרויקט

המאמר שעל בסיסו בוצע הפרויקט המעשי, מציע שיטה לסיווג הודעות HTTP לסוגי התקיפות השונות או סיווג כהודעה תקינה, בהתבסס על סט מידע בשם ECML/PKDD 07 Discovery Challenge.

סט המידע מכיל כ 50,116 הודעות מסווגות, כאשר המידע שמור בקובץ xml בודד בתצורה הבאה :

- מספר סידורי של הדוגמה
- הקשר (Context) של הדוגמה למערכות/אפליקציות המקבלות את ההודעה – מערכת ההפעלה, תוכנת שרת הWEB, האם יש שימוש ב Ldap, האם יש שימוש ב Sql, האם המערכת תומכת ב XPath
- האם ההודעה אכן בהקשר שתואר
- סוג התקיפה המשויכת להודעה – (בסוגריים כמות ההודעות בdataset לסיווג זה)
    - Valid (35006)
    - XSS (1825)
    - Sql Injection (2274)
    - Ldap Injection (2279)
    - XPath Injection (2279)
    - Path Traversal (2295)
    - Os Commanding (2302)
    - SSI (Server Side Includes) (1856)
- מיקום האינפוט הזדוני – URI , Query, Body, Header

הפרויקט יכיל מימוש לטובת סיווג ההודעות ממאגר המידע, ויכתוב את תוצאותיו לקבצי טקסט המכילים מידע עבור כל סיווג את הצלחתו בסיווג (כמות הודעות לסיווג, וכמות הודעות שסווגו לא נכון).

מתוך מסקנות ותוצאות המסווג המוצג במאמר, ניתן לראות ששימוש בערך הקונטקסט (האם הדגימה בעלת קונטקסט לשרת המורץ, לSQL,LDAP וכו') אינו משפיע על ביצועי המסווג. כיוון שאופן השימוש לא מתואר במאמר, וכן השימוש עצמו לא מועיל, במימוש זה לא אתייחס לקונטקסט כלל.

6.1. תכולות העבודה

העבודה כוללת מימוש של האלגוריתם המוצג במאמר על מאגר המידע הזמין להורדה בלינק הבא.

העבודה כוללת את התכולות הבאות:

1. למידת סוגי התקיפות, מאפייניהם, חולשות מוכרות ושיטות מוכרות לניצול מתוך המאמר
2. כתיבת תשתית פרסור קובץ ההודעות המסווגות לטובת הרצת לוגיקות חישוביות כפי שמתוארות במאמר ובחלק הקודם המתאר את האלגוריתם בשלב המקדים
3. מימוש האלגוריתם המוצע במאמר:
   a. פריסת ההודעות לפי סוגי הסיווגים בתצורה המתאימה לאלגוריתם
   b. כתיבת אלגוריתם הלמידה מתוך קבצי המידע המקוטלגים
   c. הרצת אלגוריתם הלמידה לקבלת ערכי המשקלים של כל term במסמך התקיפה (Dictionary בין המונח לבין ערך ה tfidf) לטובת סיווג הודעות
4. הרצת האלגוריתם בתצורת 10-fold cross validation לבדיקת איכות המסווג הנוצר, והשוואתו אל מול תוצאות המאמר
5. הצגת תוצאות האלגוריתם, וכתיבת ניתוח ומסקנות מתוך הנתונים שיתקבלו

6.2. סביבת פיתוח וכלי עבודה

הפרויקט נכתב בשפת python 3.8 מעל סביבת Windows x64 באמצעות Pycharm 2019.1.1.

הפרויקט משתמש בVirtual Environment של python לטובת ניהול הספריות והתלויות.

הפרויקט משתמש בספריות numpy,sklearn,pandas לטובת פעולות Data Mining שונות, ניהול מאגר הבקשות, ובעיקר לטובת חישוב ה Threshold האופטימלי.

הפרויקט משתמש בxml.etree לטובת פרסור ועבודה עם קבצי XML, הנדרשים בשלב עיבוד המידע המקדים על קובץ הלמידה המקורי.

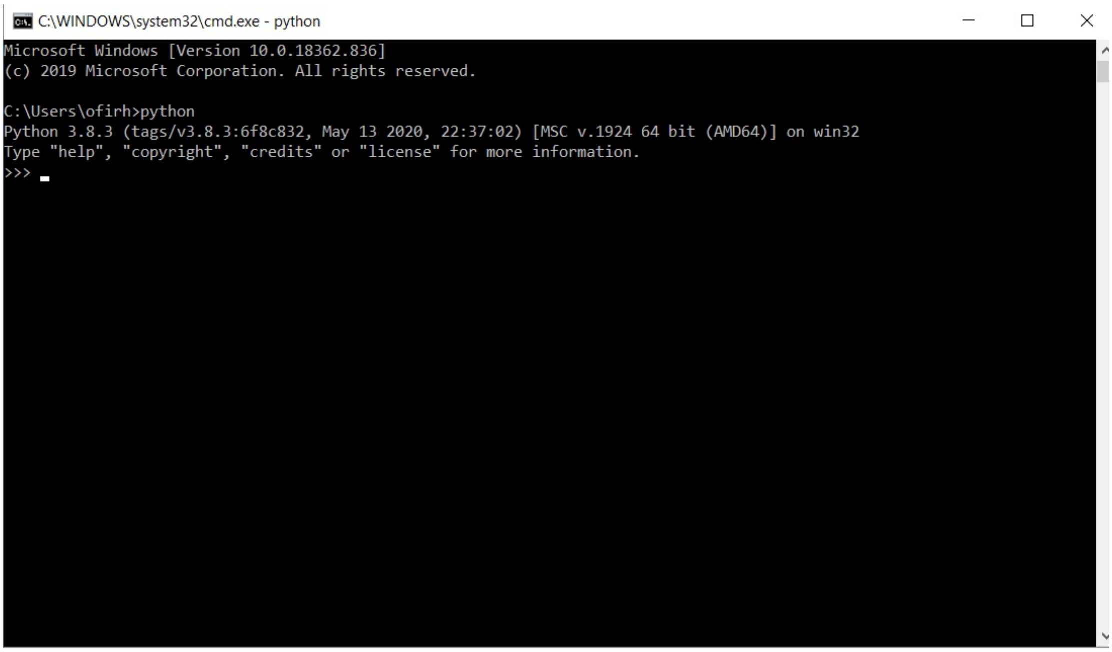
כמו כן, הפרויקט משתמש בספריות סטנדרטיות בשפה לטובת פעילויות שונות, כגון:

- json לטובת כתיבת הTf-idf המחושב עבור כל סוג תקיפה כקובץ json קריא לבני אדם, ממויין לפי ערך יורד
- Threading לטובת עבודה עם threads בתהליך חישוב הthreshold (מפורט בארכיטקטורה)

- Pickle - כתיבת אובייקטים מצומצמים לקובץ לטובת יכולת טעינה מהירה ע״י התהליך בשלב מאוחר יותר

6.2.1. הכנת הסביבה להרצה

ראשית, נוודא שמותקן אצלנו python בגרסה 3.8 ומעלה, והמערכת מכירה את התוכנית (הרצת python בcmd תפתח טרמינל של python) וכן נוודא את גרסת הספרייה:



איור 19: ווידוא גרסת הפייתון שבשימוש

על מנת להכין את סביבת הפיתוח להרצה, יש לקנפג סביבה וירטואלית להרצת תהליך הפייתון (בתוך command line):

```
python -m venv /path/to/new/virtual/environment
```

לאחר יצירת סביבת ההרצה, יש להגדיר למערכת להשתמש (״להפעיל״) את הסביבה החדשה:

```
workon [environment_name]
```

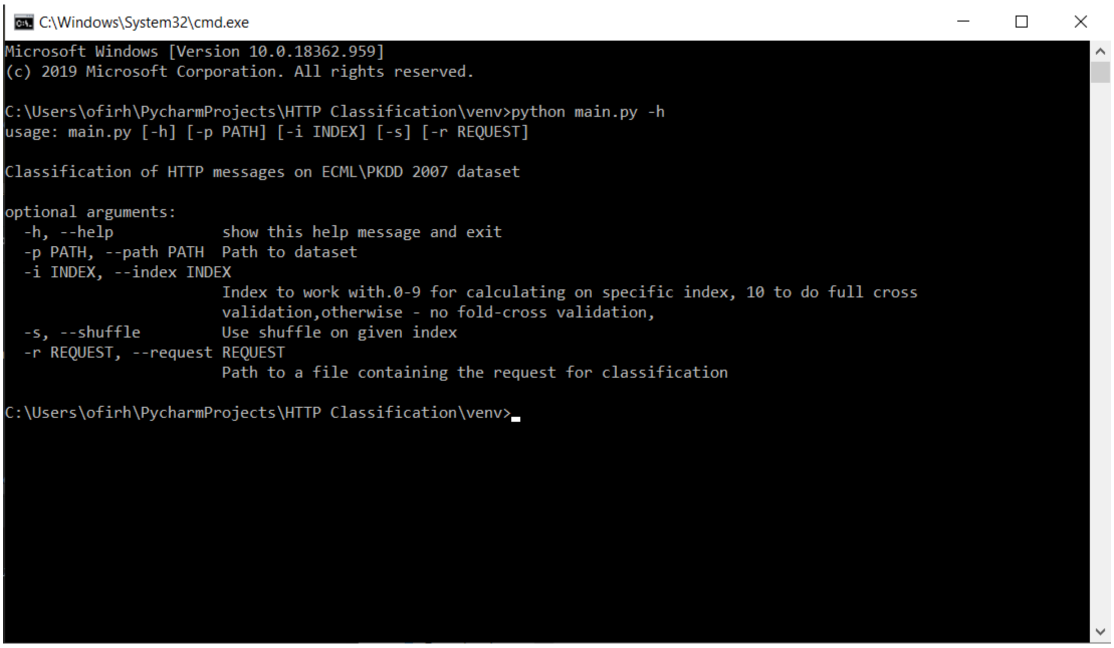
כאשר הסביבה מוגדרת ופועלת, נרצה להתקין את כלל התלויות הנדרשות בסביבה:

```
pip install -r requirements.txt
```

קובץ הrequirements.txt מכיל את כלל הספריות הנדרשות לטובת הספריה. הקובץ נמצא בתוך תיקיית venv (הסביבה הוירטואלית שהוגדרה בכתיבת הפרויקט). ההרצה תתקין את כלל הספריות הנדרשות להרצת הפרויקט.

6.2.2. הרצת הפרויקט

הרצת הפרויקט מבוצעת באמצעות הcommand arguments בתצורה הבאה: (הרצה עם דגל h– תציג את מסך הhelp)



איור 20: פלט ההרצה של התוכנית לקבלת עזרה

על מנת להריץ את המסווג יש לתת לו את הדגלים הבאים:

p- , הנתיב לקובץ הdataset.

לדוגמה C:\Users\ofirh\outputDir\ecml_pkdd_learning_dataset.xml

i-, האינדקס ב10-fold cross שנרצה לעבוד עליו.

אינדקס שבין 0-9 יריץ את המסווג על האיטרציה שבה אינדקס i הינו החלק שנבחר להיות ה test, ושאר האינדקסים מהווים את ה training. (שימושי לטובת חלוקת העבודה באמצעות מספר הרצות התוכנית במקביל, ובכך לנצל את משאבי המחשב המריץ).

אינדקס 10 יגדיר למסווג לבצע עבודה על כל האינדקסים (בעצם לבצע את כל ה 10-fold cross)

כמו כן, ניתן לעבוד ללא חלוקה כלל, ולהריץ את המסווג על כל הdataset באמצעות כל ערך אחר (לדוגמה 1-)

s- , דגל שמגדיר האם בעת החלוקה ל10 חלקים, לבצע shuffling למידע. הערבול מבוצע בצורה רנדומלית בכל הרצה, ולכן התוצאות יכולות להיות שונות בין הרצות שונות על אותם אינדקסים כאשר מבצעים ערבול. אין התייחסות לערך זה כאשר עובדים ללא חלוקה של ה dataset

r-, סיווג בקשה ספציפית. יש להזין נתיב לקובץ המכיל את הבקשה לסיווג.

יש לקחת בחשבון שהרצת שלב הלימוד יכולה לקחת עד מספר שעות, בהתאם למה שרץ. הפרויקט מבצע את כלל התהליכים הנדרשים בהרצה (כפי שמוגדר בחלק 6.3 האלגוריתם), ופעולות העיבוד המקדים כוללות חישובים וכתיבה מרובה.

כמו כן, בזמן ההרצה נוצרים קבצים הן לשימוש פנימי בריצת הפרויקט (קריאים/לא קריאים לבני אדם), והן לטובת כתיבת תוצאות ההרצה.

במידה וכלל דרישות הקדם קיימות בתיקייה (כלומר שלב הלימוד כבר בוצע), הרצה מול בקשה בודדת (r-) תהיה מהירה מאוד.

6.2.3. תוצאת ההרצה

התוכנית כותבת לתיקיית העבודה (התיקייה המכילה את ה dataset) קבצי תוצאה עבור כל סוגי התקיפות, בהתאם לפרמטרים שהתבקשו. שמות הקבצים הינם מהפורמט:

AttackType_(shuffle)?_(fold_index)?_Classification_Results.txt

לדוגמה עבור הרצה של fold-cross עם אינדקס 7, ללא shuffle נקבל בין היתר את הקובץ הבא:

SSI_7_Classification_Results.txt

עבור הרצה על כלל הdataset נקבל לדוגמה את הקובץ Valid_Classification_Results.txt.

<u>תוכן הקובץ</u>:

במהלך ביצוע הסיווג, התוכנה כותבת לקובץ עבור סיווג כושל את האינדקס במערך הבחינה של הבקשה הבעייתית (לטובת מציאה שלו), את הסיווג הבעייתי שניתן, ואת הדימיון שחושב עבור הסיווג הלא נכון (כדי להבין האם היינו יחסית בטוחים בסיווג הנ"ל)

לדוגמה, בקובץ PathTraversal_Shuffle_1_Classification_Results.txt קיימת השורה הבאה:

90 Valid,0.4312736375269691

כלומר האינדקס ה90 בסט הבחינה סווג כתקין, עם ביטחון של 43% (לא גבוה, אבל הThreshold שחושב לValid בד"כ סביב ה0.4~)

6.3. אלגוריתם

בעת קבלת נתוני המשתמש לאופן ביצוע הלמידה והסיווג, מבוצעות הפעולות הבאות:

[1] פרסור קובץ הנתונים המגיע כXML, ויצירת 8 קבצי XML חדשים – קובץ לכל סוג תקיפה
[2] עבור כל קובץ XML, שליפת התוכן הטקסטואלי מהXML, ויצירת קובץ חדש לכל סוג תקיפה, המכיל את בקשות הHTTP כטקסטואליות.
[3] אם עובדים בתצורת fold-cross:
   a. חלוקת הdataset הטקסטואלי (משלב 2) ל10 חלקים (עם אפשרות ערבוב לסדר ההודעות)
   b. עבור כלל החלקים שהוגדרו ללמידה – טען אותם לאוסף המסמכים לטובת הפעלת האלגוריתם המתואר בחלק (*חישוב ערכי tfidf*): של המסמך
[4] אם עובדים על כל ה dataset:
   a. טען את כל המסמכים שנוצרו משלב [2] לאוסף המסמכים, והפעל את שלב *חישוב ערכי tfidf*:
[5] חשב את הThreshold שעבורו יש להגדיר הודעה כValid:
   a. נרצה למצוא ערך t אופטימלי, כך שהחל ממנו והלאה המסווג יסווג את הבקשה לValid, ללא תלות בערכי הsimilarity לשאר התקיפות (כפי שמתואר בחלק 5.3.2 הפעלת המסווג, סעיף 1.b.iii)
   b. נחזיק 2 רשימות – predicted, target בגודל | training set |=n
   c. עבור כל בקשה i בtraining set, נחשב את ערך ה similarity שלו לסיווג Valid, כפי שמפורט בשלב הפעלת המסווג, סעיף 1.b.ii.
   d. את ערך הדמיון הכפל בalpha שמחושב לפי האלגוריתם המתואר ב- *מחשבים את הסתברות החלוקה בתצורה הבאה,* על מנת לנרמל את סכום ההסתברויות שהתקבלו ל1, ונשמור ערך זה ב predicted[i]
   e. אם בקשה i אכן שייכת לvalid נשמור בtarget[i] את הערך 1, אחרת -1.
   f. נמצא את ה threshold האופטימלי, כך שנקבל מקסימום TP ומינימום FP ביחס לערכים ברשימות target & predicted– כלומר מקסום ה - AUC = Area Under RoC בעת בחירת ערך t.

[6] עבור כל בקשה בtest set בצע את הסיווג לפי המתואר בחלק: *הבקשה מסווגת לאחת מ8 האפשרויות לפי אחוז ההתאמה לכל אחד מהאפשרויות*:
[7] כתוב את תוצאות הסיווג עבור כל סוג תקיפה בקובץ נפרד, כך שיכיל את המידע הבא:
   a. מספר הבקשות הכולל לתקיפה, ומספר הבקשות שלא סווגו נכון
   b. עבור סיווג כושל – האינדקס שלו בtest set, הסיווג השגוי, והביטחון (הדמיון) שחושב עבור הסיווג השגוי

6.4. ארכיטקטורה ומבנה הספריות

הפרויקט מייצר מסווג הנוצר מפרסור ועבודה מקדימה על קובץ ה dataset. האלגוריתם עצמו, וכן אופן השימוש הינם Stateless ולכן אין צורך בבניית מחלקות ואובייקטים, אלא שימוש בפונקציות ויכולות בהתאם לצורך ולסדר העבודה הנדרש.

מימוש זה מאפשר עבודה מקבילית לטובת חישוב תוצאות המסווג, נקודה קריטית שתועלה בהמשך, שכן הן ממגבלות השפה, והן מכמות הבקשות לעבודה נרצה להאיץ את זמן החישוב.

<u>הפרויקט מורכב מ2 קבצים :</u>

[1] קובץ main.py שתפקידו לקבל את קלט המשתמש הרצוי, והפעלת שרשרת הפעולות הנדרשות בהתאם לדרישת המשתמש.
[2] קובץ DatasetHandler.py המתפקד כספרייה המייחצנת API לקובץ ההרצה הראשי לטובת שלבי הטיפול ב dataset – הלמידה וההרצה של המסווג.

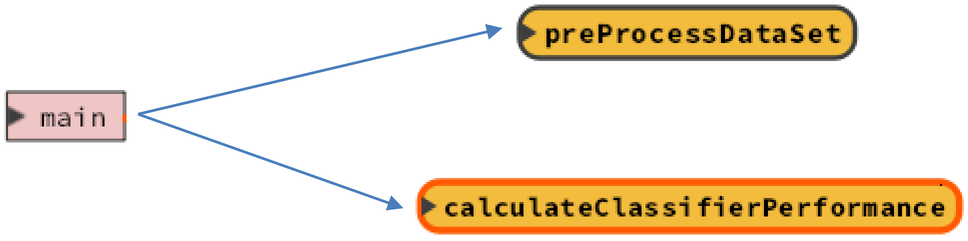
בחלקים הבאים אציג את מבנה הקבצים, וכן אפרוט את עיקר האנקדוטות בהקשרי הפונקציות הממומשות לטובת הפרויקט.

6.4.1. Main

קובץ זה אחראי על קבלת דגלי ההרצה מהמשתמש, ועל בסיסם להפעיל את הלוגיקה הנדרשת בספריית הטיפול ב dataset.

הקובץ מממש Argument Parser פשוט שתפקידו להציג למשתמש help להצגת אופן השימוש בתוכנה, וכן את הפרמטרים האפשריים לשימוש.

התוכנית הראשית משתמשת ב2 פונקציות עיקריות של DatasetHandler :



איור 21: זרימת התוכנית הראשית

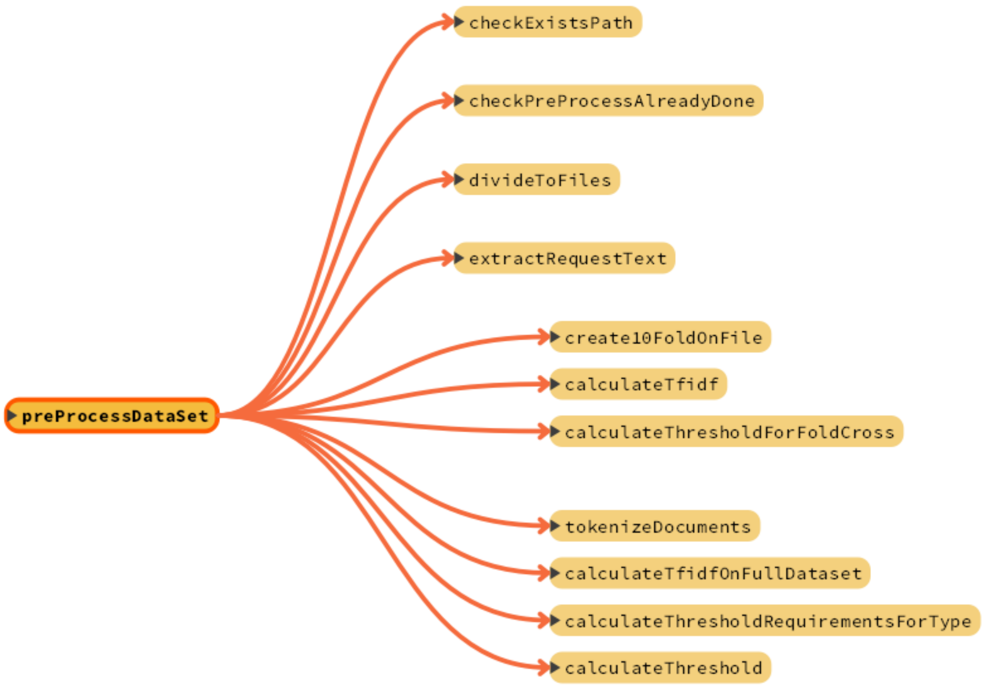
הפרמטרים איתם התוכנית הראשית קוראת לפונקציות אלו תלויה בערכים המוזנים ע״י המשתמש כarguments להרצה

6.4.2. DatasetHandler

קובץ זה אחראי על מימוש כלל הפונקציות הנדרשות לאתחול המסווג והפעלתו. ספריה זו מספקת את הAPI כלפי התוכנית הראשית כמתואר בסעיף מעלה. לטובת מימוש הAPI הנ״ל, הספרייה כוללת מספר רב של פונקציות עזר, ותתי מתודות לטובת המימושים הנדרשים. נפרט בהרחבה את הפונקציות שבשימוש על ידי התוכנית הראשית, וכתוצאה מכך את עיקר פונקציות העזר לביצוע הלוגיקה.

6.4.2.1. preProcessDataSet

פונקציה זו אחראית לביצוע שלב הלמידה לפני הפעלת המסווג על סט המבחן. כחלק מהלמידה, הפונקציה אחראית על כל הפעולות הנדרשות בעיבוד המקדים של הdataset לטובת הלמידה :



איור 22 :תהליך העבודה בשלב הpreProcessDataSet

כפי שניתן לראות באיור [22], הפונקציה מבצעת את הפעולות הבאות :

[1] בדיקת הנתיב שהגיע כקלט מהמשתמש
[2] בדיקה האם שלב העיבוד המקדים על הקלט כבר בוצע. אם לא :
- חלוקת הDataset לקבצים לפי סוג התקיפה
- הוצאת הטקסט מהxml וכתיבת קובץ הבקשות כקובץ טקסטואלי

[3] בהתאם לבקשת המשתמש, ניתן לעבוד על כל הdataset כלמידה וכסט מבחן, או לבצע 10-fold cross validation :
- באם נדרש לבצע 10-fold :
  - חלוקת הdataset המפורסר ל10 חלקים (אם לא בוצע כבר, באמצעות הבדיקה של שלב 2)
  - חישוב ערכי TFIDF על סט הלמידה
  - חישוב Threshold מתוך סט הלמידה
- באם נדרש לבצע עבודה על כלל הdataset :
  - ניתן כבר לבצע tokenize לבקשות (כפי שמתואר באלגוריתם)
  - חישוב ערכי הTf-idf על כלל הdataset
  - חישוב הThreshold :

- חישוב רשימות predicted ו target לפי סוג התקיפה
- איחוד התוצאות וחישוב Threshold

העבודה על הdataset לטובת הלמידה והסיווג הינה עבודה הלוקחת זמן חישוב רב. כתוצאה מכך, החשיבות בכתיבת הפרויקט הינה היכולת לבצע מקבול של החישובים לטובת האצת זמן הריצה.

- פעולות של העיבוד המקדים על הxml נדרשות לביצוע רק פעם אחת במהלך חיי הפרויקט. לכן, הבדיקה בשלב [2] אחראית על קיום הקבצים הנוצרים כחלק משלב העיבוד המקדים – הבקשות המחולקות לקבצים לפי סוג תקיפה, בתצורה טקסטואלית.

  *נשים לב שהבדיקה מוודאת הן שהקבצים קיימים, והן שאינם ריקים, שכן באם מריצים את התוכנה בכמה תהליכים (instances), אז התהליך הראשון מייצר את הקובץ אך עד שיסיים הוא יישאר ריק, ולא נרצה לעבוד עם קבצים ריקים (כלומר dataset ריק)

- פעולת הבדיקה על חלוקת הdataset אפשרית רק למצב בו אין shuffling למידע, שכן בכל הרצת shuffling, הערבוב מבוצע בצורה רנדומלית

<u>devideToFiles ו extractRequestText</u>

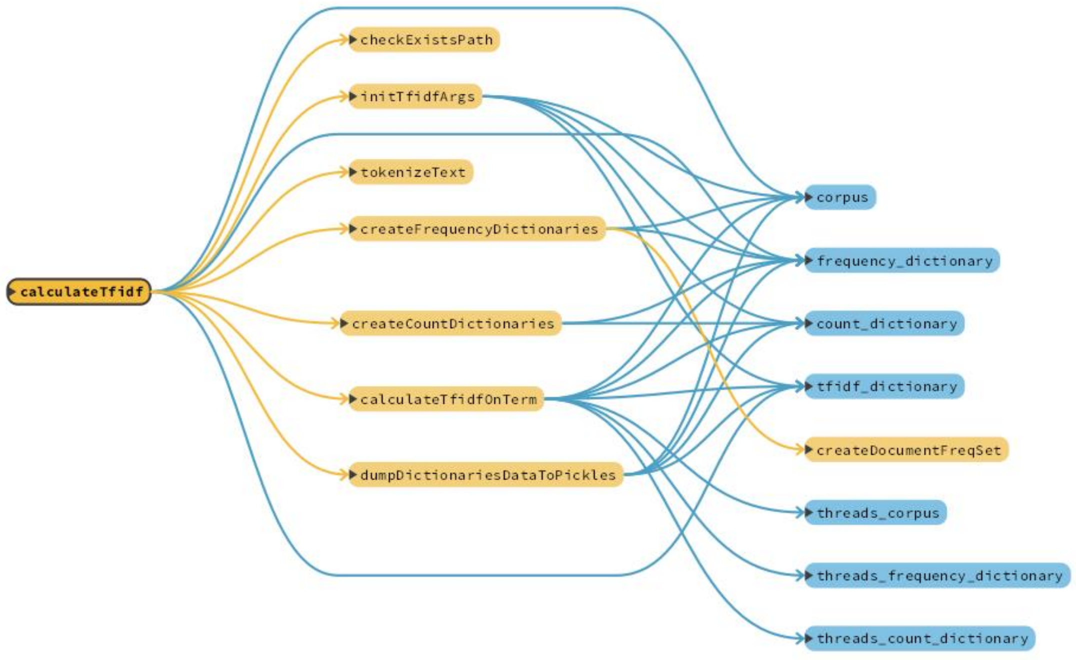
פונקציות אלו אחראיות על העיבוד המקדים הנדרש על קובץ הdataset המקורי, המגיע בפורמט xml.

לטובת החלוקה לקבצי xml לפי סוג הקובץ, הפונקציה devideToFiles משתמשת בספריה לפרסור xmlים, ובאמצעותה עוברת על כל הsamples בקובץ, ולפי הסיווג השמור באלמנט תחת הsample כותבת את האלמנט מחדש בקובץ המתאים.

הפונקציה extractRequestText בונה את בקשת הHTTP כפי שהיא נראית בשליחה תקשורתית. לטובת הבניה מחדש, הפונקציה עוברת על כל האלמנטים, שולפת את הטקסט מתוכם, וכותבת אותם לפי הסדר הנדרש : method, uri, query, protocol, headers.

<u>calculateTfidf ו calculateTfidfOnFullDataset</u>

פונקציות אלו אחראיות על חישוב ערכי הtfidf עבור כל מונח (term) בdataset.



איור 23:תהליך העבודה בעת חישוב ערכי הTFIDF

באיור [23] ניתן לראות את תרשים הזרימה של הפונקציה – הפונקציות שהיא קוראת להם (בצהוב), ומשתני העזר הגלובליים לטובת המסווג (בכחול).

בתחילת המתודה, מאתחלים את משתני העזר למסווג – המילונים (dictionaries) אשר שומרים את המידע הרלוונטי לטובת חישובי הtf-idf וביצוע הסיווג:

- Corpus – אוסף המסמכים (מאגר כלל הבקשות שאיתם לומדים)
- Frequency_dictionary – מילון הממפה בין מונח לבין מספר המסמכים שהמונח מופיע בהם
- Count_dictionary – מילון הממפה את מספר המופעים של מונח ב corpus
- Tfidf_dictionary – מילון הממפה בין מונח לבין ערך ה tf-idf שלו

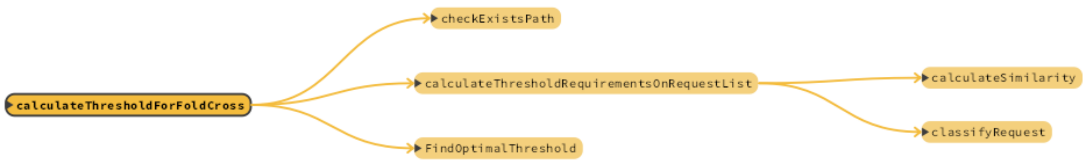
משתנים אלו מוגדרים בglobal scope כיוון שהם נדרשים במגוון רחב של פונקציות בספריה (לכל scope הספרייה בעצם), וכן זו דרך הרבה יותר קלה ונוחה להעביר תוצאות חישוב בין פונקציות מבלי להכניס את משתנים אלו כפרמטרים לכל קריאה לפונקציה.

הפונקציה calculateTfidf קוראת לפונקציות האחראיות לחישוב הערכים לטובת אותם מילוני עזר, ולבסוף עובדת על חישוב ערכי הtf-idf לטובת עדכון מילון ה tfidf_dictionary, באמצעות פונקציה האחראית לחישוב ערך tfidf עבור מונח בודד (בהתבסס על המילונים המוזכרים לעיל)

בסוף ריצת הפונקציה, הפונקציה כותבת לקובץ pickle את המילונים אותם חישבה, לטובת יכולת טעינה מהירה. פוקנצית ה preProcess תטען קובץ זה אם הוא קיים בתחילת ריצה של התוכנית.

<u>calculateThresholdForFoldCross</u>

פונקציה זו אחראית על חישוב הthreshold עבור ריצת ה fold-cross הספציפית (עבור אותו אינדקס שנבחר כtest)



איור 24: תהליך חישוב ה threshold בעבודה עם fold-cross

הפונקציה קוראת לתת פונקציה – calculateThresholdRequirementsOnRequestsList שתפקידה חישוב רשימות ה target והpredicted עבור רשימת requests המגיעה כקלט לפונקציה.

האופן בו מחושבים הערכים לטובת הרשימות הנ"ל הוא באמצעות פונקציותcalculateSimilarity ו classifyRequest שמטרתם חישוב הדמיון לסיווג כלשהו, וקבלת הסיווג המתאים ביותר (והאלפא שחושבה כחלק מהסיווג הנ"ל).

בסיום ריצת פונקצית עזר זו, ניתן לקרוא לפונקצית העזר שתפקידה לחשב את הthreshold עבור רשימות ה target ו predicted. החישוב מבוצע באמצעות ספריות חיצוניות כגון pandas, sklearn.

נקודה מעניינת לגבי calculateSimilarity – ניתן לקבל **<u>דמיון אפסי לכלל הסיווגים</u>**. כיצד יתכן?

סיטואציה זו יכולה לקרות כאשר מבצעים חלוקה של הdataset, ובעיקר כאשר לא מבצעים shuffle לdataset. נוכל להגיע למקרים בהם מונחים כלשהם חשובים קיימים רק בחלק של הtest ולא מופיעים כלל בחלק של ה learning set. בריצת הפרויקט ללא shuffle ניתקל בסיטואציות כאלו, ויודפס האינדקס שעבורו לא נמצא שום דמיון.

המאמר <u>לא מתייחס כלל לסיטואציות כאלו,</u> אך השאלה המעניינת – איך לסווג הודעות כאלו? האם לבצע בחירה רנדומלית, או שניתן לחשוב על אופציה טובה יותר?

מסקירה שלי, הדרך הטובה ביותר היא לסווג את ההודעות האלו <u>כתקינות-Valid</u> מהסיבות הבאות:

- הודעות תקינות יותר "מגוונות" מהודעות תקיפה. על הודעות תקיפה, המסווג לומד מונחים חשובים לעולם התקיפה. על עולם ה"תקין" **אין מונחים בעלי ערכי tf-idf גבוהים**, שכן רק במקרה הdataset הכיל רצפים שקצת חוזרים על עצמם. לכן בעת עבודה על דוגמאות מבחן חדשות, יש סיכוי גבוה שהמסווג יתקל במונחים שלא נתקל בהם. אם הdataset שאיתו עבד ללמידה היה מספיק "בשרי", מונחי תקיפה כבר היו אמורים להיכנס למילון ה tf-idf שלו. לעומת זאת, מונחים "תקינים" יכולים להשתנות כל רגע.

- מסווג בעולם האמיתי יעדיף FN על FP, שכן חווית המשתמש חשובה, ולכן אם אין ביטחון על סיווג הודעה, עדיף לתת לה לעבור ולא להפילה ובטעות לחסום פעולה קריטית במערכת המשלבת

<u>calculateThresholdRequirementsForType ו calculateThreshold</u>

פונקציות אלו אחראיות על חישוב ה Threshold בעת עבודה על כלל הdataset. כיוון שבמצב זה כמות הsamples שעובדים איתם גבוהה מאוד, נרצה למקבל את העבודה ולקבל ניצול משאבים אופטימלי. לשם כך, החישוב משתמש בthreads לטובת חלוקת העבודה בין כמה workers.

סדר הפעולות בין הפונקציות – calculateThresholdRequirementsForType מחשב עבור סוג התקיפה המבוקש את הtarget ו הpredicted באמצעות threads לרשימות הנגישות לthreads כמשתנים גלובליים.

כשכלל החישובים עבור כלל סוגי התקיפות הסתיימו, הפונקציה calculateThreshold מאחדת את הרשימות שחושבו על ידי כל thread, כך שניתן יהיה לחשב את ה threshold הכללי.

<u>אופן העבודה עם threads:</u>

חלוקת העבודה בין ה threads מבוצעת על ידי חלוקת ה requests_list לX חלקים, כאשר X הוא מספר הthreads (מוגדר X=8 דיפולטית).

כאשר thread_i ירצה לבצע את עבודתו ולחשב את target ו predicted לחלק שלו, הוא יצטרך גישה למילונים של tfidf,count,frequency והCorpus עצמו לטובת עדכון שלו פר בקשה חדשה.

כיוון שיש הן קריאה והן כתיבה למילונים על ידי כמה threads במקביל, מימוש נאיבי לסיטואציה זו ישתמש בmutexים (מנעולים) לטובת הגבלת הגישה וסנכרון בין הthreads.

עם זאת, עבודה עם mutexים בצורה זו תגרום לכך שבפועל לא נעבוד באמת במקביל – שכן כל פעם רק thread אחד יקבל "אסימון" לטיפול בבקשה בודדת, ועד שלא יסיים, כלל הthreads האחרים יחכו לשחרור המנעול. כלומר, בפועל נקבל עבודה טורית, פשוט על ידי כמה workers.

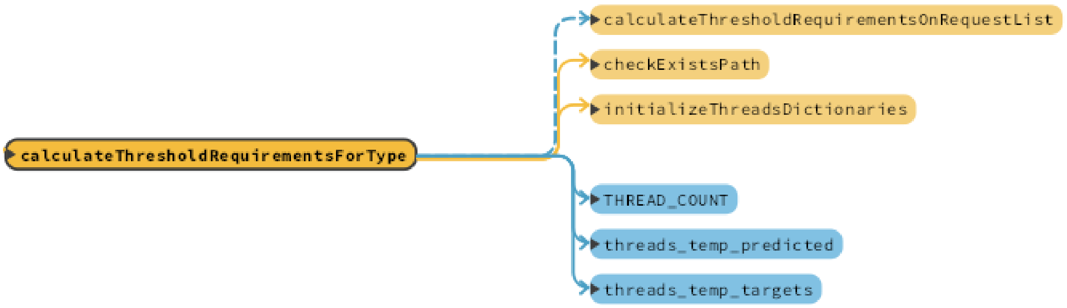
על מנת להתגבר על מכשול זה, החלטתי לשכפל את כלל מילוני העזר לטובת הthreads, כך שלכל אחד יש אוסף מילונים משלו. כלומר, מבצעים העתקה מלאה של כלל המילונים, כפול כמות הthreads הקיימים. אמנם גישה זו גורמת לצריכת זיכרון RAM גבוה יותר בכל ריצה, אך זה tradeoff עדיף ומנצל משאבים טוב יותר.

*בעת העבודה עם threads זיהיתי שכמות הליבות הפיזיות במערכת שעליה התוכנית רצה <u>נשאר 1, למרות שהגדרנו 8 threads.</u> איך זה קורה?

לצערנו, במימוש הפנימי של Python Interpreter (CPython), אין אפשרות להרצה מקבילית אמיתית (parallel ולא רק concurrent) של threads בעקבות Global Interpreter Lock (GIL).

יש צורך בGIL כיוון שהPython Interpreter עצמו <u>אינו thread-safe</u>. לכן בכל גישה ל Python Object או C API רק thread אחד יכול להשיג מנעול ולעבוד.

לטובת ניצול משאבים אמיתי, יש להריץ כמה Process של התוכנית. אופן המימוש לטובת הרצת כמה Processים מבוצע בפרויקט על ידי אפשור ההרצה של כמה instances של התוכנית, וחלוקה לוגית של העבודה (לדוגמה עבודה על אינדקס בודד מהfold 10 בקריאת המערכת) כך שניתן להריץ כמה תהליכי מערכת שהמחשב יכול להריץ (מגבלת ליבות פיזיות / מגבלת RAM.



איור 25: תהליך חישוב ה threshold בעבודה עם threads

מאיור [25] ניתן לראות את העבודה עם הרשימות הזמניות של predicted ו target של ה threads, וכן את הקריאה ל calculateThresholdRequirementsOnRequestList שכל Thread אחראי עליו.

6.4.2.2. calculateClassifierPerformance

פונקציית הAPI שמטרתה לבצע את הסיווג על הtest set, ובכך להשלים את הסיווג ולחשב את תוצאותיו.

איור 26: תהליך חישוב תוצאת המסווג

התהליך מבוצע עבור כל אחד מסוגי הסיווגים (calculateClassifierPerformanceOnType), ותוצאות הסיווג נכתבות בקובץ חדש עבור כל סוג תקיפה. החישוב מבוצע בתצורה הבאה:

1. עבור כל בקשה בסט הבקשות בצע:
   a. קבל את הסיווג המוצע על ידי המסווג באמצעות classifyRequest
   b. אם הסיווג המוצע אינו נכון, כתוב לקובץ את המספר הסידורי של הבקשה הבעייתית, את הסיווג שהמסווג נתן לו, ואת הביטחון (הדמיון) שהתקבל מהמסווג
2. לבסוף, כתוב את כמות הבקשות בסט הבדיקה שהיו בסוג תקיפה זה, ואת כמות הבקשות שלא סווגו נכון.

7. תוצאות ומסקנות

את תוצאות המסווג אנו מקבלים בקבצי הטקסט הסופיים, כפי שמפורטים ב *תוצאת ההרצה* ראשית, נגדיר מושגים קריטיים ([17]) לבדיקת התוצאות:

<u>True Positive</u>

תוצאת המסווג קבעה שהדגימה שייכת למחלקה, והדגימה אכן שייכת למחלקה.

<u>True Negative</u>

תוצאת המסווג קבעה שהדגימה לא שייכת למחלקה, והדגימה אכן לא שייכת למחלקה.

<u>False Positive</u>

תוצאת המסווג קבעה שהדגימה שייכת למחלקה, והדגימה <u>לא</u> שייכת למחלקה.

<u>False Negative</u>

תוצאת המסווג קבעה שהדגימה לא שייכת למחלקה, והדגימה <u>אכן</u> שייכת למחלקה.

	Predicted class		
Actual Class		Class = Yes	Class = No
	Class = Yes	True Positive	False Negative
	Class = No	False Positive	True Negative

איור 27: הגדרות תוצאת הסיווג

<u>Accuracy</u>

אחוז הסיווגים התואמים של המסווג. במילים אחרות: $\frac{TP+TN}{TP+FP+FN+TN}$

<u>Precision</u>

אחוז הסיווגים הנכונים שעבורם המסווג אמר "כן" שייכים למחלקה. במילים אחרות: $\frac{TP}{TP+FP}$

<u>Recall</u>

ידוע גם בתור "רגישות" (sensitivity). אחוז הסיווגים הנכונים כ"כן", מתוך כלל הסיווגים השייכים למחלקה. במילים אחרות: $\frac{TP}{TP+FN}$

<u>F1 Score</u>

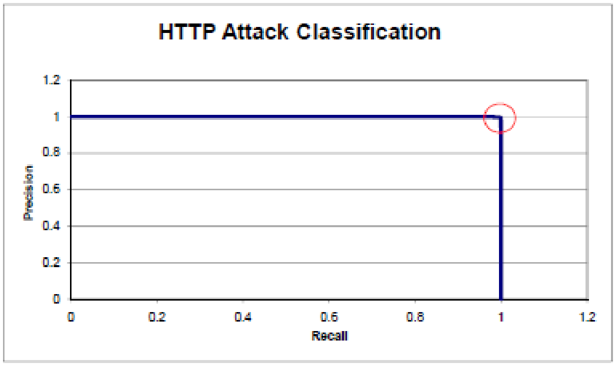
ממוצע משוקלל של ערכי ה Precision וה Recall. כלומר, המדד הנ"ל לוקח בחשבון הן FP והן FN.

אופן החישוב:

$$F1\ Score = \frac{2*(Recall\ *\ Precision)}{(Recall\ +\ Precision)}$$

כעת, נזכר בתוצאות המסווג המוצג במאמר על סט האימון :

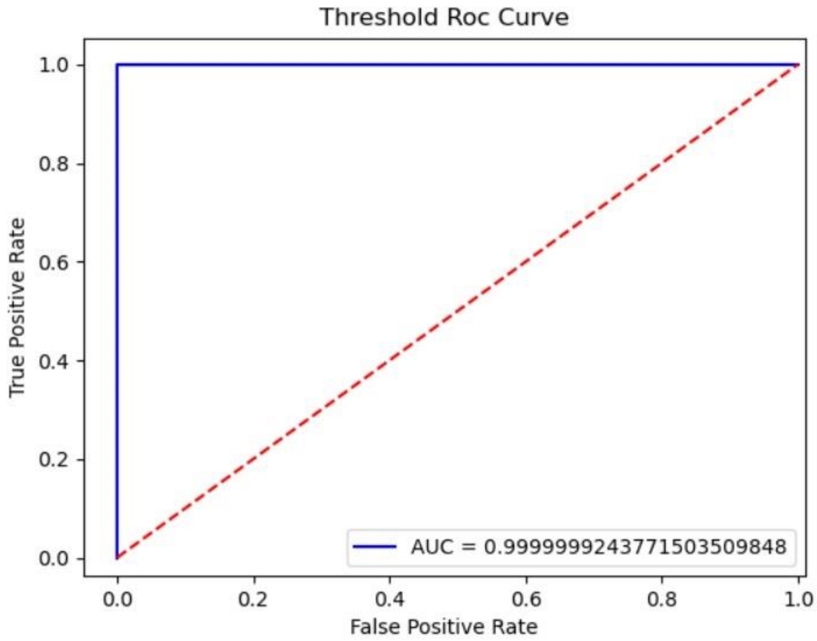
	Classifier		
	tf-idf	LM	C4
Training Set Only			
Accuracy	> 0.99	-	0.77
AUC	> 0.99	-	-
Precision	> 0.99	0.98	-
Recall	> 0.99	0.93	-
F1	> 0.99	0.96	-
Discovery Challenge (Train/Test)			
Accuracy	0.94	-	-
AUC	0.97	-	-
Precision	0.98	0.82	0.50
Recall	0.88	0.78	0.47
F1	0.93	0.80	0.48



איור 28: תוצאות המסווג במאמר

ניתן לראות אחוז הצלחה של מעל 99 אחוז בעבודת המסווג על כלל הTraining set.

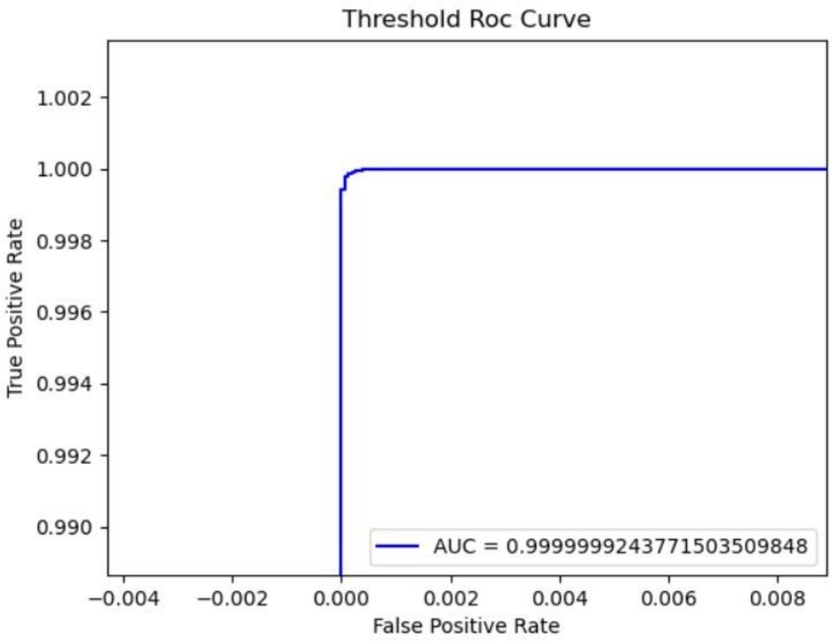
על מנת לקבל את נתוני ההצלחה על כלל הdataset, השתמשתי בכלי גרפי (matplotlib) להצגת תוצאת חישוב ה Threshold, שכן בעת עבודה על כלל הdataset ניתן לגלות את הAuc שלנו בעת חישוב הThreshold :



איור 29: תוצאות המסווג שלנו על כלל ה dataset

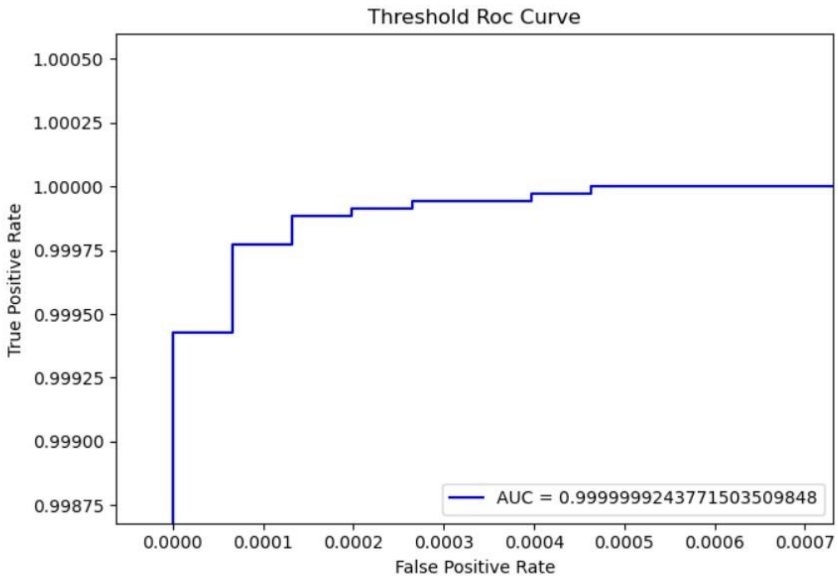
ניתן לראות כי אנחנו מקבלים AUC **של מעל 0.99.**

הthreshold שנבחר הוא בנקודת האופטימום של מקסימום TP ומינימום FP. במבט מהיר נראה שהגרף הוא "מדרגה" אחידה, אך בעת zoom ראשוני ניתן כבר לראות את הגרף הבא:



איור 30: תקריב גרף תוצאת המסווג

כעת כבר ניתן לראות את הtradeoff בין ה TP לבין הFP. ביצוע תקריב נוסף כבר מציג את הגרף בצורה הרבה יותר ברורה:



איור 31: תקריב עמוק יותר על גרף תוצאת המסווג

כעת ניתן לראות את המדרגות בין קבלת TP גבוה, והגדילה בFP. תפקיד פונקצית חישוב הthreshold הינו קבלת AUC מקסימלי.

כלומר, ניתן לראות שגם המסווג שאנו כתבנו מציג ביצועי סיווג זהים (ומרשימים) עבור עבודה על כלל ה Dataset.

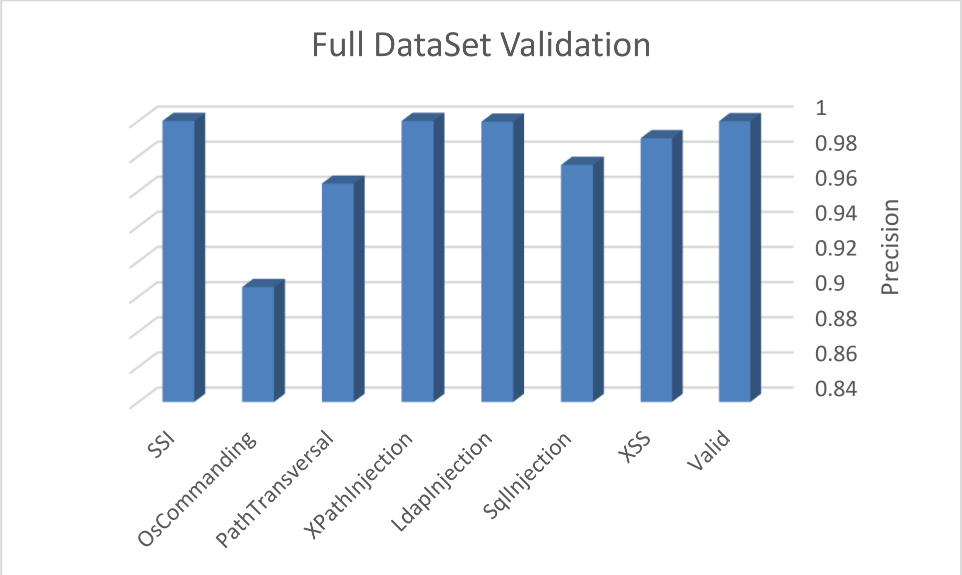
הthreshold שחושב עבור הרצת המסווג על כלל ה dataset הינו 0.34225884967011205, כלומר מספיק "ביטחון" לא גבוה. אפשר להבין שערך זה אינו גבוה, שכן כשמסתכלים על ערכי הtf-idf עבור סיווג ה Valid אנו רואים מונחים עם ערכים נמוכים משמעותית מערכי tf-idf לשאר הסיווגים. עובדה זו נובעת מכך שאין ממש מונחים שמקשרים בצורה חזקה דוגמא לValid, אלא יותר מונחים שמרחיקים את הדוגמה מValid לכיוון תקיפה כלשהי.

כלל התוצאות (ואינפורמציה נלווית) נכתבות לקבצי טקסט כפי שהוסבר למעלה. בחלקים הבאים אסכם ואציג את התוצאות שהתקבלו בצורה גרפית. ניתן לראות את הנתונים שעל בסיסם נבנו הגרפים בקובץ Excel נלווה לפרויקט. תוצאות המסווג לפי סוגי התקיפות המוצגות בגרפים הן ערכי הPrecision (דיוק) שהושגו.

7.1. תוצאות סיווג על כלל הDataset

במבחן זה, נבדוק את יכולת המסווג ללמוד מידע חשוב לטובת סיווג הבקשות **לסוג התקיפה הספציפי**. נשים לב שזו בקשה "הדוקה" יותר מאשר סיווג כתקין או לא תקין.

ניתן "לרחף" עם העכבר מעל העמודה הרלוונטית לקבלת המספר המדויק אותו הוא מייצג

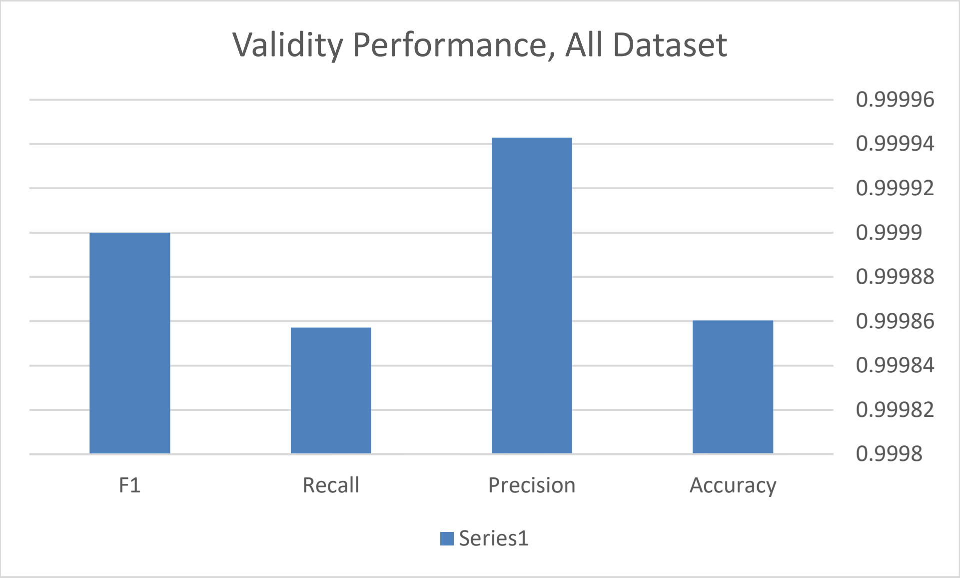


איור 32: תוצאות המסווג בעבודה על כלל הDataset

כפי שניתן לראות, המסווג מציג תוצאות של מעל 95% אחוז סיווג מדויק בכל סוגי התקיפות, למעט OsCommanding. צריך לזכור שסט המידע לא גדול, כך שלדוגמה עבור סוג תקיפה זו קיימים סה"כ 2303 דגימות, ועדיין תוצאת המסווג היא קצת מעל 90%.

ניתן לראות שבכל הקשור לסיווג מידע כתקין, המסווג מציג תוצאה כמעט מושלמת (כפי שצפינו מתוצאת בחירת הThreshold) – **מעל ל 99.98% דיוק.**

בבחינת פרמטרי Accuracy, Precision, Recall, F1 על המסווג ביכולתו לסווג מידע זדוני ומידע לגיטימי, נקבל את התוצאה הבאה:



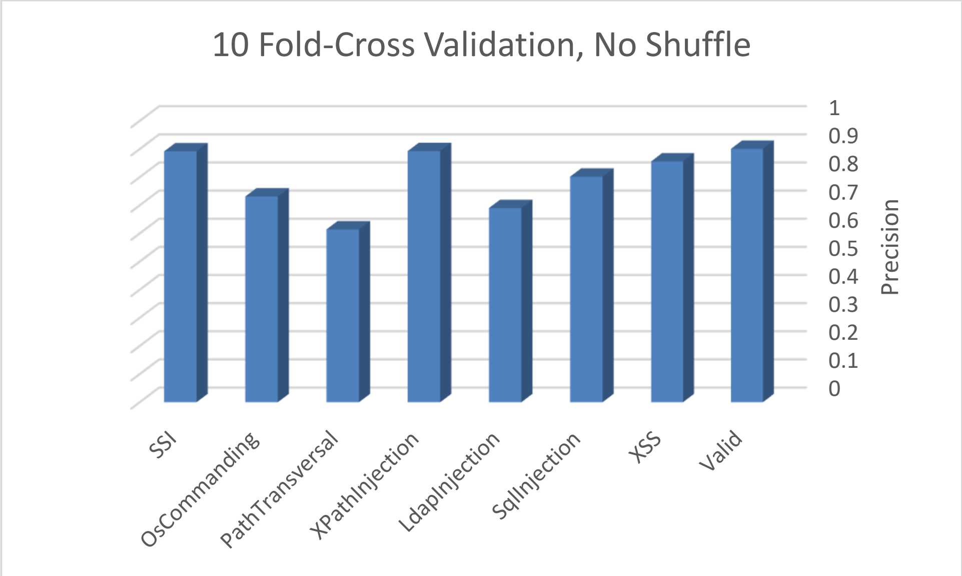
איור 33: תוצאות המסווג על כלל ה Dataset לזדוני/תקין

ניתן לראות שעבור כלל הפרמטרים מתקבלת תוצאה של מעל 99% בעבודה על כלל הDataset. כלומר, בשימוש סטנדרטי של המסווג בעולם ההגנה, שבו יש חשיבות לעצירת מתקפה יותר מאשר לסיווגה בReal-time לסוג התקיפה הספציפי, מקבלים מסווג עם שגיאה מינימלית בהחלט!

7.2. תוצאות סיווג על 10-Fold Cross Validation

תצורת עבודה זו הרצתי ב2 ואריאציות: אחת המחלקת בצורה נאיבית את הDataset ל10 חלקים, ומבצע כל פעם עבודה על חלק אחר, ותצורה בה אני מבצע shuffle למידע לפני החלוקה. הרעיון בביצוע הshuffle הוא להוריד את הסבירות שלא אקבל בסט הלמידה שלי את המונחים הקריטיים, או שאתקל יותר במונחים שלא נתקלתי בהם ולא אקבל דמיון עבורם (כפי שתואר בפסקאות קודמות).

כותבי המאמר לא התייחסו לסוגיה זו, אך נראה שיש לה השפעה גדולה מאוד בתוצאות המסווג.

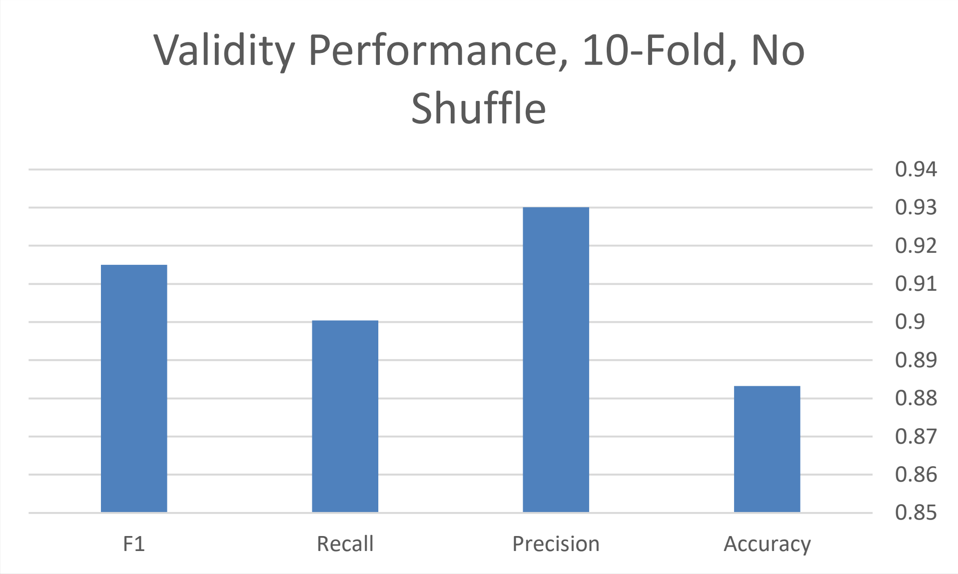


איור 34: תוצאות המסווג בהרצת 10-Fold ללא shuffle

כפי שניתן לראות באיור [34], המסווג מציג תוצאות יפות, אך לא מרשימות כפי שקיבלנו בעבודה על כלל הdataset. ניתן להסביר נקודה זו בכך שאנחנו מאבדים 10% מסט הלמידה שלנו בכל סבב, ועבור חלק מסוגי התקיפות 10% שווים לקצת מעל 200 דוגמאות.

אם נזכור מניתוח המאמר את מיקוד המונחים לכל סוג תקיפה, ואת תוצאות המסווג עם מילון מצומצם (איור [17]) – לכל סוג תקיפה כמות מונחים יחסית קטנה בעלי משמעות גדולה. זאת אומרת שאם נפספס את המונחים הללו בעת הלמידה, וניתקל בהם רק בשלב המבחן, נפגע משמעותית ביכולת המסווג שלנו לסווג נכונה לסוג התקיפה.

כעת נבחן את תוצאת המסווג בתצורה זו עבור סיווג של "זדוני/תקין", שכן אמנם המסווג יכול לאבד מידע קריטי לטובת סיווג התקיפה הספציפית, אך השפעות החוסר הללו על הסיווג הכללי יכולות להיות שונות – לדוגמה "בלבול" בין סוגי injection שונים.



איור 35: תוצאות המסווג בהרצת 10-Fold ללא shuffle לזדוני/תקין

ניתן לראות שאנו מקבלים תוצאות נמוכות בהרבה מעבודה על כלל הDataset, והעלנו מספר השערות לסיבות המשפיעות על השוני המתקבל. נשים לב שעדיין תוצאת המסווג לזדוני/תקין נחשבת טובה, שכן הדיוק הוא בסביבות ה 93% הצלחה, ותוצאת הF1 המשקללת היא מעל 90% הצלחה – כלומר עדיין מתקבל מסווג שניתן להשתמש בו בעולם האמיתי.

כפי שתיארתי למעלה, המסווג מושפע מאוד מהקלט בשלב הלמידה, ובאם יאבד מידע קריטי ללמידה, תוצאותיו יכולות להיפגע מאוד. נדגים זאת בבחינה על תוצאות המסווג על XSS בתצורה של 10-Fold ללא shuffle למידע. הטבלה באיור [36] נלקחה מקובץ הExcel הנלווה לפרויקט:

index	Total	TRUE	FALSE
0	184	92	92
1	184	151	33
2	184	70	114
3	184	182	2
4	184	181	3
5	184	181	3
6	183	174	9
7	183	180	3
8	183	179	4
9	182	180	2

איור 36: תוצאות המסווג בהרצת 10-Fold ללא shuffle על XSS

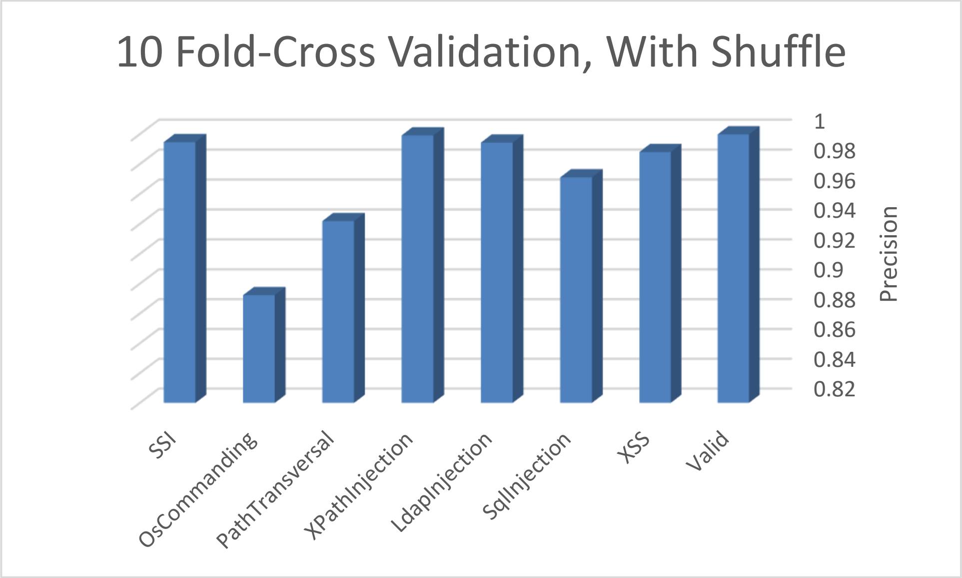
בצד שמאל זהו האינדקס בהרצת ה 10-Fold Cross, לאחריו כמות הדגימות באינדקס זה (נזכיר שאינדקס זה מהווה את האינדקס שנבחר להיות סט הבחינה, כל השאר הופכים להיות סט הלמידה), כמות הדגימות שסווגו נכונה, וכמות הדגימות שסווגו לא נכון (כל סיווג שאינו XSS במקרה זה).

אפשר לראות שעבור שלושת האינדקסים הראשונים מתקבלים תוצאות גרועות מאוד ביחס לשאר האינדקסים. הסיבה לכך, היא שב30% הראשונים בסט הדגימה נמצאים הפרמטרים הקריטיים ללמידה על XSS, ובאם איבדנו אותם והפכנו אותם רק לtest, אז נקבל הרבה שגיאות סיווג.

מתוצאה זו אנו למדים שכדאי לבצע חלוקה "שיוונית בלוגיקה" בdataset בעת החלוקה – ואת זה נבצע עם shuffle למידע לפני ההרצה.

7.3. תוצאות סיווג על 10-Fold עם shuffling

כעת שאנו יודעים שיש חשיבות גבוהה ל"תפיסת" הדגימות הקריטיות בלמידה של כל סוג תקיפה, אנו מצפים לתוצאות טובות יותר בעת ההרצה, ואף להתקרב מאוד לתוצאת המסווג על כלל סט הלמידה.



איור 37:תוצאות המסווג בהרצת 10-Fold עם shuffle

ניתן לראות שיפור משמעותי בדיוק המסווג על סט המבחן, כאשר החלוקה ל10-Fold מבוצעת עם ערבול המידע לפני החלוקה.

לא רק שהתוצאות השתפרו משמעותית ביחס להרצה הנאיבית של ה 10-Fold, אלא קיבלנו תוצאות **כמעט זהות לחלוטין** להרצה של כל הdataset ללמידה ולמבחן (איור [32])

הסיבה לתוצאות המרשימות האלו נובעת מהנקודה שהעלנו קודם – כל סוג תקיפה ניתן לסווג עם כמות יחסית קטנה, אך קריטית, של מונחים. בחלוקה רנדומלית של סט המידע אנו מגיעים למצב בו לא נפספס מונחים קריטיים בשלב הלמידה.

ניתן לראות זאת בהשוואת בחינת המסווג על דגימות מסוג XSS – הפעם עם חלוקה רנדומלית (כלומר השוואה לטבלה באיור [36]) :

index	Total	TRUE	FALSE
0	183	182	1
1	184	182	2
2	184	183	1
3	184	181	3
4	184	183	1
5	184	181	3
6	183	180	3
7	183	181	2
8	183	178	5
9	183	182	1

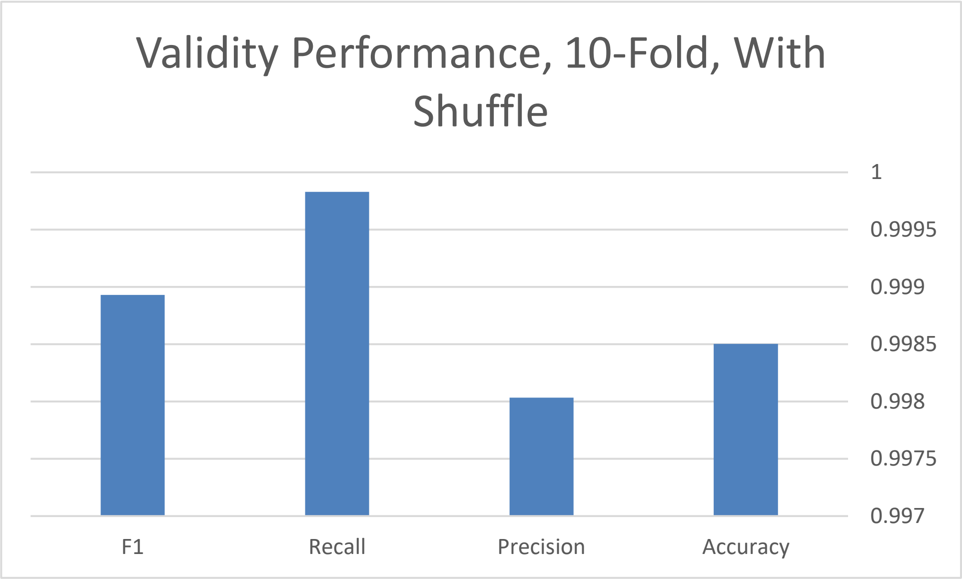
איור 38: תוצאות המסווג בהרצת 10-Fold עם shuffle על XSS

בניגוד לטבלה באיור [36], בהרצה הזו אפשר לראות שהסיווגים השגויים:

- מועטים מאוד
- מפוזרים בצורה אחידה על האינדקסים

כלומר, הסיווגים השגויים נובעים מיכולת המסווג לקלוט עוד פרמטרים לסיווג התקיפה, ולא מלמידה חלקית ביחס ל dataset.

תוצאות המסווג לטובת זיהוי "זדוני/תקין":



איור 39: תוצאות המסווג בהרצת 10-Fold עם shuffle לזדוני/תקין

כפי שניתן לראות, חזרנו לתוצאות של **מעל 99% בכלל הפרמטרים**!

תוצאות אלו מראות שעם עבודה נכונה על הdataset ניתן להגיע לתוצאות גבוהות, כמעט כמו עבודה עם כלל הdataset. כמו כן, נתונים אלו מראים שהאלגוריתם העומד בבסיס המסווג "מנצל" את המידע שאיתו אפשר ללמוד על סוגי התקיפות מתוך הdataset שמתקבל. כלומר, הפרמטרים הקריטיים ללמידה ולסיווג בהינתן dataset זה, נלמדו בהצלחה מרובה במסווג הנ"ל.

7.4. המונחים המשמעותיים ביותר לסיווג

הטבלה באיור [40] מציגה את 5 המונחים המשמעותיים ביותר שהמסווג למד על כל סוג תקיפה, כאשר הוא עבד על כלל הDataset :

Attack Type	Term	Tfidf Score
LDAP Injection	"NetscapeRoot"	0.0009010848425641216
	"objectClass"	0.0017468129371208912
	"*o"	0.0018452249335784061
	"brien*"	0.0018452249335784061
	"had*"	0.002694028403024473
Command Execution	"2e"	0.0009644316601060131
	"2f"	0.0009644316601060131
	"../winnt/system32/cmd.exe/c"	0.0009962610878322842
	".."	0.0022959627199883527
	"dir"	0.002680037814552023
Path Traversal	"file"	0.0003440039453139226
	"WINDOWS"	0.000818601980978501
	"WINNT"	0.0008854916370117636
	"."	0.004580825904940252
	".."	0.009162821135985037
SSI Attack	"include"	0.0008412641978104817
	"odbc"	0.0009021877991927508
	"statement"	0.0009021877991927508
	"virtual"	0.0016825283956209635
	"--"	0.0028847091833887886
SQL Injection	"UNION"	0.0009437028020794941
	"OR"	0.0010139460527197153
	"WHERE"	0.001020054161471039
	"FROM"	0.0011768289527550112
	"**"	0.0019942975073071506
XPATH Injection	"position"	0.0011139892347155765
	"text"	0.0012137383438117783
	"count"	0.002538376792929209
	"path"	0.0027271576756985436
	"child"	0.002782339779892657
Cross-Site Scripting	"http"	0.001201048448917709
	"document.location.replace"	0.0015726843419229836
	"javascript"	0.001628060551145624
	"alert"	0.002071070224926746
	"document.cookie"	0.003152752178408986
Valid (No Attack)	"13.228.134.190"	2.5337224725590385e-05
	"27260320301"	2.5533637320362403e-05
	"9055,045,32"	2.6122875104678458e-05
	"213.191.153.150"	2.6908525483766534e-05
	"13224"	2.8872651431486716e-05

איור 40:המונחים המשמעותיים ביותר של כל סוג תקיפה

ניתן לזהות שהמסווג מוצא מונחים מוכרים וקריטיים מכל סוג תקיפה – כגון document.cookie במסגרת תקיפת XSS (ניסיון לגנוב את עוגיית הקורבן לטובת גניבת הזהות שלו), שימוש במונחים מעולם הSQL – FROM,WHERE וכו', ".." בניסיון לחזור תיקיה אחורה במסגרת Path Traversal ועוד.

כמו כן, ניתן לראות שעבור קטגורית ה Valid לא מתקבלים ציונים גבוהים כלל, שכן אין קשר חזק, והופעת הקו 13.228.134.190 בכמות יחסית גבוהה הינו צירוף מקרים שהמסווג יודע לקלוט כבעל משמעות נמוכה לValid. עם זאת, בהינתן תקיפה חדשה שלא תכיל את אחד מהמונחים המוכרים

במסווג, וכן תכיל את כתובת הIP עלולה להשפיע על המסווג ולתת לו ביטחון גבוה יותר לכיוון ה Valid.

נקודה מעניינת נוספת, אפשר לראות שהמסווג נתן ציון גבוה למונח "http" (יש חשיבות לאותיות גדולות וקטנות במונחים) כמקושר לתקיפת XSS. מהיכרות עם עולם הHTTP (Expert Knowledge) לא היינו משייכים מונח זה <u>לאף אחד מסוגי התקיפה</u>, שכן בשימוש סביר ותקין הוא מופיע בלי בעיה.

- ניתן לראות שהמסווג נותן למונח HTTP/1.0 ולמונח HTTP/1.1 ציון 0, שכן הוא המונח מופיע <u>תמיד</u>, בכל בקשה. אך ממגבלות המסווג, הוא יודע להסתכל על המונח השלם – כלומר עם הפרוטוקול, שכן אלגוריתם הטוקניזציה שלו לא מתייחס לנקודה זו

7.5. בדיקת המסווג על דוגמאות חיצוניות

נבדוק את יכולת המסווג לבצע סיווג מוצלח לדוגמאות שאינן מהdataset כלל, שכן כפי שנכתב בחלק *8.סיכום והצעות לכיווני מחקר נוספים*, למסווג יש מגבלות שבשילוב עם dataset זה לא עלו.

<u>**בקשת XSS המסווגת בהצלחה**</u>

```
GET
/site/public/timing?%3C%21%2BXSS%3D%22%3E%20%20%20%20%3Cimg%2Bsrc%3Dxx%3Ax%2Bonerror%3Dalert%2814721850.00337%29%2F%2F%22%3E HTTP/1.1
```

איור 41:בקשה זדונית (XSS) שלא מתוך ה Dataset

הבקשה באיור [41] מאוד פשוטה, ומכילה רק את בקשת המשאב, ללא האדרים נוספים.

לאחר ביצוע טוקניזציה של המסווג נקבל את הטקסט הבא:

```
GET , /site/public/timing? , XSS , img , src , xx , x , onerror , alert , 14721850.00337 , HTTP/1.1
```

איור 42:הבקשה לאחר ביצוע טוקניזציה

כעת ניתן לראות באיור [42] את כלל הטוקנים שאיתם המסווג יבצע את החלטתו.

מתוך הרצת הפרויקט נקבל את התוצאה הבאה:

```
Request similarity for assignment Valid is: 0.00043509755149747227

Request similarity for assignment XSS is: 0.17042575511866723

Request similarity for assignment SqlInjection is: 0.0

Request similarity for assignment LdapInjection is: 0.0

Request similarity for assignment XPathInjection is: 0.0

Request similarity for assignment PathTransversal is: 0.0
```

```
Request similarity for assignment OsCommanding is: 0.0

Request similarity for assignment SSI is: 0.00011244993026721403

Normalized probability after multiplication in alpha(5.84886637147684):
0.9967974679471197

Best assignment: XSS



Top 5 terms in the request, by their tfidf value:

('alert', 0.002071070224926746)

('src', 0.0009401649743577149)

('img', 0.0007875727533886617)

('', 0.0)

('x', 0.0)
```

איור 43:תוצאת המסווג על הבקשה החדשה

באיור [43] ניתן לראות את כלל תוצאות ה"דמיון" (Similarity) לכל סוג תקיפה, את הalpha שחושבה עבור בקשה זו (הנרמול הנדרש על מנת שסכום הדמיונות [ההסתברויות] יהיה 1), ואת הסיווג הנבחר.

ניתן לראות שהסבירות לכך שהבקשה הינה מסוג XSS לאחר נרמול הינה 99.679%, כך שהמסווג מאוד בטוח בסיווג זה.

כמו כן, ניתן לראות את המונחים המשמעותיים ביותר שהופיעו בבקשה אשר השפיעו על הסיווג לתקיפה זו, וביניהם– alert, src, img אשר בעלי ערך tfidf גבוהים במילון הtfidf של XSS (המילון מופיע באיור [9]).

<u>בקשת SQLi שאינה מסווגת בהצלחה</u>

```
GET
/site/registration/create/3269?execution%3De1s1%20waitfor%20delay%270%3A0%3A3
%27%0A HTTP/1.1
```

איור 44:בקשת SQLi שאינה מהDataset

באיור [44] ניתן לראות בקשת HTTP פשוטה (ללא האדרים נוספים מעבר לבקשת המשאב).

הבקשה הזדונית משפיעה על שרת Microsoft SQL Server בכך שהיא מבקשת ממנו לחכות 3 שניות לפני החזרת התשובה. זו אינה הזרקה קלאסית, אך במידה והיא מצליחה, התוקף מבחין בקיום השרת מסוג זה, ובכך שקלט המשתמש עובר הרצה בשרת.

```
GET , /site/registration/create/3269?execution , e1s1 , waitfor , delay , 0 , 0 , 3 ,
HTTP/1.1
```

איור 45:בקשת SQLi שאינה מהDataset לאחר טוקניזציה

באיור [45] ניתן לראות את הטוקנים שאיתם המסווג עובד. ניתן לראות שהמסווג מפספס את הטוקן " execution ", שכן אלגוריתם הטוקניזציה במאמר לא מתייחסים לתו "?" כתו שאיתו מבצעים טוקניזציה, ולכן המסווג מסתכל על כל ה URI כטוקן שלם.

כמו כן, בdataset לא היו קיימים הטוקנים waitfor, delay כחלק מסיווג SQLi.

```
Request similarity for assignment Valid is: 0.0003036206267868585

Request similarity for assignment XSS is: 0.0

Request similarity for assignment SqlInjection is: 0.0

Request similarity for assignment LdapInjection is: 0.0

Request similarity for assignment XPathInjection is: 0.0

Request similarity for assignment PathTransversal is: 0.0

Request similarity for assignment OsCommanding is: 0.0

Request similarity for assignment SSI is: 0.0

Normalized probability after multiplication in alpha(3293.5838733446767): 1.0

Best assignment: Valid


Top 5 terms in the request, by their tfidf value:

('e1s1', 5.892377843160554e-07)

('', 0.0)

('0', 0.0)

('GET', 0.0)

('3', 0.0)
```

איור 46:תוצאת המסווג על בקשת הSQLi החדשה

כפי שניתן לראות באיור [46], המסווג מחליט לסווג את בקשה זו כתקינה ב100%!

ניתן לראות מתוצאת ההרצה באיור [46] שהמסווג לא מוצא דמיון כלל לאף אחת מהתקיפות, למעט לValid. הסיבה לדמיון הקיים ל Valid נובע מצירוף מקרים על המונח "e1s1" שהופיע במילון ה Valid (עם ציון נמוך מאוד), אך הופיע גם בבקשה.

המסווג לא הכיר את סוג ניצול זה כיוון שלא הופיע בשלב הלמידה, אך מומחה אבטחה היה יכול לזהות את הkeywords הבעייתיים, ולשפר את איכות המסווג באמצעות הוספה שלהם למילון העבודה של המסווג.

7.6. זמן ואופן ההרצה

כפי שתואר מעלה, פעולות העיבוד המקדים, הלמידה, והסיווג לוקחים לא מעט זמן. לשם כך הפרויקט נכתב בצורה שתאפשר ניצול משאבי מערכת מקסימליים.

נתוני המערכת: מעבד (8 cores) I7-6700HQ, 16GB RAM, 128GB SSD

הרצת סבב מלא על אינדקס בודד ב10-fold-cross (לדוגמה עבודה על חלקים 0-8, מבחן על חלק 9) לקחה בממוצע 4,055 שניות, שהם קצת מעל שעה.

בעקבות היכולת לעבוד במקביל, במערכת שלי אני מסוגל להריץ 8 עבודות כאלו במקביליות מלאה, כלומר לבצע 8 אינדקסים ב4000~ שניות.

פעולת סיווג לבקשה בודדת לוקחת בממוצע 0.4 שניות (תלוי בגודל ההודעה, מורכבות, וכו')

8. סיכום והצעות לכיווני מחקר נוספים

בפרויקט זה מימשנו הצעה יצירתית לבעיית סיווג הודעות HTTP לסוגי תקיפה ספציפיים (כלומר, נותנת מענה רחב יותר לבעיית הסיווג זדוני/תקין)

הרעיון המרכזי באלגוריתם דוגל בגישה שהודעות HTTP הינם טקסטואליים, ולכן ניתן להפעיל עליהם אלגוריתמים מעולמות ה Data-Mining שמבוצעים כיום על מסמכי טקסט חסרי קונטקסט "טכנולוגי". רעיון זה בשלב הראשוני נשמע חסר סיכוי, שכן הגישות היום לעולמות אלו כוללות "Expert Knowledge" – היכרות עם עולם התוכן, התקיפות והטכנולוגיות, לטובת ביצוע סיווג איכותי.

מה שקיבלנו במסווג הוא מעין "למידה מהירה" של עולמות התוכן – כאשר מתבוננים על ה Top Terms במילוני הTF-IDF, מבינים אילו מונחים בעלי המשמעות הגבוה ביותר לשיוך בקשה לסוג התקיפה.

בהסתכלות לאחור, גישה זו מאוד הגיונית, שכן כיוון שהודעות הHTTP הינן טקסטואליות, וביצוע התקיפה יעשה באמצעות שליחת payload זדוני <u>טקסטואלי</u>, ולכן ניתן ללמוד ולזהות אותו.

המסווג הציג תוצאות גבוהות מאוד הן בסיווג לסוג התקיפה הספציפי, ועוד יותר בסיווג הקלאסי של זדוני/תקין. המסווג מציג תוצאות המאפשרות (ביחס לdataset) הטמעה של המסווג באפליקציה אמיתית – הן מבחינת הביטחון של מטמיעי הפתרון בכמות FP נמוכים, והן בזמן הסיווג הליניארי – הדומה לעבודה שהשרת צריך לבצע לטובת בקשת הלקוח בכל מקרה.

<u>עם זאת, ברור שלמסווג זה קיימות מגבלות, וביניהן:</u>

- היכולת שלו ללמוד מונחים קריטיים תלויה באיכות ה dataset שאיתו הוא לומד
- התקיפות צריכות להשתמש ב"תבניות תקיפה" החוזרות על עצמן – כיוון שהמסווג מחפש מונחים בעלי משמעות, המסווג מתבסס על כך שהתקיפה צריכה להכיל תקיפות אלו. תקיפות מתוחכמות היום מבצעות ערפול ל payload כך שחיפוש מחרוזת ספציפית מוכרת לא תעזור, אלא רק <u>הבנה משמעותית של סוג התקיפה, ההשפעה, והיכולת של התוקף לנסות ולהתחמק</u>.
- המסווג לא מתמודד בצורה איכותית עם קידודים – אמנם ההתייחסות לקידודים היא בביצוע הtokenize למידע, אך היום ניתן באמצעות הקידוד "להחביא" את הpayload הזדוני, כך שירוץ בצורה תקינה לחלוטין בשרת הנתקף, אך התבוננות מבוססת טקסט לא תזהה זאת (גישות blacklist)
  - בdataset זה נראה שבקשות המכילות קידוד/קידוד כפול לא קיימות/מעטות מאוד ולכן תוצאות המסווג נשארו גבוהות מאוד, אך בשלב המימוש של הפרויקט, זו נקודת תורפה ששמתי אליה לב

- המסווג לא עמיד בפני תקיפות BlackBox – בהינתן תוקף זדוני שמקבל את מילון ה TF-IDF שהמסווג עובד איתו, הוא יצליח למצוא סוגי תקיפות שיעברו מתחת לראדר שלו ללא שום בעיה.

<u>הצעת מחקר עתידית לטובת שיפור המסווג, הינה התייחסות לנקודות שהעליתי מעלה בתצורות הבאות:</u>

- הוספת Expert Knowledge עבור כל אחת מהתקיפות – הוספת עוד מונחים/patterns החוזרים על עצמם בהקשר תקיפה כלשהי. ניתן לבצע זאת באמצעות הוספת מונחים למילונים, אך צריך לדעת לתת "ציון" איכותי למונחים אלו – לא גבוה מידי כדי לא לגרום ל FP גבוהים, אך לא נמוך מידי שלא ישפיע על סיווג הבקשה
- התייחסות חכמה לקידודים – במקום לבצע tokenize לurl encoded data – ביצוע decode למידע. מהיכרות עם עולם החבאת המידע, ניתן יהיה לזהות מונחים זדוניים ברגע שיפתח להם הקידוד, בניגוד למצב הנוכחי בו פשוט נאבד את המידע.
  - אם נקח לדוגמה את המונח הבעייתי <script> אז במצב בו מחליטים לקדד את כל המילה, אז נקבל באלגוריתם הנוכחי רצף של טוקנים המפרידים בין מונחים ריקים (שכן כל אות היא טוקן אחרי url encode).

    כלומר, איבדנו כל קשר ויכולת להבין את משמעות המידע והמונח הקלאסי לניסיון הזרקת סקריפט.
- יצירת context בין פרמטרים ומונחים – כרגע המסווג מבצע הסתכלות מאוד Stateless עבור כל בקשה – מסתכל מונח-מונח, ומבין האם הוא בעל משמעות. פעמים רבות יש קשר מעניין בין <u>רצף של מונחים</u>, ומיקום המונח (לדוגמה האם הוא מופיע ב url, האם בהאדר ספציפי וכו').
- עיבוי ה dataset הנוכחי באמצעות כתיבת בקשות זדוניות חדשות ומאתגרות יותר, על מנת ללמוד מונחים נוספים קריטיים לכל סוג תקיפה
- שילוב מנגנוני למידה On-The-Fly עבור אפליקציות קיימות, שכן המודל לא עמיד בפני תקיפות BlackBox בו תוקף לומד את המונחים הקריטיים איתם הוא עובד לטובת הסיווג. בשילוב מנגנוני למידה תקינה של האפליקציה עליה היא אחראית להגן, המודל יהיה עמיד יותר בפני תוקף כזה באמצעות חיזוק הפרמטרים המשוייכים לקטגוריית הValid במודל.
- מעבר של Expert Knowledge על מילון הTF-IDF לטיוב המונחים שנלמדו, והורדת מונחים בעייתיים שנלמדו בגלל ה dataset, אך בלי קשר ישיר לסוג התקיפה

9. רשימת מקורות

[1] Gallagher, Brian & Eliassi-Rad, Tina. (2009). Classification of HTTP Attacks: A Study on the ECML/PKDD 2007 Discovery Challenge, Proceeding of ECML/PKDD (2), pages 442–457, 2009

[2] "Analyzing Web Traffic, ECML/PKDD 2007 Discovery Challenge", [Online]. Available: http://www.lirmm.fr/pkdd2007-challenge/

[3] Liu, M., Zhang, B., Chen, W., & Zhang, X. (2019). A survey of exploitation and detection methods of XSS vulnerabilities. Proceeding of IEEE Access, 1–1.

[4] Rodríguez, G. E., Torres, J. G., Flores, P., & Benavides, D. E. (2019). Cross-Site Scripting (XSS) Attacks And Mitigation: A Survey. Proceeding of Computer Networks.

[5] Johari, R., & Sharma, P. (2012). A Survey on Web Application Vulnerabilities (SQLIA, XSS) Exploitation and Security Engine for SQL Injection. Proceeding of International Conference on Communication Systems and Network Technologies.

[6] "OWASP Top Ten", [Online]. Available: https://owasp.org/www-project-top-ten/

[7] "HTTP Headers", [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

[8] "Vector Space Models", [Online]. Available: https://www.sciencedirect.com/topics/computer-science/vector-space-models

[9] "Cross Site Scripting(XSS)", [Online]. Available: https://owasp.org/www-community/attacks/xss/

[10] "Cross-site scripting", [Online]. Available: https://en.wikipedia.org/wiki/Cross-site_scripting

[11] "SQL Injection", [Online]. Available: https://owasp.org/www-community/attacks/SQL_Injection

[12] "LDAP injection", [Online]. Available: https://en.wikipedia.org/wiki/LDAP_injection

[13] "XPATH Injection", [Online]. Available: https://owasp.org/www-community/attacks/XPATH_Injection

[14] "Path Traversal", [Online]. Available: https://owasp.org/www-community/attacks/Path_Traversal

[15] “Command Injection”, [Online]. Available: https://owasp.org/www-community/attacks/Command_Injection

[16] “Server-Side Includes (SSI) Injection”, [Online]. Available: https://owasp.org/www-community/attacks/Server-Side_Includes_(SSI)_Injection

[17] “Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures”, [Online]. Available: https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/

[18] “Security: SSI Injection. What? How? ”, [Online]. Available: https://medium.com/@shatabda/security-ssi-injection-what-how-fbce1dc232b9