# סמינר באבטחת המרחב המקוון – 20927

## הנושא: טכניקות למידת מכונה לזיהוי הזרקות קוד

שם הסטודנט: גיא אבן

מספר הסטודנט: 318911963

שם המנחה: פרופ' לאוניד ברנבוים

תאריך הגשה: 20/01/2025

# תוכן עניינים

# פרק 1 - מבוא להזרקת קוד

הזרקות קוד (Code Injections)[13] הן אחת הפגיעויות החמורות והנפוצות בתחום אבטחת המידע, מהוות איום על שלמות, סודיות וזמינות המידע במערכות. מתקפה זו מתבצעת כשמנוצלת נקודת תורפה בעיבוד נתוני הקלט, המאפשרת הזרקת קוד זדוני שיתבצע כחלק מהמערכת.
המערכת אינה בודקת את הקלט המוזן, מה שמאפשר לתוקף לגשת למידע רגיש, לשנות נתונים או להשתלט על המערכת.

הפופולריות של מתקפות אלו נובעת מהשפעת נתוני הקלט על זרימת הביצוע, המאפשרת שיטות תקיפה מגוונות. אחת השיטות הנפוצות היא הזרקת קוד בפרמטרים המועברים לתוכניות עזר חיצוניות, שמשפיעות על התוכנה המקורית ומאפשרות לתוקף שליטה במערכת. כדי להגן על המערכות, יש לנקוט בטכניקות כמו סינון קפדני של נתוני הקלט, שימוש בטכניקות קידוד נכונות והטמעת כלי אבטחה כמו חומות אש ליישומים (WAF).

בדרך כלל, התקפת הזרקת קוד מתרחשת בשני שלבים:

1. שלב ההזרקה: התוקף מוסיף קוד זדוני כקלט למערכת. קוד זה עשוי להיות חלק מפקודת SQL, סקריפט דפדפן, או קוד מערכת אחרת, והוא משולב בתוך הקוד החוקי של המערכת.
2. שלב הביצוע: המערכת מבצעת את הקוד שהוזרק כאילו היה חלק מהקוד הרגיל שלה. כתוצאה מכך, התוקף יכול לבצע פעולות שונות כמו גישה לנתונים רגישים, ביצוע פקודות על השרת, או השגת שליטה על המערכת.

**סוגי איומים עיקריים מהזרקות קוד**:

- גניבת מידע רגיש: הזרקת קוד, כמו SQL Injection, מאפשרת לתוקף לשלוף נתונים רגישים ממסדי נתונים, כגון פרטי משתמשים וסיסמאות, דבר שמוביל לחשיפת נתונים אישיים ולפגיעות בפרטיות המשתמשים.
- שינוי והשחתת נתונים: תוקפים יכולים לשנות או למחוק מידע קריטי במערכת באמצעות הזרקת קוד, כמו הוספת פקודות זדוניות למסד הנתונים או יצירת משתמשים עם הרשאות גבוהות.
- השתלטות על השרת: ב-Command Injection, התוקף מזריק פקודות למערכת ההפעלה דרך היישום, מה שמאפשר לו להשיג שליטה מלאה על השרת ולהפעיל תוכנות זדוניות.
- התקפות XSS (Cross-Site Scripting): תוקפים יכולים להזריק סקריפטים לדפי אינטרנט, מה שמוביל לגניבת נתוני התחברות או לביצוע פעולות בשם המשתמשים ללא ידיעתם.

**חשיבות הזיהוי של הזרקות קוד**:

זיהוי מוקדם של ניסיונות להזריק קוד זדוני הוא קריטי על מנת למנוע נזקים חמורים למערכת ולמשתמשים. זיהוי איומים אלו בזמן מאפשר למנהלי מערכות ואבטחה לנטרל את ההתקפה לפני שנגרם נזק משמעותי, ואף ללמוד את שיטות הפעולה של התוקפים על מנת לשפר את ההגנות העתידיות.

כדי להבטיח אבטחה מקסימלית, יש להשתמש בטכניקות כגון:

- סינון קלט ואימותו: בדיקת קלטים המוזנים למערכת והסרה של תווים או פקודות מסוכנות.
- שימוש בפרוטוקולים ומסגרות מאובטחות: בחירה בכלים ובטכנולוגיות שיודעים להגן מפני הזרקות קוד, כמו שימוש בהצהרות מוכנות (Prepared Statements) למניעת SQL Injection.

- <u>מעקב והתראה</u> : מערכות ניטור שיכולות לזהות דפוסי פעולה חשודים ולשלוח התראות בזמן אמת על ניסיונות תקיפה.

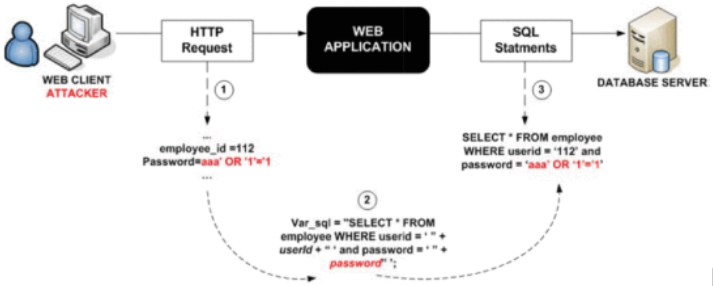
בהתחשב באיומים הפוטנציאליים הגדולים שמקורם בהזרקות קוד, חשיבות הזיהוי המהיר והטיפול היעיל בפגיעויות אלו הופכת להיות קריטית לשמירה על אבטחת מערכות מידע והגנה על נתוני המשתמשים.

## SQL injection (SQLi)

התקפת SQL Injection (SQLi) [13] היא אחת מאיומי האבטחה הנפוצים והמסוכנים ביותר ברשת. היא מנצלת את האופי הדינמי של אתרי אינטרנט המתקשרים עם מסדי נתונים כדי לשלוף או לתפעל מידע. מתקפת SQLi שולחת פקודות SQL זדוניות לשרת מסד הנתונים במטרה לחלץ נתונים רגישים, לשנות או למחוק נתונים, לבצע פקודות מערכת, או להפעיל התקפות DoS.
מתקפה זו מתרחשת כאשר קלט המשתמש מסונן בצורה שגויה, מה שמאפשר לתוקף להזריק פקודות SQL לא בטוחות.

השלבים המעורבים הם כדלקמן :

1. האקר מוצא פגיעות ביישום אינטרנט מותאם אישית ומחדיר פקודת SQL למסד נתונים על ידי שליחת הפקודה לשרת האינטרנט. הפקודה מוזרקת לתעבורה שתתקבל על ידי חומת האש.
2. שרת האינטרנט מקבל את הקוד הזדוני ושולח אותו לשרת יישומי האינטרנט.
3. שרת יישומי האינטרנט מקבל את הקוד הזדוני משרת האינטרנט ושולח אותו לשרת מסד הנתונים.
4. שרת מסד הנתונים מבצע את הקוד הזדוני במסד הנתונים. מסד הנתונים מחזיר למשל נתונים מטבלת כרטיסי אשראי.
5. שרת יישומי האינטרנט יוצר באופן דינמי דף עם נתונים כולל פרטי כרטיס אשראי ממסד הנתונים.
6. שרת האינטרנט שולח את פרטי כרטיס האשראי להאקר.




דוגמה לשלבים של sqli [3]

לדוגמא נניח שקיים לנו סקריפט שבונה שאילתת SQL על ידי שילוב מחרוזות מוגדרות מראש עם טקסט שהוזן על ידי משתמש[13]

var Shipcity;

ShipCity = Request.form (“ShipCity”);

var sql = "select * from OrdersTable where ShipCity = ' " +

ShipCity + " ' ";

הכוונה של מעצב התסריט היא שמשתמש יזין שם של עיר. לדוגמה, כאשר הסקריפט מבוצע, המשתמש מתבקש להזין עיר, ואם המשתמש מזין את Redmond, נוצרת שאילתת ה-SQL הבאה

SELECT * FROM OrdersTable WHERE ShipCity = ‘Redmond’

נניח, עם זאת, המשתמש מזין את הדברים הבאים :

Boston’; DROP table OrdersTable--

זה גורם לשאילתת SQL הבאה:

SELECT * FROM OrdersTable WHERE ShipCity =

‘Redmond’; DROP table OrdersTable--

הנקודה-פסיק הוא אינדיקטור המפריד בין שתי פקודות, והמקף הכפול הוא אינדיקטור לכך שהטקסט הנותר של השורה הנוכחית הוא הערה ולא לביצוע. כאשר שרת SQL מעבד הצהרה זו, הוא יבחר תחילה את כל הרשומות ב-OrdersTable שבהן ShipCity היא Redmond. לאחר מכן, הוא מבצע את בקשת ה-DROP, אשר מוחקת את הטבלה.

אנחנו יכולים לאפיין את התקפות SQLi מבחינת שדרת ההתקפה וסוג ההתקפה. דרכי ההתקפה העיקריות הן כדלקמן [13]:

- קלט משתמש: תוקפים מחדירים פקודות SQL דרך קלט משתמש, בעיקר מהגשת טפסים בבקשות HTTP GET או POST, מה שמאפשר לאפליקציה לגשת אליו כאל משתנה רגיל.
- משתני שרת: משתנים אלו כוללים כותרות HTTP וסביבת רשת. אם הם נרשמים למסד נתונים ללא חיטוי, תוקפים יכולים לזייף את הערכים בכותרות ולבצע מתקפותSQL דרך השאילתות.
- הזרקה מסדר שני: מתרחשת כאשר מנגנוני המניעה לא שלמים. תוקף מנצל נתונים קיימים במערכת כדי להפעיל התקפת SQL, כך שהקלט לא מגיע מהמשתמש אלא מהמערכת עצמה.
- Cookies: תוקף יכול לשנות קבצי Cookie, והשינויים יכולים להשפיע על השאילתות שנבנות על ידי השרת.
- קלט משתמש פיזי: הזרקת SQL אפשרית גם באמצעות קלט כמו ברקודים, תגיRFID או טפסי נייר, שנסרקים ומועברים למערכת.

ניתן לקבץ סוגי התקפה לשלוש קטגוריות עיקריות: inband, inferential ו- out-of-bound. התקפה inband משתמשת באותו ערוץ תקשורת להזרקת קוד SQL ואחזור תוצאות. הנתונים שאוחזרו מוצגים ישירות בדף האינטרנט של האפליקציה. סוגי התקפות inband כוללים את הדברים הבאים:

- טאוטולוגיה: צורה זו של התקפה מחדירה קוד להצהרה מותנית אחת או יותר, כך שהן תמיד מוערכות לאמיתות. לדוגמה, נניח ויש את הסקריפט הזה, שמטרתו לדרוש מהמשתמש להזין שם וסיסמה חוקיים:

$query = “SELECT info FROM user WHERE name =’$_GET[“name”]’ AND pwd = ‘$_GET[“pwd”]’”;

נניח שהתוקף שולח ״– OR 1=1 ׳ ״ עבור הפרמטר. השאילתה שתתקבל תיראה כך:

SELECT info FROM users WHERE name = ‘ ‘ OR 1=1 -- AND pwpd = ‘ ‘

הקוד המוזרק משבית את בדיקת הסיסמה (בגלל מחוון ההערות --) והופך את כל סעיף ה- WHERE לטאוטולוגיה. כתוצאה מכך, בסיס הנתונים מעריך את השאילתה כ- true עבור כל השורות בטבלה ומחזיר את כולן.

- הערת סוף שורה: הזרקת קוד לשדה מסוים מבטלת קוד לגיטימי באמצעות הוספת ״- -״ כך ששאר השאילתה תיחשב כהערה ולא כקוד לביצוע. הדוגמה לטאוטולוגיה היא מקרה מסוג זה.
- שאילתות בגיבוי: התוקף מוסיף שאילתות מעבר לשאילתה הלגיטימית, תוך ניצול תצורות שרת המאפשרות מספר שאילתות במחרוזת אחת. הדוגמה הקודמת היא מקרה כזה.

ב- inferential, אין העברה ממשית של נתונים, אך התוקף מסוגל לשחזר את המידע על ידי שליחת בקשות מסוימות ותצפית על ההתנהגות המתקבלת של האתר/שרת מסד הנתונים. סוגי התקפות inferential כוללים את הדברים הבאים [13]:

- שאילתות לא חוקיות/שגויות מבחינה לוגית: התוקף מנצל הודעות שגיאה תיאוריות מדי לאיסוף מידע על מבנה מסד הנתונים, מה שעוזר בהכנה להתקפות נוספות.
- הזרקת SQL עיוורת: התוקף שואל שאלות אמת/שקר כדי להסיק נתונים, גם כשהמערכת לא מציגה מידע שגוי. התנהגות הדף משתנה בהתאם לתשובה, מה שמספק רמזים על הנתונים.

בהתקפה out-of-bound, נתונים מאוחזרים באמצעות ערוץ אחר (למשל, מופק הודעת דואר אלקטרוני עם תוצאות השאילתה ונשלח אל הבוחן). זה יכול לשמש כאשר יש מגבלות על אחזור מידע, אבל הקישוריות היוצאת משרת מסד הנתונים רופפת.

מכיוון שהתקפות SQLi כל כך נפוצות, מזיקות ומגוונות הן לפי אפיק התקיפה והן לפי סוג, אמצעי נגד יחיד אינו מספיק. במקום זאת יש צורך בסט משולב של טכניקות.

ניתן לסווג את אמצעי הנגד הללו לשלושה סוגים: קידוד הגנתי, זיהוי ומניעת זמן ריצה. התקפות SQLi רבות מצליחות מכיוון שמפתחים השתמשו בשיטות קידוד לא מאובטחות. לפיכך, קידוד הגנתי הוא דרך יעילה להפחית באופן דרמטי את האיום מ- SQLi. דוגמאות לקידוד הגנתי כוללות את הדברים הבאים [13]:

- שיטות קידוד הגנתיות ידניות: בדיקות קלט, כמו אימות סוגי נתונים והתאמת דפוסים, יכולות למנוע התקפות SQLi על ידי זיהוי קלט חשוד.
- הכנסת שאילתה עם פרמטרים: מונע SQLi על ידי הפרדה בין מבנה השאילתה לפרמטרי הערך, כך שקלט המשתמש לא משנה את השאילתה.
- SQL DOM: מספק מחלקות לאימות סוגי נתונים ובריחה, ומשתמש באנקפסולציה (encapsulation) כדי לבנות שאילתות בצורה בטוחה דרך API בדוק, המיישם שיטות קידוד מומלצות.

פותחו מגוון שיטות זיהוי, כולל הבאות:

- מבוסס חתימה: טכניקה זו מנסה להתאים דפוסי התקפה ספציפיים. גישה כזו חייבת להיות מעודכנת כל הזמן וייתכן שלא תפעל נגד התקפות בשינוי עצמי.
אופן פעולת זיהוי מבוסס חתימה[4]

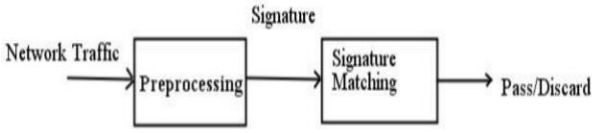


Fig.1 Signature Based Detection

- מבוסס אנומליה: גישה זו מנסה להגדיר התנהגות נורמלית ואז לזהות דפוסי התנהגות מחוץ לטווח הנורמלי. נעשה שימוש במספר גישות. במונחים כלליים, ישנו שלב אימון, בו המערכת לומדת את טווח ההתנהגות הנורמלית, ולאחר מכן שלב הגילוי בפועל.
אופן פעולת זיהוי מבוסס אנומליה[4]

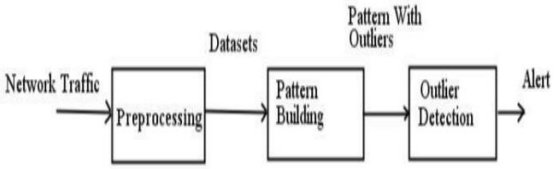


Fig.2 Anomaly Based Detection

- ניתוח קוד: טכניקות ניתוח קוד כוללות שימוש בחבילת בדיקה לאיתור פגיעויות של SQLi. חבילת הבדיקות נועדה ליצור מגוון רחב של התקפות SQLi ולהעריך את תגובת המערכת.

לבסוף, פותחו מספר טכניקות למניעת זמן ריצה כאמצעי נגד של SQLi. טכניקות אלו בודקות שאילתות בזמן ריצה כדי לראות אם הן תואמות מודל של שאילתות צפויות. כלים אוטומטיים שונים זמינים למטרה זו.

## (XSS) Cross site scripting

מחלקת פגיעויות נוספת נוגעת לקלט שסופק על ידי משתמש אחד, שמועבר למשתמש אחר, הידועה כהתקפת Cross Site Scripting (XSS) [13]. פגיעות זו כוללת הכללת קוד סקריפט בתוכן HTML של דף אינטרנט המוצג בדפדפן המשתמש. הקוד יכול להיות ב-JavaScript,ActiveX, VBScript או שפות סקריפט אחרות הנתמכות בדפדפן.

למרות שדפדפנים מגבילים גישה לנתונים בין דפים שונים מאותו אתר, התקפות XSS מנצלות הנחות אלו כדי לעקוף את הבדיקות ולהשיג גישה לנתונים רגישים, כמו תוכן עמוד וקבצי Cookie. התוקפים משתמשים במגוון שיטות להזריק סקריפט זדוני לדפים המוחזרים למשתמשים. הגרסה הנפוצה ביותר היא **reflected XSS**, שבה התוקף מוסיף סקריפט זדוני בנתונים המסופקים לאתר, ואם תוכן זה מוצג למשתמשים אחרים ללא בדיקה, הם יפעילו את הסקריפט בהנחה שהנתונים מהימנים. שימוש נרחב בתוכניות כמו ספרי אורחים ובלוגים מחייב בדיקות תוכן כדי למנוע התקפות כאלה.

לדוגמא

Thanks for this information, its great!

<script>document.location='http://hacker.web.site/cookie.cgi?'+document.cookie</script>

כאשר טקסט זה נשמר על ידי יישום ספר מבקרים, הוא מציג מעט טקסט ולאחר מכן מפעיל קוד JavaScript. קוד זה מחליף את תוכן המסמך במידע מהסקריפט של התוקף, המסופק עם קובץ ה-cookie. התקפה זו מאפשרת לתוקף לגשת לקובץ ה cookie-של המשתמש ולהתחזות לו באתר המקורי, מה שמוביל להחלפת תוכן הדף עם המידע מהסקריפט.

כדי למנוע התקפות כאלה, יש לבדוק את הקלט שסופק על ידי המשתמש ולהסיר או "לברוח" מקוד מסוכן. חשוב לציין שהתוקף עשוי לקודד את התווים הקשורים לקוד הסקריפט ( HTML snippets) כדי להקשות על הזיהוי.

**אמצעים להגנה:**

1. <u>קידוד פלט</u>: יש לוודא שכל המידע שנשלח בחזרה ללקוח מקודד באופן מתאים (HTML, CSS, JavaScript וכו'). לדוגמה, שימוש בפונקציות קידוד כמו htmlspecialchars() ב-PHP או encodeForHTML() בספריות אבטחה של Java .
2. <u>הימנעות משימוש במשתני קלט ישירות בתגובות</u>: יש לטפל במשתנים המגיעים מהקלט (כגון קלט ממשתמש) באופן זהיר, ולוודא שלא ניתן להזריק אותם ישירות למקומות רגישים כמו תגים או סקריפטים.
3. <u>מדיניות תוכן מאובטח (Content Security Policy - CSP)</u>: הטמעת CSP יכולה להגדיר אילו מקורות תקפים להורדת סקריפטים, ובכך למנוע טעינה וביצוע של סקריפטים זדוניים.
4. <u>אימות קלט</u>: בדקו וסננו קלטים מכל המקורות (URL, טפסים וכו') כדי לוודא שהמידע תקין ומתאים לפני שמבצעים עיבוד נוסף.

סוג נוסף של התקפת XSS הוא **XSS stored**, שהיא מסוכנת במיוחד. התקפה זו מתרחשת כאשר אתר מאחסן קלט מהמשתמש מבלי לבדוק אם הוא מכיל קוד זדוני. במקרה זה, הקוד הזדוני נשמר בבסיס הנתונים ומוצג למשתמשים אחרים המבקרים בדף המושפע.

**אמצעים להגנה:**

1. <u>קידוד פלט</u>: בדיוק כמו בהגנה מפני XSS Reflected, יש לקודד כל תוכן שמגיע ממקור חיצוני לפני הצגתו למשתמש, כולל תוכן המאוחסן בשרת.

2. בדיקה וסינון של קלטים: בעת קבלת קלט מהמשתמשים (כמו בהודעות או פוסטים), סננו כל קלט בצורה מדויקת כדי להסיר סקריפטים פוטנציאליים לפני השמירה במסד הנתונים.

3. שימוש בספריות מוכרות לאבטחה: יש להשתמש בספריות ובמסגרות עבודה (frameworks) שמספקות פונקציות מובנות לאבטחת קלטים ופלטים, כמו OWASP ESAPI.

4. בידוד קבצים ותכנים המוזרקים: אם יש שימוש בקבצים שמועלים על ידי משתמשים או תוכן דינמי, בידודו בסביבה נפרדת עשוי למנוע ממנו לגרום לנזק או להזריק קוד לתוכן לגיטימי.

**סקירה על טכניקות זיהוי הזרקות קוד מסורתיות**

הזרקת קוד (Code Injection) היא מתקפה שבה תוקף מחדיר קוד זדוני ליישום במטרה לשנות את התנהגותו או להפיק מידע רגיש. כפי שמציין סטנסילב [1], מתקפות אלו לא תמיד משנה את המבנה התחבירי של הפלט, מה שהופך אותן לקשות לזיהוי בשיטות קונבנציונליות. לעיתים, קוד זדוני עשוי להיראות כמו קוד לגיטימי, דבר המאתגר את השיטות הסטטיות שאינן משתמשות בלמידת מכונה .שיטות זיהוי מסורתיות כמו פילטרים, סינון קלט (input sanitization) וניתוח דינמי התפתחו, אך זיהוי ההתקפות קשה כשאין שינוי במבנה התחבירי של הפלט או כשמתבצעת מתקפה אחרת שמשנה אותו. לכן, יש צורך בפיקוח הדוק על שלבי הקלט וביצוע הקוד.

1. סינון קלטים (Input Validation)
הוא משמש להגנה מפני הזרקות קוד וגם לטכניקה לזיהוי ניסיונות הזרקה. יש בדיקת קלט משתמש לזיהוי תווים מיוחדים ודפוסים חשודים, כמו פקודות SQL או JavaScript, ומונע הזרקות קוד.

2. קידוד קלט (Input Encoding)
המרת תווים רגישים בקלט למופעים מיוחדים כדי למנוע פרשנות כקוד במהלך הריצה. כמו התווים <, >, & וכדומה.

3. דפוסים מבוססי חתימות (Signature-Based Detection)
שימוש בחתימות של התקפות מוכרות, כמו ב-Web Application Firewalls (WAF), כדי לזהות דפוסי קלט חשודים שמבוססים על התקפות עבר.

4. בדיקות היגיון (Logic-Based Detection)
זיהוי התקפות באמצעות ניתוח דפוסי התנהגות תקינה וזיהוי חריגות בלוגיקת הפעולות. לדוגמה, פקודות המועברות לבסיס הנתונים אמורות להיות צפויות ומתאימות לאופני השימוש הרגילים. אם מתקבלת פקודה שאינה תואמת את הלוגיקה הצפויה, היא תסומן כהתקפה אפשרית.

5. שימוש בארגז חול (Sandboxing)
הרצת קוד חשוד בסביבה מבודדת כדי לזהות פעולות לא צפויות, כמו גישה למשאבי מערכת. טכניקה זו משמשת בדרך כלל לזיהוי התקפות סקריפטים כמו XSS או התקפות על שרתי יישומיים.

6. שימוש ב-Canary Values
הכנסה תווים מיוחדים לאזורים רגישים ובודק שינויים בלתי צפויים המעידים על ניסיון התקפה.

7. זיהוי התנהגותי (Behavioral Analysis)
ניתוח דפוסי פעולה של משתמשים ומאתר שינויים חריגים המעידים על פעילות חשודה.

# פרק 2 - מבוא ללמידת מכונה

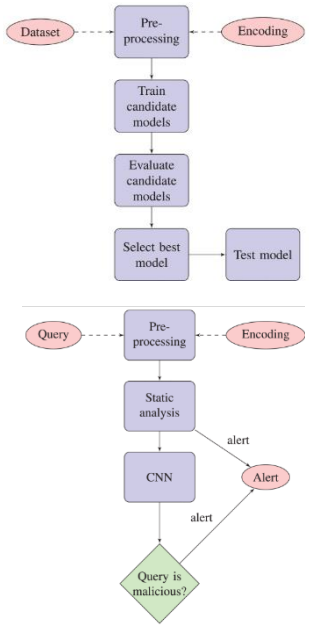
למידת מכונה (Machine Learning) היא תחום מרכזי במדעי המחשב ובאינטליגנציה המלאכותית (AI), שמטרתו לאפשר למחשבים "ללמוד" מתוך הקלט שניתן להם [2] ולהפוך ניסיון לידע ולמומחיות. התהליך מתבצע על ידי שימוש באלגוריתמים המקבלים נתוני אימון – המייצגים ניסיון מצטבר – ומהם מפיקים תובנות שמאפשרות לתוכנה לבצע משימות בצורה יעילה יותר ולהשתפר עם הזמן. בניגוד לאלגוריתמים מסורתיים, הפועלים על בסיס חוקים מוגדרים מראש, אלגוריתמים של למידת מכונה מציעים גישה גמישה ודינמית, המבוססת על למידה מניסיון.

תכונה זו הופכת את למידת המכונה לטכנולוגיה מובילה בתעשייה המודרנית, כולל בתחום אבטחת הסייבר, שבו יש צורך מתמיד להתמודד עם איומים משתנים ומתפתחים. על ידי למידה מנתונים קיימים והסקת מסקנות מדויקות יותר על בסיס מידע חדש, למידת מכונה מאפשרת לזהות דפוסים ולגלות פרצות אבטחה באופן שמסייע להגן על מערכות ממוחשבות בצורה חכמה ומתקדמת.

כלל האצבע בלמידת מכונה קובע שככל שההנחות המוקדמות שמכניס האדם לתהליך חזקות ומוגדרות יותר, כך יהיה קל יותר למכונה ללמוד מדוגמאות נוספות ולהסיק מסקנות ביעילות. עם זאת, יש לכך חסרון – ההסתמכות על הנחות מוקדמות מגבילה את גמישותה של המכונה, וכתוצאה מכך היא עשויה להתקשות להסתגל לנתונים חדשים או חריגים שאינם עולים בקנה אחד עם אותן הנחות [2].

## שלבי הפעולה של למידת מכונה

בדרך כלל, תהליך הלמידה של אלגוריתם למידת מכונה מתחלק לשני שלבים עיקריים:




1. שלב האימון (Training Phase): האלגוריתם לומד ממידע קיים, הידוע כ"סט האימון". המידע כולל דוגמאות רבות, ובמקרים של למידה מפוקחת, הוא כולל גם את התוצאה הרצויה עבור כל דוגמה. במהלך שלב זה, האלגוריתם לומד לזהות את הקשרים בין הקלט לפלט.

   שלב האימון [3] (איור למעלה)

2. שלב הבחינה (Testing Phase): לאחר שהאלגוריתם עבר את שלב האימון ולמד את הדפוסים, הוא נבחן על מידע חדש שלא נחשף אליו בעבר. בשלב זה, המערכת נדרשת לבצע תחזיות או החלטות על בסיס המודל שלמד.

   שלב הבחינה [3] (איור למטה)

תהליך זה מאפשר למערכות מבוססות למידת מכונה להשתפר בביצועיהן לאורך זמן, ככל שהן נחשפות ליותר מידע ונתונים. היכולת הזו ללמוד מניסיון ולהסתגל לתנאים חדשים היא אחת מהתכונות הבולטות שמעניקות יתרון ללמידת מכונה לעומת גישות מסורתיות.

## סוגי למידת מכונה

קיימים מספר סוגי למידת מכונה, כאשר כל סוג מתאים לבעיות שונות:

1. למידה מפוקחת (Supervised Learning): סוג זה של למידה מבוסס על נתונים מתויגים. לכל דוגמה בסט האימון יש קלט וגם פלט ידוע מראש. האלגוריתם לומד לזהות את הקשר בין הקלט לפלט, כך שבהמשך הוא יוכל לחזות את הפלט עבור נתונים חדשים. לדוגמה, במערכת לחיזוי מחירי בתים, האלגוריתם לומד מתוך נתוני קלט של מאפייני הבתים (כגון גודל, מספר חדרים, מיקום וכו') ומהמחירים שבהם הבתים נמכרו, ואז הוא יכול לחזות את המחיר של בית חדש על בסיס מאפייניו.

2. למידה לא מפוקחת (Unsupervised Learning) : כאן האלגוריתם עובד עם נתונים לא מתויגים, כלומר, אין פלט ידוע מראש. מטרת האלגוריתם היא לזהות תבניות בתוך הנתונים, כמו חלוקה לקבוצות דומות או זיהוי חריגות. לדוגמה, בניתוח אשכולות (Clustering), האלגוריתם מחלק את הנתונים לקבוצות על בסיס הדמיון ביניהם, מבלי לדעת מראש מהי הקבוצה ה"נכונה."

3. למידה חצי-מפוקחת (Semi-Supervised Learning) : משלבת נתונים מתויגים ולא מתויגים, תוך שימוש בכמות קטנה של מתויגים לשיפור התחזיות גם עבור הלא מתויגים, מתאימה כשיש מעט דוגמאות מתויגות.

4. למידת חיזוק (Reinforcement Learning) : האלגוריתם לומד מהתנסות עם הסביבה, מקבל תגמולים על פעולות טובות ועונשים על שגיאות, ומשפר את הביצועים לאורך זמן, כמו בתוכנות משחקים.

**תפקיד למידת מכונה בזיהוי הזרקות קוד**

כאמור הזרקות קוד הן מתקפות סייבר נפוצות שבהן תוקף מזריק קוד זדוני ליישום או למסד נתונים כדי להשיג שליטה במערכת או לגשת למידע רגיש. אחת מסוגי ההתקפות הידועות ביותר היא **SQL Injection**, שבו תוקף מזריק פקודות SQL זדוניות דרך שדות קלט של יישום אינטרנטי במטרה לקבל גישה לא מורשית למסד הנתונים.

למידת מכונה מציעה פתרונות מתקדמים לזיהוי ולמניעת מתקפות מסוג זה, תוך הסתמכות על ניתוח דפוסי התנהגות חריגים על בסיס נתוני עבר. באמצעותה ניתן לזהות לא רק התקפות מוכרות, אלא גם כאלה שטרם נצפו. בליבת הגנה זו עומדת יכולתה של למידת מכונה לזהות דפוסים חריגים ולתחקר תעבורת רשת לצורך איתור פעילות זדונית.

מערכות למידת מכונה ממלאות תפקיד מרכזי באבטחת סייבר, במיוחד בזיהוי התקפות כגון SQL Injection. אלגוריתמים מבוססי למידה, כמו רשתות נוירונים קונבולוציוניות (CNN) ורשתות חוזרות (RNN) , מאפשרים זיהוי בזמן אמת של ניסיונות תקיפה מתוחכמים. מודלים אלו מאומנים על נתונים קודמים של בקשות תקינות וזדוניות, ולומדים להבחין בין פעילות רגילה לחריגה, גם אם מדובר בניסיונות תקיפה שלא נתקלו בהם בעבר.

במודלים מונחים כמו SVM או CNN, המערכות לומדות לזהות דפוסים זדוניים מתוך דוגמאות קיימות של תעבורת רשת. בצורה זו, הן מסוגלות לאתר בקשות זדוניות גם במצבים שבהם שיטות התקיפה משתנות. לעומת זאת, למידה לא מונחית מתמקדת בזיהוי אנומליות ללא ידע מוקדם, ומספקת זיהוי של התנהגות חשודה על סמך תבניות חריגות בתעבורת רשת.

השילוב בין שיטות למידה מונחית ולא מונחית משפר את הדיוק בזיהוי התקפות ומגביר את האפקטיביות של מערכות הגנה בעולם הסייבר המודרני. מערכות מבוססות למידת מכונה הופכות לכלי קריטי במלחמה כנגד מתקפות מתקדמות והגנה על מערכות קריטיות מפני איומים מתמשכים.

**יתרונות למידת מכונה בזיהוי הזרקות קוד**

למידת מכונה מאפשרת לזהות ולהתמודד עם מתקפות מתקדמות באופן מהיר ויעיל, תוך ניצול יתרונות משמעותיים :

1. יכולת זיהוי בזמן אמת : האלגוריתם יכול לעקוב אחרי פעילות חשודה במערכת או ברשת, ולהתריע או לנקוט פעולה אוטומטית על סמך המידע שהוא לומד בזמן אמת.

2. גילוי מתקפות חדשות : בעוד שמערכות אבטחה מסורתיות מסתמכות על דפוסים ידועים לזיהוי פגיעויות, אלגוריתמים של למידת מכונה מסוגלים לזהות התקפות חדשות שאינן תואמות לדפוסים קיימים. אלגוריתמים אלה מתבססים על למידה לא מפוקחת שמזהה חריגות בהתנהגות הרשת או היישום.

3. יכולת להתמודד עם כמויות נתונים גדולות: ניתוח מהיר של כמויות נתונים עצומות, מה שמאפשר זיהוי יעיל יותר של פגיעויות.

4. הסתגלות לשינויים: האלגוריתם לומד ומשתפר עם הזמן כדי לזהות התקפות חדשות או שינויים בתבניות קיימות.

**יישומים נפוצים של למידת מכונה בזיהוי פגיעויות**

במחקרים שנעשו על זיהוי הזרקות קוד, נמצא כי למידת מכונה, ובפרט רשתות נוירונים עמוקות (Deep Neural Networks - DNN), משחקות תפקיד משמעותי. רשתות נוירונים עמוקות נבנות כך שהן מסוגלות ללמוד דפוסים מורכבים מתוך הנתונים ומסוגלות לזהות קשרים חבויים.

בנוסף, נעשה שימוש בשיטות אחרות כמו מכונות וקטורים תומכים (SVM) לזיהוי פגיעויות כמו SQL Injection. המודלים מאומנים על נתונים מסומנים, שכוללים דוגמאות של בקשות רשת תקינות ודוגמאות של בקשות זדוניות. לאחר מכן, המודל מסוגל לסווג בקשות חדשות בצורה מדויקת.

**אתגרי היישום**

למרות היתרונות הרבים של למידת מכונה בתחום אבטחת הסייבר, ישנם גם אתגרים רבים:

1. מחסור במאגרי נתונים מתאימים: יש קושי באספת מספיק נתונים מתויגים לאימון יעיל, במיוחד עבור התקפות חדשות או מורכבות.

2. התאמה יתרה (Overfitting): שהמודל מתאים מדי לנתוני האימון, הוא עלול להיכשל בתחזיות על נתונים חדשים. טכניקות כמו Dropout עוזרות להפחית את הבעיה.

3. דרישות מחשוב גבוהות: למידת מכונה, במיוחד רשתות נוירונים עמוקות, דורשת משאבי מחשוב חזקים לאימון על כמות גדולה של נתונים.

**מודל פורמלי של למידת מכונה**

1. קלט הלומד מורכב מהדברים הבאים:
   א. מרחב התחום (X)- כל הערכים האפשריים שהמכונה עשויה לקבל כקלט. לדוגמה, במשימת חיזוי מחירי בתים, הערכים יכולים להיות גודל הבית או מספר חדרים.
   ב. מרחב התוויות (Y)- כל התוויות או הפלטים שהמודל יכול לחזות, כמו מחיר הבית (ערך מספרי) או קטגוריה במקרה של סיווג.
   ג. סט אימון (S)- בוצה של דוגמאות מסומנות, שבה לכל ערך מ-X יש תווית ב-Y, המשמשת לאימון המודל כדי ללמוד את הקשר ביניהם.
2. פלט ניבוי- הלומד צריך להוציא כלל ניבוי באמצעות פונקציה בשם "מנבא"(h) . פונקציה זו, שהיא **פונקציית מטרה** לא ידועה מראש, מגדירה את הקשר בין דוגמות במרחב X לפלטים במרחב Y (h: X → Y). תהליך הלמידה שואף לאמוד את הפונקציה הזו על בסיס נתוני האימון.
3. אלגוריתם הלמידה (A)- תהליך מתמטי המפיק פונקציה משוערת f (היפתזה או מודל) מסט האימון S. האלגוריתם מנסה לאמוד את הפונקציה האמיתית h על סמך הדוגמאות ב- S. בהתחלה, המקרים מ- X מג'ונרטים בהתפלגות הסתברותית Dשאינה מוכרת ללומד.
4. מדדי הצלחה (פונקציית האובדן)- פונקציית האובדן L מודדת את ה"טעות" בין הפלט המשוער f על ערך ב-x לבין הפלט האמיתי Y. מטרת האלגוריתם היא למזער את פונקציית האובדן על פני סט האימון, כלומר להפחית את הפער בין התחזיות לערכים האמיתיים. השגיאה של h היא ההסתברות לכך ש- f(x)≠h(x) סבור מופע אקראי x בהתפלגות D.

$$L_{\mathcal{D},f}(h) \stackrel{\text{def}}{=} \mathbb{P}_{x\sim\mathcal{D}}[h(x) \neq f(x)] \stackrel{\text{def}}{=} \mathcal{D}(\{x : h(x) \neq f(x)\}).$$

5. <u>סיכון אמפירי/ טעות אמפירית</u>- הסיכון מייצג את הציפייה של פונקציית האובדן על פני כל הדוגמאות במרחב $X$. כיוון שלא ניתן לחשב את הסיכון עבור כל הדוגמאות, האלגוריתם מעריך אותו באמצעות הדוגמאות בסט האימון.

$$L_S(h) \stackrel{\text{def}}{=} \frac{|\{i \in [m] : h(x_i) \neq y_i\}|}{m},$$

where $[m] = \{1, \ldots, m\}$.

6. <u>רגולריזציה</u> - שיטה להוספת מגבלות על המודל כדי למנוע התאמת יתר(Overfitting), שבו המודל מתאים יתר על המידה לנתוני האימון ואינו מצליח לחזות טוב בנתונים חדשים. היא כוללת הוספת עונש למודל על מורכבות יתר.
7. <u>הכללה</u>- מתארת את היכולת של המודל לבצע תחזיות נכונות על נתונים חדשים שאינם נכללו בסט האימון. מטרה מרכזית של למידת מכונה היא לבנות מודל עם יכולת הכללה טובה, כך שיוכל להתמודד בהצלחה עם נתונים שלא נראו קודם.

# פרק 3 - טכניקות למידת מכונה נפוצות לזיהוי הזרקות קוד

## למידת מכונה מבוססת עצי החלטה (Decision Trees)

### מבוא לעץ החלטה

עץ החלטה הוא שיטה פופולרית בלמידת מכונה לסיווג ורגרסיה. הוא בנוי כמו עץ, שבו כל צומת מייצג תכונה וכל ענף הוא החלטה על ערך מסוים של משתנה. העץ מתפצל עד שהדאטה מחולק לתתי-קבוצות הומוגניות, והעלים מייצגים את תוצאות הסיווג. המודל מאפשר קבלת החלטות "מבוססות חוקים" והופך אותו להבנה גם עבור לא מומחים. מטרתו היא ליצור מבנה המפריד קבוצות נתונים בדיוק תוך מינימום החלטות, כמו סיווג פירות לפי צבע וגודל.

### הגדרה פורמלית

עץ החלטה הוא "מנבא" עם פונקציה $h: X \to Y$ , המנבאת את התווית המשויכת Y למופע X על-ידי מעבר משורש העץ לעלה הנמצא בעץ. בכל צומת במסלול של בין שורש לעלה, הצאצא היורש נבחר על ידי בסיס פיצול מרחב הקלט.

בדרך כלל, כלל הפיצול מבוסס על ידי אחת התכונות הנמצאות של X או על ידי קבוצה מוגדרת מראש של כללי פיצול, והעלה מכיל תווית ספציפית של Y. כלומר נעבור לילד הימני או השמאלי של הצומת על בסיס $1_{[X_i < \theta]}$ כאשר $i \in [d]$ הוא האינדקס של התכונה הרלוונטית ו- $\theta \in R$ הוא הסף. במקרים כאלה, אנו יכולים לחשוב על עץ החלטה כחלוקה של מרחב המקרים, $X = R^d$, לתאים ,כאשר כל עלה של העץ מתאים לתא אחד. מכאן נובע שעץ עם k עלים יכול לחלק קבוצה של k מקרים. לפיכך, אם נאפשר עצי החלטה בגודל שרירותי, נקבל מחלקת השערה של ממד VC אינסופי (הוא הגודל המקסימלי של קבוצה $C \subset X$ שניתן לקבל על ידי חילוק C לשתי תת קבוצות על ידי H. אם H יכול לחלק קבוצות בגודל שרירותי אנו אומרים של-H יש ממד VC אינסופי).

גישה כזו יכולה בקלות להוביל overfitting. כדי להימנע מ- overfitting, אנו יכולים להסתמך על עקרון אורך התיאור המינימלי (MDL) ולכוון ללמוד עץ החלטות שמצד אחד מתאים היטב לנתונים ומצד שני אינו גדול מדי. במימד VC לצורך הפשטות, נניח שלכל מופע יש וקטור עם d ביטים מכאן ש$i=[d]$ ולכן לכל איבר בX מצויין ביט יחיד. לפיכך, עץ החלטה עם $2^d$ יהיה בעומק של d+1 ואז מספר הדוגמאות שצריך ללמוד בגודל $2^d$. פה נכנס הMDL , ראשית עלינו להגדיר שפת תיאור לעצי החלטה, שהיא נטולת קידומת ודורשת פחות סיביות לעצי החלטה קטנים יותר. עץ עם n צמתים יתואר ב- n+1 בלוקים, כל אחד בגודל $\log_2(d+3)$ סיביות. n הבלוקים הראשונים מקודדים את צמתי העץ, בDFS (סדר תחילי), והבלוק האחרון מסמן את סוף הקוד.

כלומר כל בלוק מציין אם הבלוק הנוכחי הוא או צומת פנימי מסוג $1_{[X_i=1]}$ עבור $i \in [d]$ או עלה עם ערך 1 או עלה עם ערך 0 או סוף הקוד. לפיכך נקבל את החסם הבא המתאר שבסיכוי של לפחות $1-\delta$, השגיאה האמיתית של המודל ($L_d(h)$) תהיה קטנה או שווה לשגיאה האמפירית ($L_s(h)$) בתוספת גורם נוסף. הגורם הנוסף תלוי בממד VC של המודל ,בגודל קבוצת האימון וברמת הביטחון.

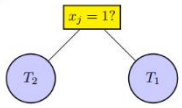
$$L_{\mathcal{D}}(h) \le L_S(h) + \sqrt{\frac{(n+1)\log_2(d+3) + \log(2/\delta)}{2m}}.$$

<u>רעיון אלגוריתם לגידול העץ</u>: הליך גידול עץ החלטה מתחיל בעץ עם עלה בודד (השורש) שמקבל תווית לפי הצבעת רוב על סט האימונים. לאחר מכן, בכל איטרציה, אנו בוחנים פיצול של עלה אחד ומחשבים מדד "רווח" לשיפור. בין כל הפיצולים האפשריים, אנו בוחרים את זה שממקסם את הרווח, או מחליטים לא לפצל את העלה.

הפסאודו קוד הבא משתמש בפעולות רקורסיביות כאשר נקרא תחילה עם ID3(S,[d]) ומחזיר עץ החלטה. פעולה Gain(S,i) מקבלת סט אימון S ואינדקס i ומעריכה את ה"רווח" של פיצול העץ על פי התכונה ה-i-ית.

$\texttt{ID3}(S, A)$

INPUT: training set $S$, feature subset $A \subseteq [d]$
**if** all examples in $S$ are labeled by 1, return a leaf 1
**if** all examples in $S$ are labeled by 0, return a leaf 0
**if** $A = \emptyset$, return a leaf whose value = majority of labels in $S$
**else** :
  Let $j = \text{argmax}_{i \in A}\, \texttt{Gain}(S, i)$
  **if** all examples in $S$ have the same label
    Return a leaf whose value = majority of labels in $S$
  **else**
    Let $T_1$ be the tree returned by $\texttt{ID3}(\{(\mathbf{x}, y) \in S : x_j = 1\}, A \setminus \{j\})$.
    Let $T_2$ be the tree returned by $\texttt{ID3}(\{(\mathbf{x}, y) \in S : x_j = 0\}, A \setminus \{j\})$.
    Return the tree:

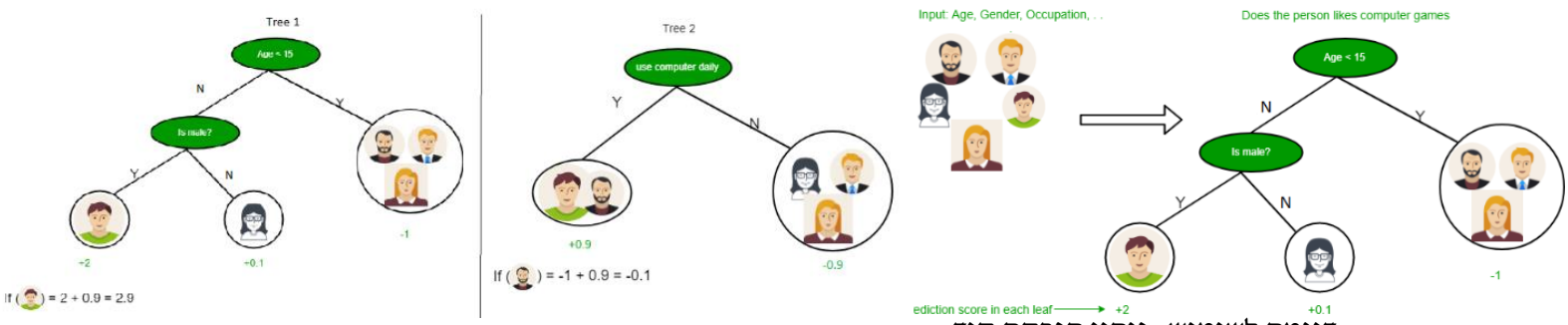



לאחר שנקבל את העץ על ידי ID3 נקבל שהעץ עדיין יחסית גדול ולכן נבצע פעולת "גיזום" על העץ המתקבל. בדרך כלל, הגיזום מתבצע על ידי מעבר מלמטה למעלה על העץ. כל צומת עשוי להיות מוחלף באחד מתתי-העצים שלו או בעלה, בהתבסס על חסימה או הערכה כלשהי של $L_D(h)$, פסאודו קוד אפשרי לפעולת הגיזום :

Generic Tree Pruning Procedure

**input:**
  function $f(T, m)$ (bound/estimate for the generalization error
    of a decision tree $T$, based on a sample of size $m$),
  tree $T$.
**foreach** node $j$ in a bottom-up walk on $T$ (from leaves to root):
  find $T'$ which minimizes $f(T', m)$, where $T'$ is any of the following:
    the current tree after replacing node $j$ with a leaf 1.
    the current tree after replacing node $j$ with a leaf 0.
    the current tree after replacing node $j$ with its left subtree.
    the current tree after replacing node $j$ with its right subtree.
    the current tree.
  let $T := T'$.

לדוגמא עץ החלטה קטן, נשים לב כי הדוגמא מדגישה את אחד היתרונות בעצי החלטה והוא שהמסווג (האלגוריתם) פשוט מאוד להבנה ולפירוש(דוגמאות לשאלות מסווגות) :




<u>דוגמה לשימוש : זיהוי הזרקת קוד</u>

ניתן להשתמש בעצי החלטה כדי לזהות קלטים חשודים המבוססים על מאפיינים כמו אורך הקלט, נוכחות של תווים מיוחדים (למשל < ,> ," ,' ) והימצאות מילות מפתח שקשורות לשאילתות SQL (כמו SELECT, INSERT, DROP).

**בניית עץ החלטה לזיהוי הזרקת קוד**

נניח שאנו רוצים לאמן עץ החלטה שמסווג קלטים לפי "תקין" או "מזיק". הנתונים מורכבים מתכונות כמו :

1. <u>אורך הקלט</u> : קלטים ארוכים במיוחד עלולים להכיל קוד זדוני.
2. <u>נוכחות תווים מיוחדים</u> : מתקפות SQL עשויות לכלול תווים כמו ',"' או ; .
3. <u>מילות מפתח חשודות</u> : קלטים המכילים מילות מפתח כמו SELECT,DROP או INSERT.

עץ ההחלטה עשוי להתחיל בשאלה כמו: "האם אורך הקלט גדול מ-100 תווים?". אם כן, העץ עשוי לשאול האם הקלט מכיל תווים מיוחדים. אם כן, העץ יסווג את הקלט כחשוד. אחרת, העץ ימשיך לשאול שאלות נוספות על נוכחות תווים או מילות מפתח, עד שהקלט יסווג.

**יתרונות של עצי החלטה**

1. פשטות ופרשנות: עץ החלטה הוא מודל שקל להבין ולפרש. בזכות המבנה ההיררכי שלו, ניתן לעקוב אחר כל שלב בתהליך קבלת ההחלטות. תכונה זו מאפשרת לאנשי אבטחת מידע להבין במהירות מדוע מערכת סיווגה קלט מסוים כמסוכן או כתקין.
2. התמודדות עם נתונים קטגוריאליים ומספריים: עצים החלטתיים יכולים לעבוד עם משתנים קטגוריאליים (כגון צבע או שם) וגם עם משתנים מספריים (כגון אורך או משקל). כך אפשר ליישם את המודל במגוון רחב של בעיות סיווג ורגרסיה.
3. אין הנחת ליניאריות: בניגוד למודלים סטטיסטיים אחרים כמו רגרסיה ליניארית, עצי החלטה אינם מניחים שהקשר בין התכונות לתוצאה הוא ליניארי. הם מסוגלים לזהות קשרים מורכבים בין התכונות השונות ולהתמודד היטב עם נתונים בעלי יחסים לא ליניאריים.
4. יכולת לטפל בערכים חסרים: עץ החלטה יכול להתמודד עם נתונים חסרים בצורה יעילה על ידי התעלמות מהתכונות החסרות או בניית חוקים חלופיים המבוססים על מידע קיים.
5. יכולת הכללה טובה במודלים מורכבים: עץ החלטה הוא בסיס לשיטות מתקדמות יותר כמו **יער אקראי (Random Forest)** ו-**Gradient Boosting**, שבהן נבנים מספר עצים שמשתלבים יחד כדי לשפר את דיוק התחזיות.

**חסרונות של עצי החלטה**

1. התאמת יתר (Overfitting): עץ החלטה יכול להיות מורכב מדי אם לא מתבצע גיזום(pruning), מה שגורם לו להתאים באופן מדויק לנתוני האימון, ובכך לפגוע ביכולתו להכליל על נתונים חדשים. לדוגמה, עץ המנתח פרטים קטנים עלול להניב התראות שווא(false positives) על מתקפות.
2. רגישות לשינויים בנתונים: שינויים קטנים בנתונים עשויים לשנות את מבנה העץ באופן משמעותי. הבחירה הראשונה לפיצול יכולה להשתנות בעקבות הבדל קטן, מה שעלול להוביל לעץ שונה לחלוטין ולחוסר יציבות.
3. ביצועים נמוכים במקרים מורכבים: עצי החלטה בודדים עשויים לא להצליח במקרים שבהם הקשרים בין התכונות לתוצאה מורכבים. הם מציגים ביצועים פחות טובים לעומת שיטות מתקדמות כמו יערות אקראיים או Gradient Boosting .
4. חוסר יעילות בעיבוד נתונים מסובכים: עצי החלטה עלולים להפוך לגדולים ומורכבים במערכי נתונים עם הרבה תכונות, מה שיכול לגרום לקושי בהפקת תוצאות טובות ללא גיזום קפדני או שימוש בשיטות כמו MDL (Minimum Description Length), המגבילות את מספר הצמתים.

## למידת מכונה מבוססת SVM (supported vector machines)

**מבוא ל-SVM**

מכונת וקטורים תומכים (SVM, Support Vector Machine) היא אחד מהמודלים המתקדמים ביותר בתחום למידת המכונה, המשמשת בעיקר לפתרון בעיות סיווג ורגרסיה. המודל מבוסס על הרעיון של מציאת "מישור ההפרדה המיטבי" (Optimal Separating Hyperplane) המפריד בין דוגמאות משני מחלקות שונות בצורה מדויקת ככל האפשר. מישור ההפרדה הוא קו או משטח ריבועי במרחב רב-ממדי שמפריד בין שתי קבוצות של נתונים, כך שהמרחק בין הנתונים הקרובים ביותר מכל מחלקה לקו ההפרדה (מה שנקרא **מרווח**) יהיה הגדול ביותר.

## עקרונות בסיסיים של SVM

מטרתו של SVM היא למצוא את המישור ההפרדה שממקסם את המרווח בין המחלקות השונות. בשיטה זו, אנו מתמקדים בעיקר בדוגמאות שנמצאות על קווי הגבול שבין המחלקות, המכונות **וקטורים תומכים** (Support Vectors). הווקטורים התומכים הם הדוגמאות החשובות ביותר מכיוון שהן מגדירות את מיקום המישור המפריד. שאר הדוגמאות אינן משפיעות על המודל במידה משמעותית.

כאשר מדובר במערכי נתונים מורכבים שאינם ניתנים להפרדה באמצעות קו ישר במרחב הדו-ממדי או התלת-ממדי, SVM מאפשר הרחבה של המרחב בעזרת טרנספורמציה לוגית המכונה **גרעין (Kernel)** . הגרעין מאפשר לנו "לשדרג" את המידע למרחב ממדי גבוה יותר, שבו הדאטה הופך לניתן להפרדה בצורה ליניארית.

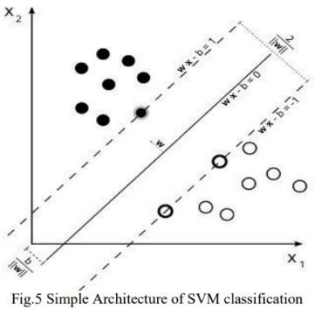
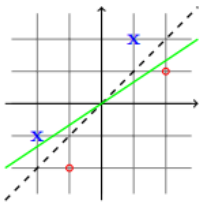


Fig.5 Simple Architecture of SVM classification

ארכיטקטורה של SVM[4] (איור 5)

### הגדרה פורמלית

נניח ויש לנו סט אימון S כאשר $x_i$ שייך ל $R^d$ ו- $y_i$ שייך לפלוס מינוס 1. נוכל לומר כי סט האימון הוא ניתן להפרדה לינארית אם קיים רווח (halfspace) כך ש $y_i=sign(<w,x_i>+b)$ לכל i . באופן שקול נוכל לרשום את התנאי כך : $\forall i \in [m],\quad y_i(\langle \mathbf{w}, \mathbf{x}_i\rangle + b) > 0.$ כל הרווחים (halfspace) שהם (w,b) המקיימים את התנאי נקראים השערת ERM.



באינטואיציה נרצה לבחור את ההיפר מישור (hyperplane) העובר כאשר המרחק ממנו לנקודות הוא מינימלי, לאינטואיציה הזו נשתמש בקונספט הנקרא שוליים (margin) שהם השוליים של היפר-מישור ביחס לסט אימון מוגדרים כמרחק המינימלי בין נקודה במערך האימון לבין המישור. אם להיפר-מישור יש מרווח גדול, אז הוא עדיין יפריד בין מערך האימונים גם אם נפריע מעט בכל מופע. נוכל להבין זו ויזואלית על ידי האיור (ליד) :

המרחק בין נקודה x להיפר מישור מוגדרת כך : $(\mathbf{w},\, b)\ \ where\ \|\mathbf{w}\| = 1\ \ is\ |\langle \mathbf{w}, \mathbf{x}\rangle + b|.$

Hard-SVM הוא כלל למידה שבו אנו מחזירים היפר-מישור ERM המפריד בין מערך האימונים עם השוליים הגדולים ביותר האפשריים והוא מוגדר כך[2] :

$$\underset{(\mathbf{w},b):\|\mathbf{w}\|=1}{\operatorname{argmax}} \ \min_{i\in[m]} |\langle \mathbf{w}, \mathbf{x}_i\rangle + b| \quad \text{s.t.} \quad \forall i,\ y_i(\langle \mathbf{w}, \mathbf{x}_i\rangle + b) > 0.$$

Hard-SVM

**input:** $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
**solve:**

$$(\mathbf{w}_0, b_0) = \underset{(\mathbf{w},b)}{\operatorname{argmin}} \|\mathbf{w}\|^2 \text{ s.t. } \forall i,\ y_i(\langle \mathbf{w}, \mathbf{x}_i\rangle + b) \geq 1 \qquad (15.2)$$

**output:** $\hat{\mathbf{w}} = \frac{\mathbf{w}_0}{\|\mathbf{w}_0\|}, \quad \hat{b} = \frac{b_0}{\|\mathbf{w}_0\|}$

ניסוח מקביל נוסף של כלל Hard-SVM כבעיית אופטימיזציה ריבועית מוגדר כך(איור למטה) :

באופן אינטואיטיבי, Hard-SVM מחפש w של נורמה מינימלית בין כל הווקטורים המפרידים בין הנתונים ועבורם $|<w,x_i>+b|>=1$ לכל i. במילים אחרות, אנחנו אוכפים את השוליים להיות 1, אבל עכשיו היחידות שבהן אנחנו מודדים את הסקאלה של השוליים עם הנורמה של w. לכן, מציאת השוליים של ההיפר מישור הגדולות ביותר מסתכמת במציאת w שהנורמה שלו מינימלית.

**מורכבות הדגימה של hard SVM -** נזכיר כי ממד VC של חצאי מרחבים ב- $R^d$ הוא d + 1. מכאן שמורכבות המדגם של למידת חצאי מרחבים גדלה עם ממדיות הבעיה. יתר על כן, משפט היסוד של הלמידה אומר לנו שאם מספר הדוגמאות קטן משמעותית מ-d חלקי אפסילון אז אף אלגוריתם לא יכול ללמוד חצי מרחב מדויק. זה בעייתי כאשר d הוא גדול מאוד. כדי להתגבר על בעיה זו, נניח הנחה נוספת על התפלגות הנתונים הבסיסית. בפרט, נגדיר הנחה של "הפרדה עם γ שוליים" ונראה שאם הנתונים ניתנים להפרדה עם γ שוליים אז סיבוכיות המדגם תחומה מלמעלה על ידי פונקציה של $1/\gamma^2$ . מכאן שגם אם הממד גדול מאוד (או אפילו אינסופי), כל עוד הנתונים דבקים ביכולת ההפרדה עם הנחת השוליים עדיין יכולה להיות לנו סיבוכיות מדגם

קטנה. אין סתירה לגבול התחתון שניתן במשפט היסוד של הלמידה מכיוון שאנו מניחים כעת הנחה נוספת על התפלגות הנתונים הבסיסית.

**תנאים אופטימליים וקטורי תמיכה-** השם "Support Vector Machine" נובע מהעובדה שהפתרון של Hard-SVM, $w_0$ , נתמך על ידי (בטווח ליניארי של) הדוגמאות שנמצאות בדיוק במרחק של $1/\|w_0\|$ מההיפר-מישור המפריד. וקטורים אלה נקראים אפוא וקטורי תמיכה. כדי לראות זאת, אנו מסתמכים על תנאי האופטימליות של פריץ ג'ון. הלמה של פריץ ג'ון:

LEMMA 15.9 (Fritz John) *Suppose that*

$$\mathbf{w}^\star \in \operatorname*{argmin}_{\mathbf{w}} f(\mathbf{w}) \quad s.t. \ \ \forall i \in [m],\ g_i(\mathbf{w}) \le 0,$$

*where* $f, g_1, \ldots, g_m$ *are differentiable. Then, there exists* $\boldsymbol{\alpha} \in \mathbb{R}^m$ *such that* $\nabla f(\mathbf{w}^\star) + \sum_{i \in I} \alpha_i \nabla g_i(\mathbf{w}^\star) = \mathbf{0}$, *where* $I = \{i : g_i(\mathbf{w}^\star) = 0\}$.

**דוגמה לשימוש: זיהוי הזרקת קוד**

כדי להשתמש ב- SVM לזיהוי מתקפות של הזרקת קוד, נבנה מודל המסווג קלטים כ"חשודים" או "תקינים" בהתבסס על מאפיינים שונים של הקלט. לדוגמה, עבור מתקפות SQL Injection ,נגדיר את שלבי היישום.

**שלבי יישום SVM לזיהוי הזרקת קוד**

1. <u>איסוף והכנת נתונים</u>: בשלב זה יש לאסוף דוגמאות קלט שנשלחו למערכת, כולל דוגמאות של מתקפות מוצלחות ונתונים תקינים. כל דוגמה תתויג כ"מתקפה" או "קלט תקין", ויש לבנות מערך תכונות שייצג את הדוגמאות הללו. לדוגמה נוכל להגדיר את התכונות הבאות:

   א. <u>אורך הקלט</u>: מתקפות עלולות לכלול קלטים ארוכים ולא תקניים.

   ב. <u>תווים מיוחדים</u>: קלט המכיל תווים כמו ',"' או ; עשוי להיות סימן למתקפה.

   ג. <u>מילות מפתח של SQL</u>: מילות מפתח כמו SELECT, DROP, INSERT, או DELETE עשויות להצביע על קלט שנוצר בניסיון לבצע מתקפה.

   ד. <u>תבניות לא רגילות</u>: תבניות של קלטים לא סטנדרטיים או נוכחות של משתנים לא תואמים בין כמות הפלט והכניסה.

   ה. <u>שיעור התו/המילה הנחשד לקלט</u>: ניתן לחשב תדירות הופעת תווים חשודים או מילות מפתח חשודות מתוך כלל הקלט.

2. <u>בניית מודל SVM</u>: מודל SVM משתמש בתכונות שנאספו כדי לבנות מישור הפרדה בין קלטים חשודים לנתונים תקינים. כל קלט מיוצג כנקודה במרחב רב-ממדי, עם ממדים המייצגים תכונות כמו אורך הקלט ומספר תווים חשודים. SVM שואף למצוא את מישור ההפרדה האופטימלי, ובשיטות גרעין(Kernel), כגוון RBF - Radial Basis Function ניתן לשפר את ההפרדה גם כאשר קלטים תקינים וזדוניים דומים.

3. <u>אימון המודל</u>: המודל מתאמן על נתונים מתויגים (קלט תקין או מתקפה) ומסתמך על וקטורים תומכים – דוגמאות קריטיות בגבול ההפרדה בין הקלטים. לאחר האימון, המודל יכול לסווג קלטים חדשים לפי התכונות שנבנו.

4. <u>בדיקה והערכה</u>: לאחר האימון, יש לבדוק את המודל על קבוצת נתונים נפרדת (קבוצת בדיקה) להערכת דיוק הזיהוי שלו במניעת זיהוי שגוי של קלטים תקינים.

5. <u>פריסה והפעלה</u>: לאחר האימון והבדיקה, המודל נפרס כחלק ממערכת האבטחה. קלטים חשודים מסווגים על ידי ה-SVM, ואם זוהו כחשודים, המערכת יכולה לחסום או להעביר אותם לבחינה נוספת.

**כיצד SVM פועל בזיהוי קוד זדוני**

בהנחה שקיים מערך נתונים המכיל קלטים שנאספו ממערכות תוכנה, ניתן לבנות סיווגים עבור הקלטים הללו בהתבסס על התכונות שהוזכרו. לדוגמה, נוכל להזין את הקלט למודל SVM שעבר אימון על תבניות שונות של קלטים בטוחים ומסוכנים. אם SVM יזהה דפוס שאינו עומד בקנה אחד עם קלטים תקינים, הוא יסווג את הקלט כחשוד וימליץ על סינון או חסימה.

**יתרונות של SVM**

1. <u>דייקנות במקרים של מרווח ברור בין המחלקות</u>: SVM מספק פתרון מדויק ויעיל כאשר יש מרווח ברור בין המחלקות השונות, והוא מבטיח שההחלטה תתקבל בהתבסס על הדוגמאות הקריטיות ביותר (הווקטורים התומכים).
2. <u>יעילות במקרים מורכבים</u>: בעזרת גרעינים, SVM יכול לפתור בעיות שבהן הנתונים אינם ניתנים להפרדה בצורה ליניארית. לדוגמה, מתקפות זדוניות עשויות להופיע בצורות מגוונות וקשות להבחנה, והיכולת של SVM להשתמש בגרעינים מאפשרת להרחיב את המודל גם למקרים מורכבים יותר.
3. <u>מניעת התאמת יתר(Overfitting)</u>: SVM מתמקד בדוגמאות הקריטיות (וקטורים תומכים) ולא בפרטים הקטנים, מה שמונע התאמת יתר לנתונים רעשים, בניגוד למודלים כמו עצי החלטה.
4. <u>מודל גמיש</u>: יכולתו של SVM לשלב גרעינים שונים מאפשרת לו להתמודד עם בעיות מורכבות שונות במגוון תחומים, כולל אבטחת מידע, זיהוי תמונות, וטקסט.

**חסרונות של SVM**

1. <u>קושי באימון על מערכי נתונים גדולים</u>: אימון SVM על דאטה גדול הוא מאתגר מבחינה חישובית, במיוחד עם גרעינים למקרים לא ליניאריים, ודורש משאבים רבים.
2. <u>רגישות לבחירת פרמטרים</u>: הצלחת המודל תלויה בבחירת פרמטרים נכונה כמו סוג הגרעין, ודורשת ניסיון ושימוש בטכניקות כמו Cross-Validation.
3. <u>פרשנות מורכבת</u>: המורכבות של SVM מקשה על הבנת ההחלטות, בניגוד למודלים כמו עץ החלטה, מה שמסבך הסבר של תוצאות. דבר זה עלול להיות בעייתי במקרים שבהם חשוב להבין את ההיגיון מאחורי הסיווג (כמו באבטחת מידע).
4. <u>קושי בזיהוי רב-מחלקתי</u>: SVM מתאים בעיקר לסיווג בינארי, כך שבמקרים רב-מחלקתיים (Multiclass Classification) נדרש שילוב של מודלים, מה שמסבך את התהליך.

**<u>למידת מכונה מבוססת רשתות נוירונים מלאכותיות (רשתות עמוקות)</u>**

רשת עצבית מלאכותית היא מודל חישובי המהווה השראה מהמבנה של רשתות עצביות במוח האנושי, שבו נוירונים מחוברים ביניהם ומעבירים מידע דרך חיבורים מורכבים. באופן דומה, רשתות עצביות מלאכותיות מבוססות על יחידות עיבוד שנקראות נוירונים מלאכותיים, אשר מקבלות כקלט סכום משוקלל של פלטים מהנוירונים המחוברים אליהן. רשת עצבית מתוארת כגרף מכוון שבו הצמתים מייצגים את הנוירונים, והקשרים בין הצמתים מייצגים את המשקלים הסינפטיים – המשפיעים על האופן שבו המידע מועבר ברשת.

במוח, היכולת לבצע פעולות מורכבות כמו זיהוי תבניות ושליפה מזיכרון נובעת מהארגון המורכב של הנוירונים ומהיכולת לשנות את החיבורים ביניהם. בדומה לכך, רשתות עצביות מלאכותיות לומדות מהניסיון באמצעות תהליך שבו משקלות הקשרים ביניהם מעודכנים. אלגוריתם הלמידה משנה את המשקלים הסינפטיים בצורה מבוקרת והיררכית, במטרה לשפר את ביצועי הרשת.

הגמישות של המוח מתבטאת ביכולתו להתאים את עצמו לשינויים, וזהו עיקרון שמיושם גם ברשתות עצביות מלאכותיות – במיוחד בתחומים שבהם המידע משתנה כל הזמן, כמו זיהוי

איומים ואבטחת מידע. לדוגמה, בזיהוי הזרקות קוד, הרשת יכולה לשפר את יכולותיה ולהתמודד עם סוגי התקפות חדשים על ידי עדכון משקלי הקשרים בהתאם לנתונים שנאספו.

רשתות עצביות מלאכותיות מאחסנות מידע בחיבורים שבין הנוירונים, הידועים גם כמשקלים סינפטיים, ורוכשות ידע חדש באמצעות תהליך הלמידה. יכולת זו מאפשרת להן להתמודד עם בעיות חישוב מורכבות ולהשיג את מטרותיהן, כמו זיהוי דפוסים, הבנת נתונים והתרעה על איומים, תוך התאמה דינמית של המשקלים על פי האתגרים הניצבים לפניהן.

**יתרונות של רשתות נוירונים**

כוח החישוב של רשתות נוירונים נובע, בראש ובראשונה, מהמבנה המבוזר והמקבילי שלהן. מעבר לכך, לרשתות נוירונים יש את היכולות ללמוד ולכן, להכליל. הכללה ברשתות נוירונים פירושה הפקת פלט סביר עבור קלט שלא הופיע בשלב האימון (למידה). שתי יכולות עיבוד המידע שהוזכרו מאפשרות לרשתות נוירונים לפתור בעיות מורכבות וקשות לפתירה. בפועל, רשתות נוירונים אינן יכולות לספק פתרון כאשר המערכת כולה מורכבת רק מהרשת עצמה. גישת עיצוב המערכת צריכה לכלול רשתות נוירונים כחלק אינטגרלי עוד בשלב בעיצוב. בהינתן בעיה מורכבת, יש לפרקה לבעיות יותר קטנות ופשוטות ולהקצות רשת נוירונים לכל אחת ואחת מהן, כאשר כל רשת נוירונים בנויה לטפל בבעיה הספציפית שהוקצתה לה.

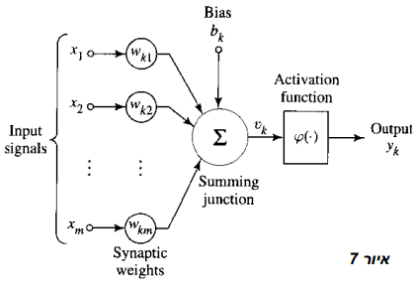
רשימת יתרונות ותכונות רצויות של רשתות נוירונים[7]:

- אי-ליניאריות – אי-ליניאריות ברשת נוירונים מתבטאת בכך שהקשרים בין הנוירונים מבוססים על פונקציות עיבוד אי-ליניאריות, מה שהופך את הרשת כולה לאי-ליניארית. תכונה זו חשובה במיוחד מכיוון שאותות קלט רבים, כמו קול, הם אי-ליניאריים מטבעם, והרשת מסוגלת להתמודד עם מורכבות כזו בצורה מבוזרת לאורך כל הנוירונים.
- מיפוי קלט-פלט – מיפוי קלט-פלט בלמידה מפוקחת נעשה על ידי עדכון המשקלים הסינפטיים של הרשת במהלך תהליך האימון. לכל דוגמה בסט האימון יש קלט ותגובה רצויה, והרשת לומדת לצמצם את הפער בין התגובה האמיתית שלה לתגובה הרצויה על ידי שינוי המשקלים שוב ושוב עד להשגת יציבות. גישה זו דומה להסקה סטטיסטית א-פרמטרית, שבה אין הנחות מוקדמות על המודל, אלא ההחלטות מתקבלות ישירות מתוך הדוגמאות. למשל, בסיווג תבניות, נדרש להתאים קלט לאחת מכמה קטגוריות מוגדרות. בגישה א-פרמטרית, גבולות ההחלטה בין הקטגוריות נקבעים ישירות על סמך הדוגמאות בסט האימון, ללא שימוש במודל הסתברותי כלשהו. הדבר יוצר דמיון משמעותי בין מיפוי קלט-פלט ברשתות עצביות לבין הסקה סטטיסטית א-פרמטרית, שבה אין הנחות מוקדמות על המודל.
- הסתגלות – לרשתות עצביות יש יכולת להסתגל לשינויים בסביבה על ידי עדכון המשקלים הסינפטיים. ניתן לאמן רשת מחדש כך שתהיה יעילה יותר אם תנאי הסביבה משתנים. כאשר הסביבה אינה קבועה והסטטיסטיקות משתנות, ניתן לעצב את הרשת כך שתעדכן את המשקלים תוך כדי עבודה. ככל שהמערכת יכולה להסתגל בצורה טובה יותר, תוך שמירה על יציבות, כך היא תהיה חסינה (robust) יותר לשינויים. עם זאת, יש להיזהר מהסתגלות מהירה מדי, שעלולה לגרום לתגובות לא יציבות להפרעות אקראיות. לכן, יש לבחור קבועי זמן מתאימים לעדכוני המשקלים.
- תגובה מבוססת ראיות – רשת עצבית יכולה לספק לא רק את הקטגוריה של הפלט אלא גם את רמת הביטחון בהחלטה, מה שיכול לשפר את הביצועים על ידי סינון החלטות לא חד-משמעיות.
- עמידות בכשלים – רשתות עצביות שמיושמות בחומרה יכולות להמשיך לתפקד היטב גם בתנאים לא אופטימליים, בגלל האופי המבוזר שלהן. נזק מקומי לנוירון או לקשר לא יגרום לקריסה כוללת.
- מימוש על גבי מעגלים משולבים – המקביליות של רשתות עצביות מאפשרת ביצוע מהיר של חישובים מורכבים, מה שהופך אותן למועמדות טובות למימוש על גבי מעגלי VLSI, המאפשרים התנהגות מורכבת.

- <u>אחידות אנליטית ועיצובית</u> – רשתות עצביות משתמשות באותם מרכיבים בסיסיים (נוירונים מלאכותיים) בתחומים שונים, מה שמקל על שיתוף תאוריות ואלגוריתמים בין רשתות שונות, וגם על יצירת רשתות מודולריות.

**ייצוג של רשתות נוירונים מלאכותיות**

הנוירון הוא יחידת עיבוד המידע הבסיסית הנחוצה לתפקוד רשת הנוירונים. באיור 7 ניתן לראות מודל של נוירון, היוצר את הבסיס לעיצוב רשת נוירונים מלאכותית. ניתן לזהות 3 אלמנטים בסיסיים במודל המוצג:




איור 7

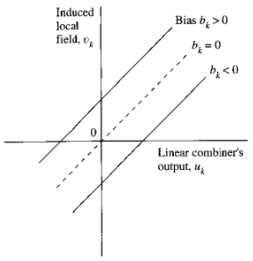
א. סט של סינפסות (או קישורים), אשר כל אחת מהן מאופיינת במשקל (או חוזק) משלה. הלכה מלעשה, אות כלשהו $x_j$, המגיע בתור קלט לסינפסה j המחוברת לנוירון k , מוכפל במשקל הסינפטי $w_{k,j}$.

ב. מחבר (adder) של אותות הקלט, לאחר הפעלת הפרמוטציה באמצעות המשקלים הסינפטיים. עד כה (הכפלת האותות בקבועים וסכימתם) יצרנו משלב ליניארי (linear combiner).

ג. פונקציית הפעלה (activation function) להגבלת האמפליטודה של אות הפלט. כפי שנראה בהמשך ,פונקציית ההפעלה היא הגורם לכך שפונקציית הפלט של הנוירון יכולה להיות אי-ליניארית.

בדרך כלל, הפלט בצורתו המנורמלת, לאחר הפעלת פונקציית ההפעלה, שוכן בטווח [0,1] אך לעיתים גם בטווח [-1,1] . כפי שניתן לראות באיור, ישנו עוד אות אשר מעובד ע״י הנוירון והוא איננו אות קלט. אות זה נקרא אות הטיה (bias) והוא מסומן בסימון $b_k$. מטרת אות זה היא הגברה או הנמכה של האות המסוכם טרם כניסתו לפונקציית ההפעלה. במינוח מתמטי ניתן לתאר נוירון k ע״י זוג המשוואות הבאות: $(1.1)\ u_k = \sum_{j=1}^{m} w_{kj}x_j$ וגם $(1.2)\ y_k = \varphi(u_k + b_k)$

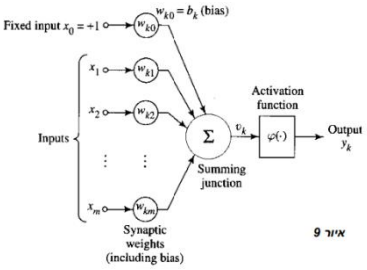
כאשר $x_1,x_2,\ldots x_m$ הינם אותות הקלט הנקראים גם מאפיינים (features), $w_{k1},w_{k2},\ldots w_{km}$ הינם המשקלים הסינפטיים של הנוירון, $u_k$ הוא הפלט של המשלב הליניארי על אותו הקלט, $b_k$ הוא אות ההטיה, $\Phi(\,.\,)$ היא פונקציית ההפעלה ו $y_k$ הוא אות הפלט של הנוירון. השימוש באות ההטיה נותן אפקט של טרנספורמציה אפינית (affine transformation) על הפלט של המשלב הליניארי כמתואר בנוסחה: $(1.3)\ v_k = u_k + b_k$




איור 8

הקשר בין הפלט של המשלב הליניארי $u_k$ לבין פוטנציאל ההפעלה ( activation potential or induced local field) מושפע מהחיוביות או השליליות של אות ההטיה כמתואר באיור 8.

כיוון שאות ההטיה הוא אות קלט (אמנם חיצוני אך עדיין אות קלט) ניתן לשנות את המשוואות (1.2) ו- (1.3) לצורה: $(1.4)\ v_k = \sum_{j=0}^{m} w_{kj}x_j$ וגם: $(1.5)\ y_k = \varphi(v_k)$




איור 9

כך שהוספנו אות קלט חדש (הוא אות ההטיה) וגם סינפסה חדשה אשר דרכה עובר האות. ניתן למצוא תיאור סכמתי של המתואר באיור 9. ערך האות החדש הוא: $(1.6)\ x_0 = +1$ והמשקל הסינפטי הוא: $(1.7)\ w_{k0} = b_k$

**רשתות עמוקות (Deep Neural Networks, DNN)** הן הרחבה של ANN , המורכבת ממספר רב של שכבות נסתרות (hidden layers) . הרעיון המרכזי ברשתות עמוקות הוא שהוספת שכבות מאפשרת לרשת ללמוד ייצוגים מורכבים יותר של המידע בקלט, ולזהות דפוסים נסתרים שאינם גלויים ברשתות רדודות.

**מבנה בסיסי של רשת נוירונים**

1. שכבת הקלט: השכבה הראשונה ברשת, שמקבלת את המידע הגולמי מהקלט.
2. שכבות נסתרות: שכבות ביניים המעבדות את הקלט באמצעות פונקציות אקטיבציה שונות. כל שכבה מוסיפה רמות מורכבות נוספות לייצוג של הקלט.
3. שכבת הפלט: השכבה האחרונה ברשת, שמספקת את התוצאה הסופית על פי המשימה (למשל, סיווג או חיזוי).

**תהליך הלמידה**

תהליך הלמידה של רשתות נוירונים כולל שני שלבים עיקריים:

1. העברת קדימה (Forward Propagation): בשלב זה הקלט עובר דרך הרשת, וכל נוירון מחושב על פי המשקל שלו. התוצאה היא פלט סופי שמתקבל בשכבת הפלט.
2. העברת אחורה (Backpropagation): לאחר קבלת הפלט, מחשבים את השגיאה על בסיס הפער בין הפלט הנחזה והפלט הרצוי. לאחר מכן השגיאה מחולקת ומשמשת לעדכון המשקלים ברשת.

**פונקציות אקטיבציה**

פונקציות האקטיבציה קובעות את אופן ההמרה של סכום הקלט של נוירון לפלט. דוגמאות לפונקציות נפוצות:

- ReLU (Rectified Linear Unit): מחזירה את הערך המקורי אם הוא חיובי, אחרת 0. משמשת לרוב ברשתות עמוקות.
- Sigmoid: מחזירה ערך בטווח בין 0 ל-1.
- Tanh: מחזירה ערך בטווח בין -1 ל-1.

**דוגמה: שימוש ברשתות נוירונים עמוקות לזיהוי הזרקת קוד**

זיהוי הזרקת קוד (Code Injection) הוא אתגר קריטי במערכות אבטחה, שכן מתקפות אלו מנצלות חולשות בקלט של התוכנה כדי להזריק קוד זדוני. דוגמה נפוצה לכך היא מתקפות SQL Injection, שבהן תוקף מצליח להכניס פקודות SQL זדוניות דרך קלט המשתמש במערכת מבוססת SQL .

**שלבי יישום רשת נוירונים לזיהוי מתקפות XSS :**

1. איסוף והכנת נתונים:
האתגר המרכזי בזיהוי מתקפות XSS הוא הבנת ההקשר שבו קוד זדוני מופיע בקלט. יש לאסוף דוגמאות של קלטים לגיטימיים ודוגמאות של קלטים המכילים קוד זדוני XSS, כולל:
   - תגי HTML זדוניים כמו <script>, <img> עם אירועים כמו onerror או onload.
   - שימוש באלמנטים כמו alert(), document.cookie, eval() בתוך תגי HTML .
   - תווים כמו ", &, <> או ' המשמשים לרוב לסיום והתחלת תגים חדשים.

תכונות רלוונטיות לקלטים אלו עשויות לכלול:

- נוכחות תווים מיוחדים: ", ', &, <, >.
- סוגי הפונקציות או האירועים שמופיעים בקוד: onerror, onload, alert(), eval().

- אורך הקלט: לעיתים קרובות קוד XSS ארוך ומורכב מקלט לגיטימי.
- מבנה התגיות והקשרים בין המילים והתווים.

2. בניית הרשת:
לצורך זיהוי מתקפות XSS , ניתן לבנות רשת נוירונים עמוקה שבה כל שכבה לומדת תכונות מורכבות יותר של הקלט. למשל, השכבות הראשונות עשויות ללמוד לזהות את תבניות התווים הזדוניים (כגון <script>), השכבות הבאות עשויות ללמוד את הקשרים הפנימיים בין האלמנטים השונים בקלט, ושכבות מתקדמות יותר יוכלו לזהות את דפוסי ההתנהגות המסוכנים שמצביעים על מתקפת XSS . הרשת יכולה לכלול שכבות קונבולוציה (Convolutional Layers) אם רוצים לבצע ניתוח מבוסס רצפים(Sequence-based analysis) של הקלט, או לחלופין שכבות Fully Connected לאחידות בעיבוד המידע.

3. אימון הרשת:
יש לאמן את הרשת על דוגמאות רבות של קלטים תקינים וזדוניים. במהלך האימון, הרשת תלמד לזהות את הדפוסים והתבניות שמאפיינים מתקפות XSS. במהלך תהליך ה-backpropagation, הרשת תעדכן את המשקלים שלה על סמך השגיאות בפלטים עד שתהיה מסוגלת לסווג בצורה מדויקת יותר קלטים זדוניים ותקינים.

4. בדיקה והערכה:
את הרשת בודקים על קבוצת בדיקה נפרדת שלא נכללה באימון כדי להעריך את יכולתה לזהות מתקפות XSS . מודדים את הצלחת הרשת באמצעות מדדים כמו דיוק (accuracy), רגישות (recall) , סף זיהוי כוזב (false positive rate) , ועוד.

5. פריסה ושימוש בזמן אמת:
לאחר האימון, הרשת יכולה לפעול בזמן אמת כדי לסרוק קלטים שנשלחים על ידי משתמשים באתר ולזהות ניסיונות להזריק קוד XSS זדוני. לדוגמה, אם משתמש שולח קלט הכולל את הקוד הבא:

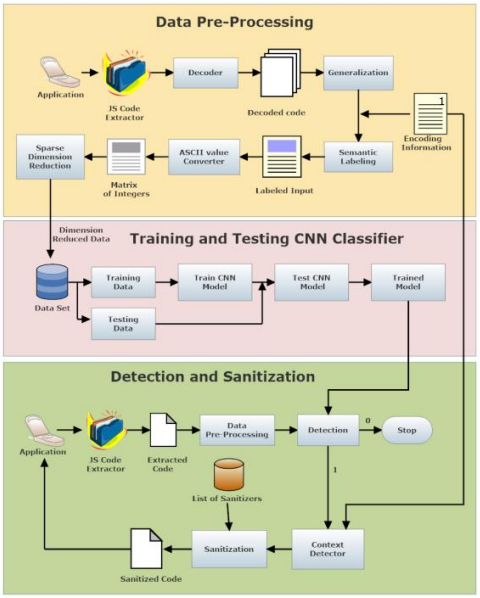


Fig. 4: Architecture of ConvXSS

```
<img src="x" onerror="alert('XSS')">
```

הרשת יכולה לזהות שמדובר במתקפה על פי ניתוח הדפוסים בקוד, לדוגמה:

- נוכחות תגים חשודים כמו <img>.
- שימוש בפונקציות כמו onerror שמשמשות לביצוע קוד JavaScript .
- פלט של הודעת alert() שהוא סימן מובהק לתקיפה פשוטה.

דרך פעולה של convXSS לזיהוי הזרקת קוד[5] (איור 4).

**יתרונות וחסרונות של רשתות נוירונים עמוקות בזיהוי הזרקת קוד**

**יתרונות:**

1. זיהוי דפוסים מורכבים: רשתות עמוקות מסוגלות ללמוד דפוסים מורכבים ביותר בקלטים, מה שמאפשר לזהות מתקפות שהן עדינות ואינן טריוויאליות.
2. יכולות התאמה עצמית: רשתות לומדות ומשתפרות עם הזמן, מתאימות את עצמן לשינויים בדפוסי המתקפות, מה שמוביל לשיפור עקבי בזיהוי מתקפות XSS.

3. <u>יכולת עבודה עם נתונים גולמיים</u> : רשתות עמוקות אינן דורשות תמיד מיפוי תכונות מתוחכם, ויכולות לעבוד עם נתונים גולמיים כמו טקסט ישירות. מסוגלות להתמודד עם קלטים מגוונים ומורכבים כמו HTML ו- JavaScript.

**חסרונות:**

1. <u>דרישות משאבים גבוהות</u> : רשתות נוירונים עמוקות דורשות כוח עיבוד גדול ואחסון זיכרון רב לאימון ולעבודה עם נתונים גדולים. יש צורך בחומרה מתקדמת (כמו GPU) כדי לאמן ולהפעיל את הרשת בצורה יעילה.

2. <u>מורכבות האימון</u> : אימון רשת נוירונים עלול לקחת זמן רב, במיוחד אם אין גישה לחומרה מתקדמת (כמו GPU). כמו כן, דרושה התאמה של פרמטרים שונים כגון גודל הרשת, פונקציות אקטיבציה, שיעור הלמידה ועוד.

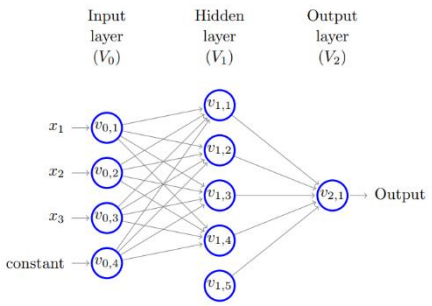
3. <u>חוסר פרשנות</u> : לעיתים קרובות קשה להבין את ההחלטות שמתקבלות ברשתות עמוקות. בעוד שמודלים פשוטים יותר כמו עצים החלטה יכולים לספק הסבר ברור לכל החלטה, רשתות עמוקות מתאפיינות ב"ארגז שחור", דבר המקשה על ההבנה מדוע קלט מסוים זוהה כמתקפה.

4. <u>נטייה להתאמת יתר (Overfitting)</u> : רשתות עמוקות עשויות ללמוד דפוסים ספציפיים לנתוני האימון בצורה כזו שהן אינן מתאימות טוב לנתונים חדשים. יש צורך באמצעים להפחתת תופעה זו, כגון regularization או dropout . כלומר, קיימת סכנה שהרשת תלמד את הדפוסים הייחודיים של הנתונים האלו ולא תצליח להכליל טוב מספיק על נתונים חדשים.

**רשתות עצביות- הזנה קדימה (Feedforward)**

הרעיון מאחורי רשתות עצביות הוא שתאי עצב רבים יכולים להיות מחוברים יחד על ידי קישורי תקשורת כדי לבצע חישובים מורכבים. מקובל לתאר את המבנה של רשת עצבית כגרף שהצמתים שלו הם תאי העצב וכל קצה (מכוון) בגרף מקשר בין הפלט של תא עצב כלשהו לקלט של תא עצב אחר.

רשת עצבית הזנה קדימה מתוארת על ידי גרף אציקלי מכוון, $G = (V, E)$, ופונקציית משקל על הקצוות, $w : E \to R$. צמתים של הגרף מתאימים לנוירונים. כל תא עצב בודד מעוצב כפונקציה סקלרית פשוטה, $\sigma : R \to R$. נתמקד בשלוש פונקציות אפשריות עבור σ : פונקציית הסימן, $\sigma(a) = \mathrm{sign}(a)$, פונקציית הסף, $\sigma(a) = 1_{[a>0]}$, ופונקציית הסיגמואיד (נפוצה במיוחד בלמידת מכונה, בעיקר ברשתות עצביות מלאכותיות, שבהן היא משמשת כפונקציית שפעול "activation function" לצורך הכנסת אי-לינאריות למודל) , $\sigma(a) = 1/(1 + \exp(-a))$, שהיא קירוב חלק לפונקציית הסף. אנו קוראים σ פונקציית "ההפעלה" של תא העצב. כל קצה בגרף מקשר את הפלט של תא עצב כלשהו לקלט של תא עצב אחר.




הקלט של תא עצב מתקבל על ידי לקיחת סכום משוקלל של התפוקות של כל הנוירונים המחוברים אליו, כאשר השקלול הוא לפי w. כדי לפשט את תיאור החישוב המבוצע על ידי הרשת, אנו מניחים עוד כי הרשת מאורגנת בשכבות. כלומר, ניתן לפרק את קבוצת הצמתים לאיחוד של תת-קבוצות מפורקות (לא ריקות), $V = \uplus_{t=0}^{T} V_t$, כך שכל קצה ב-E מחבר צומת כלשהו ב-$V_{t-1}$ לצומת כלשהו ב-$V_t$, עבור חלק מ-$t \in [T]$. השכבה התחתונה, $V_0$, נקראת שכבת הקלט. הוא מכיל $n + 1$ נוירונים, כאשר n הוא המימדיות של מרחב הקלט. עבור כל $i \in [n]$, התפוקה של נוירון i ב-$V_0$ היא פשוט $x_i$ . תא העצב האחרון ב-$V_0$ הוא תא העצב ה"קבוע", שתמיד פולט 1. אנו מציינים על ידי $v_{t,i}$ את תא העצב ה-i של שכבת t ועל ידי $o_{t,i}(x)$ את

הפלט של $v_{t,i}$ כאשר הרשת מוזנת עם וקטור הקלט x. לכן, עבור $i \in [n]$ יש לנו $o_{0,i}(x) = x_i$ ועבור $i = n + 1$ יש לנו $o_{0,i(x)} = 1$. כעת אנו ממשיכים בחישוב בצורה שכבה אחר שכבה. נניח שחישבנו את התפוקה של תאי העצב בשכבה t. לאחר מכן, אנו יכולים לחשב את התפוקה של הנוירונים בשכבה t + 1 כדלקמן. תקן כמה $v_{t+1}, j \in V_{t+1}$. תן ל- $a_{t+1,j}(x)$ לציין את הקלט ל- $v_{t+1,j}$ כאשר הרשת מוזנת עם וקטור הקלט x. אז

$$a_{t+1,j}(\mathbf{x}) = \sum_{r:\,(v_{t,r},v_{t+1,j})\in E} w((v_{t,r}, v_{t+1,j}))\, o_{t,r}(\mathbf{x}),$$

ו- $o_{t+1,j}(\mathbf{x}) = \sigma\left(a_{t+1,j}(\mathbf{x})\right)$

כלומר, הקלט ל- $v_{t+1,j}$ הוא סכום משוקלל של היציאות של הנוירונים ב- $V_t$ המחוברים ל- $v_{t+1,j}$. שכבות $V_1, \ldots, V_{T-1}$ נקראות לעתים קרובות שכבות נסתרות. השכבה העליונה, $V_T$, נקראת שכבת הפלט. בבעיות חיזוי פשוטות, שכבת הפלט מכילה נוירון יחיד שהפלט שלו הוא הפלט של הרשת. אנו מתייחסים ל-T כמספר השכבות ברשת (למעט $V_0$), או כ"עומק" הרשת (איור עמוד קודם).

השוואה בין הטכניקות השונות[4] (האיור שבעמוד).

| S. No. | Machine Learning Technique | Pros | Cons |
|---|---|---|---|
| 1 | Neural Networks | Ability to generalize from limited, noisy and incomplete data.<br>Does not need expert knowledge and it can find unknown or novel intrusions. | Slow training process so not suitable for real-time detection.<br>Over-fitting may happen during neural network training. |
| 2 | Support Vector Machine | Better learning ability for small samples.<br>High training rate and decision rate, insensitiveness to dimension of input data. | Training takes a long time.<br>Mostly used binary classifier which cannot give additional information about detected type of attack. |
| 3 | Fuzzy Logic | Reasoning is Approximate rather than precise.<br>Effective, especially against port scans and probes. | High resource consumption Involved.<br>Reduced, relevant rule subset identification and dynamic rule updation at runtime is a difficult task. |
| 4 | Decision Tree | Decision Tree works well with huge data sets.<br>High detection accuracy. | Building a decision tree is computationally intensive task. |

## <u>אתגרים בזיהוי הזרקות קוד באמצעות למידת מכונה</u>

זיהוי הזרקות קוד הוא אתגר מתמשך במערכת אבטחת המידע, במיוחד כאשר התקפות אלה מתפתחות עם הזמן והופכות ליותר מתוחכמות. למידת מכונה, עם היכולות שלה לזיהוי דפוסים ולמידה מהעבר, מספקת אפשרויות מרתקות לגילוי מתקפות הזרקת קוד. עם זאת, ישנם מספר אתגרים משמעותיים ביישום למידת מכונה לצורך זה, בהם איסוף נתונים, טיפול בבעיות חוסר איזון בנתונים, התמודדות עם התקפות אנטי-זיהוי, ותכנון מערכת לזיהוי יעיל. נפרט כעת את האתגרים והפתרונות המוצעים לכל אחד מהם, ונציג כיצד ניתן לפתח מערכת לזיהוי הזרקות קוד באמצעות למידת מכונה.

### 1. איסוף נתונים לאימון המודלים

השלב הראשון והקריטי ביותר בפיתוח מערכת למידת מכונה הוא איסוף נתונים עליהם המודל יוכל להתאמן וללמוד. לצורך גילוי הזרקות קוד, הנתונים חייבים לכלול דוגמאות של פעולות לגיטימיות כמו גם דוגמאות של התקפות זדוניות.

- <u>מגוון רחב של התקפות</u>: הזרקות קוד יכולות לבוא לידי ביטוי בהרבה דרכים שונות. לדוגמה:
  - <u>SQL Injection</u>: תקיפה שבה התוקף מחדיר פקודות SQL דרך קלט לא מבוקר במטרה להוציא מידע רגיש מהמסד נתונים.
  - <u>Cross-Site Scripting (XSS)</u>: מתקפה בה תוקף מזריק קוד JavaScript אל דפי אינטרנט במטרה לגנוב מידע מהמשתמשים.
  - <u>Command Injection</u>: מתקפה שבה התוקף מחדיר פקודות מערכת דרך ממשק הקלט של היישום.

  כל אחת מהתקפות אלה מצריכה איסוף נתונים מיוחדים שיכולים לשמש לאימון המודלים, ומבלי מגוון רחב של נתונים, המודל עלול להיכשל בזיהוי סוגים מסוימים של התקפות.

- <u>מחסור בנתונים אמיתיים / סטים לא מאוזנים</u>: קשה להשיג נתונים על תקיפות הזרקת קוד, כי ארגונים לא רוצים לחשוף פגיעויות או נתונים על התקפות. חוסר בתיעוד של תקיפות עלול להטות את מודלי הלמידה המלאכותית ולהפחית את היכולת לזהות פעילויות זדוניות.

- תיוג נתונים: תהליך תיוג הנתונים דורש כוח עבודה מקצועי ואמין, שיידע לזהות מהי פעולה תקינה ומהי התקפה. תיוג לא נכון של הנתונים עלול לגרום לאימון מודל לא יעיל, מה שעלול להוביל לבעיות בזיהוי עתידי של התקפות.
  - False Positives: כאשר המודל מזהה פעולה תקינה כזדונית, מה שיכול להוביל לשיבושים בפעילות המערכת ולפגוע בביצועים ובאמינות.
  - False Negatives: כאשר המודל נכשל בזיהוי מתקפה, ומגדיר אותה כפעולה תקינה. מצב זה מסוכן במיוחד שכן התקפה עלולה להתרחש מבלי שגורמי האבטחה יהיו מודעים לכך.
- פרטיות נתונים: איסוף ואחסון של כמויות גדולות של נתוני תנועת אינטרנט מעלה חששות בנוגע לפרטיות ,במיוחד כאשר מדובר במידע רגיש. הבטחת פרטיות הנתונים ועמידה בתקנות כמו GDPR חיונית.

**פתרונות אפשריים**:

- שימוש בנתונים סינתטיים: כדי להתגבר על המחסור בנתונים אמיתיים, ניתן ליצור נתונים סינתטיים המדמים התקפות הזרקת קוד. כמו,הרצת סימולציות של SQL Injection או XSS על מערכות ניסוי ולתעד את התוצאות. כך ניתן לאמן את המודל על בסיס נתוני התקפות מדומות שדומים למצבים אמיתיים.
- שימוש במאגרי נתונים ציבוריים: קהילת אבטחת הסייבר משתפת לעיתים נתונים ממאגרים ציבוריים, כגון מאגרי נתונים של וקטורי תקיפה. דוגמאות למאגרים כאלה כוללים את מאגר OWASP או CSIC 2010 HTTP Dataset , המכילים דוגמאות של תקיפות רשת.
- שיטות תיוג מתקדמות: שימוש בשיטות תיוג חצי אוטומטיות יכול להקל על תהליך תיוג כמויות גדולות של נתונים. ניתן להיעזר במודלים ראשוניים שיתייגו חלק מהנתונים, ולאחר מכן מומחי אבטחה יבדקו את נכונות התיוג.

**2. טיפול בחוסר איזון בנתונים**

אחד האתגרים הגדולים בלמידת מכונה הוא כאשר מערך הנתונים אינו מאוזן. במקרה של זיהוי הזרקות קוד, רוב הנתונים יהיו דוגמאות לפעולות חוקיות, בעוד שמספר הדוגמאות של התקפות יהיה קטן יחסית. מצב זה יוצר בעיית "חוסר איזון" (data imbalance) .

- מודלים מוטים: כאשר הנתונים אינם מאוזנים, המודל עלול להיות מוטה לטובת הקבוצה הגדולה (פעולות חוקיות) ולהיכשל בזיהוי התקפות נדירות, מה שיכול להוביל לתוצאה מוטעית.
- תלות במדדים סטנדרטיים: מדדי ביצוע כמו דיוק אינם מתאימים, שכן מודל יכול להציג דיוק גבוה למרות שלא מזהה את רוב ההתקפות.
- מורכבות המודל ופרשנות
  - התאמה יתר: מודלים מורכבים עלולים להתאים יתר על המידה לנתונים, מה שמוביל לביצועים גרועים על נתונים חדשים. טכניקות רגולריזציה יכולות לסייע.
  - פרשנות: הבנת ההחלטות של המודל חשובה לאמינותו, אך מודלים כמו רשתות נוירונים עמוקות יכולים להיות קשים לפרשנות, מה שמקשה על זיהוי הטיות או פגיעויות פוטנציאליות.
  - סטייה של המודל: אפייני תנועת האינטרנט יכולים להשתנות, ומודלים עשויים להזדקק לאימון מחדש באופן קבוע כדי להסתגל לאיומים חדשים.

**פתרונות אפשריים** :

- Oversampling ו-Undersampling : ניתן להשתמש בטכניקות של דגימה מחדש (Resampling) כדי להגדיל את מספר הדוגמאות מה- minority class (התקפות) או להקטין את מספר הדוגמאות מה- majority class (הפעולות התקינות). טכניקות כמו SMOTE (Synthetic Minority Over-sampling Technique) מאפשרות ליצור דוגמאות סינתטיות חדשות מהכיתה הזדונית.

- שימוש במודלים מתקדמים : מודלים כמו Random Forest ו- Gradient Boosting מצליחים להתמודד טוב יותר עם נתונים לא מאוזנים על ידי מתן משקלות שונים לדוגמאות בהתאם למורכבותן.

- מדדים אלטרנטיביים : כדי לקבל תמונה מדויקת יותר על ביצועי המודל, יש להשתמש במדדים כמו F1-Score (משקלול בין Precision ו-Recall), Precision-Recall Curve , או Area Under Precision-Recall Curve (AUPRC) . מדדים אלו נותנים תמונה טובה יותר לגבי היכולת של המודל לזהות התקפות נדירות.

**3. התמודדות עם התקפות אנטי-זיהוי**

בעידן הנוכחי, תוקפים אינם מסתפקים בניסיון הזרקות קוד בלבד, אלא גם מנסים להערים על מערכות הזיהוי ולהתחמק מהן. ההתקפות הללו מכונות "התקפות אנטי-זיהוי", והן נועדו להסוות את כוונותיהם האמיתיות.

- התקפות יריבות(adversarial attack) : תוקפים משנים שינויים מינוריים בקוד ההזרקה כדי לגרום לו להיראות לגיטימי בעיני המודל.

- התקפות פולימורפיות : התקפות שמשנות את המבנה שלהן עם כל ניסיון תקיפה, מה שמקשה על זיהוי הדפוסים המסורתיים.

- התחזות לתנועה חוקית : תוקפים מתחזים לפעולות חוקיות במערכת, כך שהמודל מתקשה להבחין בין פעולה רגילה לתקיפה.

**פתרונות אפשריים** :

- אימון כנגד יריבות : אימון המודל על דוגמאות זדוניות מסוות (adversarial examples) כדי לשפר את יכולתו להתמודד עם ניסיונות הסוואה עתידיים.

- שימוש במודלים מרובים : גישת Ensemble Learning שמבוססת על מספר מודלים שונים במקביל, מה שמגדיל את הסיכוי לזהות תקיפות.

**4. תכנון מערכת לזיהוי הזרקות קוד באמצעות למידת מכונה**

בניית מערכת לזיהוי הזרקות קוד מבוססת למידת מכונה דורשת תהליך מתודולוגי, שמתחיל באיסוף נתונים וניתוחם וממשיך בבחירה נכונה של האלגוריתמים, פיתוח המערכת, והבטחת יכולת לתחזוקה ותגובה בזמן אמת. ניתן לחלק את תהליך פיתוח המערכת למספר שלבים עיקריים :

שלב 1 : איסוף נתונים וטרום עיבוד

כפי שצוין, הנתונים הם הבסיס להצלחה של כל מערכת למידת מכונה. יש לאסוף נתונים נרחבים ומגוונים ממקורות שונים :

- מקורות נתונים : נתוני יומני מערכת (log files) , תעבורת רשת (network traffic) , דגימות תקיפות ידועות, ומאגרי מידע קוד פתוח המיועדים לאבטחת סייבר (כמו OWASP).

- טרום עיבוד הנתונים :

- ניקוי נתונים: למשל, הסרה של שדות מיותרים או כפילויות.
- הנדסת תכונות(feature engineering): זיהוי של תכונות רלוונטיות כגון תבניות בקלט המשתמש, רצפי בקשות, אורך הקלט, ותכונות נוספות שיכולות להעיד על ניסיונות תקיפה. הזיהוי המתבצע מתוך כמויות הנתונים העצומות המיוצרות על ידי יישומי אינטרנט הוא אתגר ובחירה בתכונות שגויות יכולה לפגוע בביצועי מודלי הלמידה המלאכותית. כמו כן, חילוץ תכונות משמעותיות מנתונים גולמיים ,כגון בקשות ותגובות HTTP, יכול להיות מורכב .טכניקות כמו n-grams, TF-IDF והטמעות מילים עשויות להידרש כדי ללכוד את התבניות הבסיסיות.
- הנדסת תכונות עבור התקפות מעורפלות: תוקפים יכולים להשתמש בטכניקות ערפול כדי להתחמק מזיהוי, ולכן מודלי למידה מלאכותית חייבים להיות בעלי מסוגלות לחלץ תכונות מקוד מעורפל ביעילות.
- תיוג נתונים: תיוג כל בקשה כתקינה או זדונית על מנת לאפשר אימון מפוקח למודל.

שלב 2: בחירת מודלים מתאימים

לאחר שלב איסוף וטרום עיבוד הנתונים, יש לבחור את המודלים המתאימים לאימון המערכת. הבחירה תלויה בטבע הנתונים והדרישות של המערכת. ציינו לפני כן את האלגוריתמים האפשריים האלה: עצים רנדומליים (Random Forests) המורכב מעצי החלטה (שיטה זו טובה במיוחד לזיהוי דפוסים מורכבים בנתונים), Support Vector Machine (SVM) המסוגל למצוא את קו ההפרדה (היפר-מישור) ייתן את ההפרדה המקסימלית בין דוגמאות חיוביות לשליליות ורשתות נוירונים עמוקות (Deep Neural Networks), המסוגלות להתמודד עם דפוסים נסתרים ולא ליניאריים בצורה יעילה במיוחד.

הכשרת המודל: המודל נדרש לעבור אימון ראשוני עם סט נתונים מפולח, כאשר יש להקפיד להפריד בין הנתונים המשמשים לאימון לבין אלה המיועדים לבחינה, כדי למנוע תופעת overfitting.

שלב 3: הערכת המודל ושיפורו

לאחר האימון, יש להעריך את ביצועי המודל כדי לוודא שהוא מסוגל לזהות התקפות זדוניות ביעילות תוך שמירה על מספר קטן של אזעקות שווא. נוכל להשתמש במדדי ביצועים האלה כדי להעריך את המודל:

- דיוק (Accuracy): אחוז המקרים בהם המודל צדק.
- Precision: מדד המודד את אחוז התחזיות הנכונות מתוך התחזיות החיוביות.
- Recall: אחוז הדוגמאות החיוביות שהמודל הצליח לזהות מתוך כלל הדוגמאות החיוביות.
- F1 Score: משקלול בין Precision ו-Recall.
- Area Under Curve (AUC): מדד המתאר את הביצועים של המודל ביחס לאיזון בין אזעקות שווא והתקפות אמיתיות.

- אופטימיזציה: יש לשפר את המודל במידת הצורך באמצעות כוונון של פרמטרים שונים (Hyperparameters) והתאמתם לתנאי המערכת ולסוגי התקפות חדשות.

שלב 4: פריסה ומעקב רציף

לאחר אימון והערכת המודל, ניתן לפרוס אותו בסביבת אמת, אך יש לוודא שהפריסה נעשית בצורה מתודולוגית כך שהמודל יוכל להשתלב במערכת הקיימת באופן שקוף ומאובטח.

- אינטגרציה עם מערכות קיימות : יש לוודא שהמודל יכול לפעול בשיתוף פעולה עם מערכות האבטחה הקיימות (כגון חומות אש, מערכות SIEM וכו'). ניתן להשתמש בטכנולוגיות כגון Docker לצורך קונטיינרים מאובטחים לפריסת המודל.

- ניטור ובקרה : חשוב מאוד להמשיך לנטר את ביצועי המודל גם לאחר פריסתו. יש לוודא שהמודל שומר על רמת זיהוי גבוהה ואינו מייצר אזעקות שווא רבות, ולהגיב בהתאם לשינויים בדפוסי התקיפות. נוכל להשתמש בכלי ניטור ובקרה כגון Grafana לניתוח גרפי של ביצועי המודל בזמן אמת. ELK Stack (Kibana, Logstash,Elasticsearch ) היא גם דוגמה לפתרון מבוסס קוד פתוח המאפשר לנטר את הביצועים של מערכות בזמן אמת.

שלב 5 : תחזוקה ועדכונים

זיהוי הזרקות קוד אינו תהליך חד-פעמי, שכן האיומים משתנים כל הזמן. על המודל להתעדכן על בסיס קבוע על מנת לשמור על רמת הגנה גבוהה.

- עדכוני מודלים : תוקפים מתפתחים ומשתמשים בטכניקות חדשות כדי לעקוף את מנגנוני הזיהוי. יש צורך לעדכן את המודלים עם נתונים חדשים ומעודכנים, כמו גם להתאים את המודל לשינויים בטכנולוגיה ובהתקפות חדשות.

- מנגנוני למידה רציפה : ניתן להשתמש במנגנוני למידה רציפה (Continual Learning) כדי שהמודל יתעדכן אוטומטית לפי הדפוסים שהוא מזהה בזמן אמת. כך ניתן לשמור על מערכת מאובטחת לאורך זמן, מבלי להפסיק את פעילותה לצורך עדכון.

**בעיות נוספות**

- התקפות עוינות: תוקפים יכולים ליצור קלטים זדוניים שנועדו להטעות מודלי למידה מלאכותית, כך שיתפסו כתקינים.
- העלמה: שימוש בטכניקות העלמה להסוואת קוד זדוני, מה שמקשה על המודלים לזהותו.
- התקפות אפס יום: התקפות חדשות ולא ידועות שעשויות להופיע, מה שמקשה על המודלים לזהות אותן ללא נתוני אימון מוקדמים.
- דרישות משאבים : אימון ופריסה של מודלים עשויים להיות יקרים מבחינה חישובית, ודורשים משאבי מחשוב בעלי ביצועים גבוהים לניתוח נתונים גדולים ומודלים מורכבים.
- קנה מידה: מערכות למידה מלאכותית צריכות להיות ניתנות להרחבה כדי להתמודד עם עלייה בנפח התנועה והופעת וקטורי התקפה חדשים, כשמסגרות מחשוב מבוזרות יכולות לסייע בכך.

**פתרונות נוספים**

- גישות היברידיות : שילוב בין זיהוי מבוסס חתימות לבין טכניקות למידה מכונה יכול לספק הגנה מקיפה יותר.

- שיטות אנסמבל: אנסמבלים של מספר מודלי למידה מלאכותית יכולים לשפר את העמידות והדיוק.

- בינה מלאכותית ניתנת להסבר: פיתוח טכניקות להפיכת מודלי למידה מלאכותית לניתנים לפרשנות יותר יכול לעזור לזהות הטיות ופגיעויות פוטנציאליות.

- למידה מתמשכת: אימון מחדש קבוע של מודלי למידה מלאכותית על נתונים חדשים יכול לעזור להם להסתגל לאיומים מתפתחים.

על ידי התמודדות עם אתגרים אלה וניצול ההתקדמות האחרונה בתחום הלמידה המלאכותית , ניתן לפתח פתרונות יעילים וניתנים להרחבה לזיהוי הזרקות קוד והגנה על יישומי אינטרנט מפני התקפות זדוניות.

**כלים טכנולוגיים לפיתוח מערכת זיהוי הזרקות קוד**

כלים טכנולוגיים שיאפשרו את פיתוח המערכת בצורה יעילה ומקצועית:

- Scikit-learn: ספריית Python המכילה כלים לפיתוח מודלי למידת מכונה בסיסיים כגון עצים רנדומליים ו-SVM.
- TensorFlow ו-PyTorch: מסגרות עבודה מתקדמות לפיתוח רשתות נוירונים ורשתות עמוקות. הן תומכות בשימוש במודלים מתקדמים המאפשרים התמודדות עם דפוסים מורכבים ביותר.
- Docker: מאפשר להריץ את המודלים בסביבת קונטיינר מבודדת, שמאפשרת פריסה מאובטחת של המודל בסביבת הארגון.
- Grafana ו-ELK Stack- מספקים פתרונות ניטור ואנליזה, שמסייעים לעקוב אחר ביצועי המודל בזמן אמת ולהבטיח תגובה מהירה לאיומים חדשים.

# פרק 4 - גישות למידת עמוקה/מכונה לזיהוי הזרקות קוד

## רשתות נוירונים קונבולוציוניות -CNN (למידה מפוקחת/לא מפוקחת)

רשתות עצביות קונבולוציוניות (CNNs) הן כלים חזקים ויעילים גם בזיהוי הזרקות קוד. למרות שהן פותחו במקור כדי לנתח תמונות ולזהות דפוסים ויזואליים, יכולתן לזהות תבניות מקומיות הפכה אותן לשימושיות גם בזיהוי תבניות זדוניות בקוד תוכנה.

**מרכיבים עיקריים של CNNs :**

1. שכבת קונבולוציה : מבצעת פעולות קונבולוציה על נתוני הקלט ומחפשת תבניות חוזרות באזורים מקומיים, כאשר תכונות פשוטות מזוהות בשכבות הראשונות, ותכונות מורכבות יותר, כמו תבניות קוד בעייתי, בשכבות העמוקות יותר.
2. שכבת איגום (Pooling) : תפקידה להקטין את ממדי הנתונים כדי לצמצם את הסיבוכיות החישובית ולמנוע התאמת יתר. התהליך הנפוץ הוא Max Pooling, שבו נשמר הערך הגבוה ביותר מתוך אשכול נתונים.
3. שכבה מחוברת לחלוטין : זו השכבה שבה מחברים את כל הנוירונים מכל שכבות קודמות כדי לבצע סיווג סופי או חיזוי. לעיתים קרובות משתמשים בפונקציית Softmax לסיום התהליך ולקביעת התוצאה.
4. פונקציות הפעלה(Activation Functions) : אלו תפקודים לא ליניאריים כמו ReLU , המוסיפים אי-ליניאריות לתהליך הלמידה ומאפשרים לרשת ללמוד תבניות מורכבות יותר.

**ארכיטקטורה של CNNs :** המבנה של CNN מורכב משכבות קלט, שכבות מוסתרות ושכבת פלט, כאשר הסידור שלהן מאפשר חילוץ מידע בצורה היררכית. שכבות הקונבולוציה מזהות תבניות מקומיות, שכבות האיגום מצמצמות את המידע ושכבות מחוברות לחלוטין מבצעות את הסיווג הסופי.

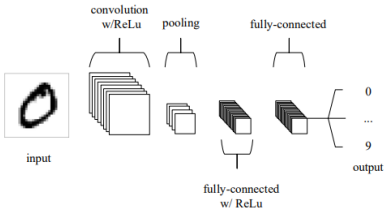


Fig. 2: An simple CNN architecture, comprised of just five layers

ארכיטקטורה של CNN [6] (איור 2)

**אימון CNNs :** האימון כולל תהליך התפשטות לאחור (Backpropagation) וירידה הדרגתית (Gradient Descent) כדי למזער את השגיאות, תוך שימוש בנתונים מתויגים כדי לשפר את יכולות הזיהוי של הרשת. טכניקות כמו נשירה (Dropout) ונורמליזציה של אצווה ( Batch Normalization) עוזרות למנוע התאמת יתר.

**יתרונות של CNNs :**

- זיהוי אוטומטי של תכונות מהקלט ללא צורך בחילוץ ידני.
- חסינות יחסית לשינויים במיקום ובמבנה של תבניות זדוניות בקוד.
- פשטות מבנה השיתוף במשקל, שמקטינה את המורכבות החישובית ומונעת התאמת יתר.

**מגבלות של CNNs :**

- קושי בפרשנות של התוצאות שהרשת מספקת, מה שמקשה על אימוץ טכנולוגיה זו בתחומים קריטיים כמו אבטחת מידע.
- פגיעות להתקפות יריבות, שבהן שינוי קטן בקלט יכול להוביל לתוצאות שגויות.
- תלות בכמות גדולה של נתוני אימון מתויגים לצורך ביצועים מיטביים.

רשתות עצביות קונבולוציוניות מהוות כלי יעיל לזיהוי תבניות זדוניות בקוד, אך כמו כל טכנולוגיה, הן דורשות הבנה זהירה של היתרונות והמגבלות שלהן.

### CNNs בלמידה מפוקחת לזיהוי הזרקות קוד

ביישומים לזיהוי הזרקות קוד, הקלט יכול להיות מיוצג כרצפי תווים, מחרוזות או מבנים בינאריים, והייצוגים הללו יכולים להיכנס לרשת CNN. הרשת לומדת את מבנה הקוד באמצעות שכבות קונבולוציה, ומאתרת תבניות רלוונטיות שמעידות על קוד זדוני. לדוגמה, היא עשויה לזהות תבניות כמו ניסיונות הזרקת SQL, באמצעות זיהוי מחרוזות חשודות כמו OR 1=1 או שימוש מופרז בפונקציות כמו SELECT.

### דוגמה ליישום

במאגר נתונים לזיהוי מתקפות SQL Injection, כל קטע קוד מסומן כ"בטוח" או "זדוני". רשת ה- CNN מקבלת את הקטעים ועוברת דרך מספר שכבות קונבולוציוניות, שם היא מבצעת חישובי קונבולוציה כדי לזהות קשרים ודפוסים ברצף הקוד. לדוגמה, תו מסוים יכול להופיע לעתים קרובות בהקשר של מתקפה, והרשת לומדת שכשנוכח בתבנית מסוימת, יש סיכוי גבוה שמדובר במתקפה. לאחר סיום הלימוד, ניתן להפעיל את הרשת על קוד חדש כדי לקבוע אם הוא בטוח או זדוני.

### יתרונות של CNNs בלמידה מפוקחת

1. <u>זיהוי דפוסים מקומיים</u>: CNNs מצטיינות בזיהוי דפוסים מקומיים, מה שמאפשר להן לזהות תבניות רציפות בקוד כמו שמזהות פיקסלים סמוכים בתמונות.
2. <u>אוטומציה של זיהוי מאפיינים</u>: CNNs אוטומטיות את תהליך זיהוי המאפיינים, ומסוגלות לזהות מתקפות ללא צורך בהתערבות אנושית.
3. <u>יכולת סקיילינג</u>: לאחר האימון CNNs ,מסוגלות להתמודד עם כמות גדולה של נתונים ולעבד קטעי קוד במהירות.
4. <u>ביצועים בזמן אמת</u>: CNNs יכולות לזהות מתקפות בזמן אמת, מיד עם הזנת הקוד למערכת, ובכך להגן על מערכות תוכנה מפני הזרקות.

### חסרונות בלמידה מפוקחת

1. <u>משאבים חישוביים גבוהים</u>: אימון רשת CNN דורש כוח חישובי רב, במיוחד עם נתונים מורכבים כמו קוד, מה שעשוי להוות אתגר בפרויקטים עם מגבלות משאבים.
2. <u>תלות במאגר נתונים מתויג</u>: כדי לאמן רשת CNN בצורה טובה, יש צורך במאגר נתונים מתויג. יצירת מאגר כזה דורשת זמן, משאבים, וכוח עבודה מיומן. ללא דאטה מתויג איכותי, הרשת עלולה לא להיות מסוגלת לזהות מתקפות בצורה מדויקת.

### CNNs בלמידה לא מפוקחת לזיהוי הזרקות קוד

בלמידה לא מפוקחת, המערכת מזהה דפוסים בנתונים ללא תוויות, מה שמאפשר לה לזהות מתקפות חדשות שלא היו ידועות בעבר. במקום להסתמך על קוד מתויג, הרשת מתמקדת בזיהוי דפוסים חריגים שמעידים על פעילות לא נורמלית. לדוגמה, רשת CNN יכולה לנתח קטעי קוד ממערכת תוכנה וללמוד את המבנה של קוד "בטוח". כאשר היא נתקלת בקוד חדש שונה מהדפוסים המוכרים, היא עשויה לסמן אותו כחשוד, דבר שעשוי להעיד על ניסיון להזרקת קוד.

### דוגמה ליישום

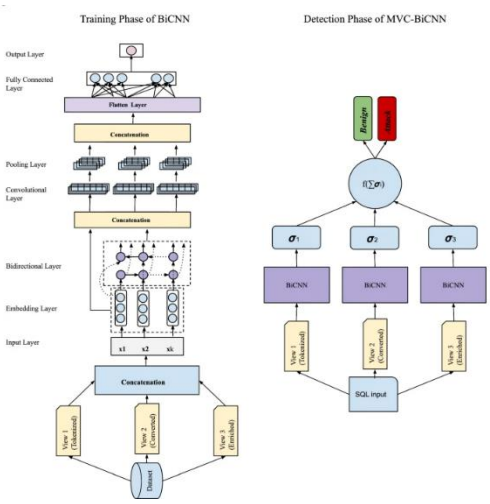
נניח שאנחנו מעוניינים לזהות מתקפות הזרקת SQL שלא זוהו קודם. תחילה, נאמן את הרשת על מאגר נתונים המכיל קוד נורמלי ותקין. לאחר מכן, כאשר הרשת תיתקל בקטע קוד בעל מבנה שונה או בלתי צפוי, כמו שורות קוד המשלבות שימוש בלתי רגיל במשתנים או מחרוזות קלט לא צפויות, היא תוכל לזהות זאת כחריגה.

**יתרונות של CNNs בלמידה לא מפוקחת**

1. יכולת לזהות מתקפות חדשות: המערכת לא מוגבלת למאגר נתונים מתויג, ומסוגלת לזהות חריגות מהמבנה התקין של הקוד, דבר שעשוי להוביל לזיהוי מתקפות בלתי מוכרות.
2. גמישות בנתונים: למידה לא מפוקחת לא דורשת מאגר נתונים מתויג. זהו יתרון משמעותי כאשר קשה ליצור תוויות מדויקות לכל קטעי הקוד.
3. פשטות בניהול הדאטה: מכיוון שאין צורך בתוויות, התהליך של הכנת הדאטה לאימון המערכת פשוט יותר בהשוואה ללמידה מפוקחת.

**חסרונות בלמידה לא מפוקחת**

1. אזעקות שווא: המערכת עשויה לסמן קטעי קוד לגיטימיים כחשודים, מה שמוביל לעלייה במספר האזעקות השגויות ולתהליך סינון נוסף שגוזל זמן.
2. מורכבות אימון: קשה להעריך את ביצועי הרשת ללא תוויות, ולכן היא מתאימה את עצמה במהלך תהליך הלמידה, דבר שמאתגר יותר בהשוואה ללמידה מפוקחת.
3. קושי בזיהוי מתקפות ספציפיות: למידה לא מפוקחת אינה תמיד מסוגלת לזהות את סוג המתקפה באופן מדויק, כי היא לא לומדת מהמאפיינים של מתקפות ידועות.




שימוש בשיטות אלו במערכות אבטחת מידע מהווה אבן דרך חשובה בהגנה על תוכנה מפני מתקפות זדוניות, ובמיוחד במניעת מתקפות הזרקת קוד.

לדוגמא שלב האימון ושלב הזיהוי של Bidirectional CNN[8] (איור ליד).

**רשתות נוירונים חוזרות (RNNs) לזיהוי הזרקות קוד (למידה מפוקחת/לא מפוקחת)**

**RNNs בלמידה מפוקחת לזיהוי הזרקות קוד**

רשתות נוירונים חוזרות (RNNs) הן סוג של רשתות נוירונים המיועדות להתמודד עם מידע סדרתי או תלוי זמן, כלומר, הן מסוגלות לשמר מידע מהעבר ולעבד אותו יחד עם המידע הנוכחי. יכולת זו הופכת אותן לכלי עוצמתי במיוחד בזיהוי תבניות בקוד, כולל זיהוי מתקפות הזרקת קוד.

בזיהוי הזרקות קוד, נתונים יכולים להגיע בצורת רצפי טקסטים (כמו SQL) או סדרות של קריאות פונקציה שמגיעות ממערכות קלט שונות. RNNs יכולות לנתח כל פקודה בתורה, לשמור מידע חשוב מהפקודות הקודמות ולהשתמש בו כדי לנתח את הפקודות הבאות. לדוגמה, הזרקת SQL לא תמיד מתגלה על ידי פקודה בודדת; לעיתים מתקפה מורכבת ממספר פעולות עוקבות שמובילות בסופו של דבר לניצול חולשה.

**דוגמה ליישום**

ביישום מעשי לזיהוי מתקפות הזרקת קוד, ניתן לאמן את רשת ה-RNN על מאגר נתונים המכיל דוגמאות של רצפי קוד הכוללים הזרקות קוד SQL וקוד תקין. המערכת תלמד לזהות את הדפוסים הקשורים להזרקת קוד, כמו ניסיונות לא חוקיים לשנות שאילתות דינמיות. לדוגמה, הרשת תוכל ללמוד להבחין ברצף פקודות כגון:

```
SELECT * FROM users WHERE username='admin'--
```

שמעיד על ניסיון לעקוף את המערכת ולהשיג גישה לא מורשית.

הרשת תחילה תקלוט את הפקודות הנכנסות ותעבור שלב אחרי שלב על כל פקודה, תוך שהיא זוכרת את הפקודות הקודמות. כאשר היא מזהה דפוס שמזכיר מתקפה, היא תוכל לסמן את הקוד כחשוד ולהתריע על כך.

**יתרונות של RNNs בלמידה מפוקחת**

1. יכולת לזכור מידע רציף: היתרון המרכזי של RNNs הוא יכולתן לשמר מידע מהעבר ולהשתמש בו בעת עיבוד הנתונים החדשים. זה חשוב מאוד בזיהוי מתקפות הזרקת קוד, שכן פעמים רבות מתקפות מתפרשות על פני מספר פעולות ולא ניתנות לזיהוי באמצעות ניתוח של פעולה אחת בלבד.
2. יכולת ניתוח דפוסים רציפים בקוד: ברשתות RNN, כל קלט תלוי בקלטים הקודמים לו. לכן, הן מתאימות במיוחד למצבים בהם יש צורך לעקוב אחרי רצף פעולות כדי לזהות מתקפה.
3. זיהוי מתקפות מורכבות: היכולת של RNN לשמר מידע מפעולות קודמות מאפשרת לזהות דפוסים מורכבים של מתקפות, גם כאשר הפקודות נראות לגיטימיות בתחילה.
4. שיפור הדיוק בלמידה: בזכות השימוש בתבניות רציפות, RNNs עשויות לספק רמת דיוק גבוהה יותר באיתור מתקפות הזרקת קוד בהשוואה לשיטות אחרות.

**חסרונות של RNNs בלמידה מפוקחת**

1. בעיה עם מידע ארוך טווח: RNNs נוטות לשכוח מידע רחוק ככל שהן מתקדמות ברצף, מה שעלול להוביל לאובדן מידע קריטי.
2. אימון ממושך: תהליך האימון מורכב ודורש זמן רב, במיוחד כאשר מדובר ברצפים ארוכים.
3. דרישות חישוביות גבוהות: אימון ופריסה של RNNs מצריכים משאבים חישוביים רבים, במיוחד עם כמויות גדולות של נתונים רציפים.

**RNNs בלמידה לא מפוקחת לזיהוי הזרקות קוד**

רשת RNN בלמידה לא מפוקחת יכולה ללמוד את ההתנהגות התקינה של קוד ולזהות כאשר יש סטייה מההתנהגות הנורמלית (דפוס חריג שיכול להעיד על הזרקת קוד). לדוגמה, היא יכולה לנתח רצפי פקודות בקוד וללמוד מהו הרצף התקין והנורמלי שמתקיים בשגרה. כאשר היא תזהה חריגה בדפוסים המוכרים, היא תוכל להצביע על סיכון פוטנציאלי.

**דוגמה ליישום**

נניח שאנחנו רוצים לזהות מתקפות חדשות. ניתן להפעיל רשת RNN לא מפוקחת על מאגר נתונים גדול של קוד תקין, כדי ללמד את המערכת דפוסים שגרתיים. לאחר האימון, אם המערכת נתקלת ברצף פקודות חריג, היא תסמן אותו כחשוד. לדוגמה, אם הקוד בדרך כלל כולל שאילתות SQL רגילות והרשת מזהה פקודות בלתי צפויות שמעידות על ניסיון הזרקת קוד, היא תתריע על כך.

**יתרונות של RNNs בלמידה לא מפוקחת**

1. זיהוי מתקפות חדשות: המערכת לא תלויה בדוגמאות מתויגות, והיא מזהה מתקפות חדשות דרך דפוסים חריגים בקוד.
2. גמישות רבה יותר בנתונים: אין צורך במאגר נתונים מתויג, מה שמאפשר גמישות רבה יותר בסוגי הנתונים לאימון.

3. יכולת הסתגלות לסביבות חדשות: בלמידה לא מפוקחת, רשת RNN יכולה להתאים את עצמה לדפוסים חדשים במערכת עם הזמן. זה חשוב במיוחד במערכות שמתעדכנות לעיתים קרובות, וזקוקות לפתרונות גמישים לאבטחת מידע.

**חסרונות של RNNs בלמידה לא מפוקחת**

1. קושי בזיהוי ספציפי של מתקפות: מאחר שהרשת מתמקדת בזיהוי חריגות כלליות, ייתכן שהיא לא תוכל לזהות את סוג המתקפה באופן מדויק, אלא רק לסמן את הקוד כחשוד.
2. אזעקות שווא: הרשת עלולה לייצר אזעקות שווא (false positives), כלומר, לסמן קוד תקין כחשוד. כך עלולה לגרום לירידה ביעילות המערכת.
3. מורכבות ניהול הדאטה: למרות שאין צורך בתוויות, יש צורך בכמות גדולה של דאטה או חשיפה למגוון רחב של תרחישים.

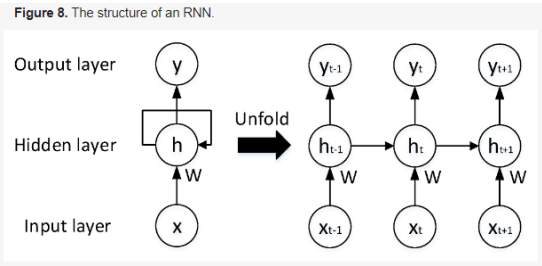


**Figure 8.** The structure of an RNN.

ארכיטקטורה של RNN [9] (איור 8)

**למידת מכונה מבוססת רשת עצבית לזיהוי הזרקות קוד (למידה מפוקחת/לא מפוקחת)**

**למידת מכונה מבוססת רשת עצבית בלמידה מפוקחת לזיהוי הזרקות קוד**

רשת עצבית מאומנת על נתונים מתויגים של קוד לגיטימי מול קוד זדוני, כולל הזרקות SQL, קריאות מערכת והזרקות בצד הלקוח. לאחר האימון, היא יכולה לנתח קוד חדש על סמך התבניות שלמדה. אם היא אומנה לזהות מתקפות הזרקת SQL, היא תזהה קלטים זדוניים כמו 'OR 1=1; --' ותתריע על המתקפה.

**דוגמה ליישום**

ראשית, נבנה מאגר נתונים עם דוגמאות של שאילתות לגיטימיות וזדוניות, מתויגות כ"לגיטימי" או "הזרקה". לאחר מכן, נבנה רשת עצבית עמוקה (deep neural network) שתלמד להבדיל בין השאילתות. הרשת תזהה דפוסים כמו מבנים של שאילתות ותחביר להזרקות. לאחר האימון, היא תנתח שאילתות חדשות ותחליט אם מדובר במתקפה או בקוד לגיטימי, כמו זיהוי ניסיון לעקוף אימות עם תווים מיוחדים.

**יתרונות של רשת עצבית בלמידה מפוקחת**

1. יכולת זיהוי מתקדמת: רשתות עצביות מזהות דפוסים מורכבים, מה שמאפשר להן לזהות מתקפות הזרקת קוד גם אם הן מוסתרות בקוד לגיטימי.
2. יכולת ללמוד מתקפות מגוונות: הן לומדות מתבניות שונות, מה שמקנה להן יכולת לזהות מגוון רחב של מתקפות, פשוטות ומורכבות.
3. שיפור מתמשך: הזנת נתונים נוספים במהלך האימון משפרת את יכולת הזיהוי של מתקפות חדשות.
4. אוטומציה של זיהוי מתקפות: הרשת מזהה מתקפות בזמן אמת, מפחיתה את הצורך בהתערבות ידנית.

**חסרונות של רשת עצבית בלמידה מפוקחת**

1. צורך בכמות נתונים גדולה לאימון: נדרש מאגר נתונים גדול שמכיל דוגמאות מתויגות של קוד זדוני ולגיטימי. הכנת מאגר נתונים כזה היא משימה מורכבת וגוזלת זמן.
2. קושי בהתמודדות עם מתקפות חדשות: יתכן שהרשת לא תזהה מתקפות חדשות שלא נלמדו בעבר.
3. דרישות חישוביות גבוהות: תהליך האימון דורש משאבים רבים ועיבוד מורכב.

**למידת מכונה מבוססת רשת עצבית בלמידה לא מפוקחת לזיהוי הזרקות קוד**

רשת עצבית בלמידה לא מפוקחת יכולה ללמוד את המבנה הבסיסי של קוד במערכת מסוימת ולזהות אם קוד כלשהו חורג מהמבנה הצפוי או מתנהג בצורה לא תקינה. זיהוי חריגות מאפשר לרשת לזהות מתקפות חדשות שלא נראו בעבר, גם אם אין דוגמאות מתויגות של המתקפות הללו.

**דוגמה ליישום**

נניח שמערכת מבוססת רשת עצבית לומדת את כל הקוד הנכנס לאפליקציה מסוימת ומשווה אותו לקוד התקין של האפליקציה. אם רשת העצבים מזהה קוד חדש שלא מתיישר עם הדפוסים שהיא למדה כתקינים, כמו למשל רצף של פקודות SQL לא צפויות, היא תסמן את הקוד כחריג. בכך המערכת יכולה לאתר ניסיונות חדשים למתקפות, גם אם הם לא הופיעו בעבר במאגרי הנתונים של הארגון.

**יתרונות של רשת עצבית בלמידה לא מפוקחת**

1. <u>זיהוי מתקפות חדשות</u>: מתמקדת בזיהוי דפוסים חריגים, מה שמאפשר לה לזהות מתקפות חדשות או וריאציות שלא נלמדו.
2. <u>יכולת זיהוי דינמית</u>: מתעדכנת בזמן אמת לדפוסים חדשים במערכת, חשובה במערכות שמתעדכנות לעיתים קרובות.
3. <u>זיהוי אוטומטי של חריגות</u>: מזהה התנהגות לא שגרתית ללא הגדרה מוקדמת של כללים, יעיל במקרים שבהם חוקי אבטחה מסורתיים לא פועלים.

**חסרונות של רשת עצבית בלמידה לא מפוקחת**

1. <u>אזעקות שווא רבות</u>: עלולה לסמן קוד תקין כחשוד, דבר שמפחית את האפקטיביות של המערכת.
2. <u>קושי בזיהוי מתקפות מורכבות</u>: ייתכן שלא תוכל לזהות את סוג המתקפה או להבין את כוונתו.
3. <u>צורך בנתונים רבים ללמידה</u>: עדיין דרושה כמות גדולה של נתונים כדי ללמוד דפוסים שגרתיים ולהיחשף למגוון רחב של תרחישים.

**<u>למידת מכונה לזיהוי הזרקות קוד באמצעות מודלים מבוססי אשכולות(Clustering)</u>**

בזיהוי מתקפות הזרקת קוד, טכניקות מבוססות אשכולות (Clustering) יכולות לספק גישה יעילה ואפקטיבית לניתוח אוטומטי של דפוסים במערכות שונות, במיוחד כאשר לא קיים ידע מוקדם על דפוסי התקפה מסוימים.

**למידת מכונה מבוססת אשכולות בלמידה לא מפוקחת לזיהוי הזרקות קוד**

למידה לא מפוקחת מבוססת אשכולות מיפתה נתונים לקטגוריות ללא תוויות מראש. האלגוריתם אוסף נתונים לא מתויגים, יוצר אשכולות על פי דמיון, ומזהה חריגות שמעידות על מתקפות הזרקת קוד. כשקוד חדש נכנס, האלגוריתם בודק אם הוא תואם לאשכולות הקיימים או חורג מהם, מה שעשוי להעיד על הזרקת קוד.

**דוגמה ליישום אשכולות לא מפוקחים**

נניח שמפתחים מודל לזיהוי מתקפות SQL. האלגוריתם לומד את השאילתות ומסווג אותן לאשכולות לפי מבנה. כאשר מתבצעת שאילתה חדשה, האלגוריתם מחליט אם היא מתאימה לאשכולות הקיימים. שאילתות חריגות, כמו שימוש בתווים לא צפויים, יזוהו כחשודות.

**יתרונות של שימוש באשכולות בלמידה לא מפוקחת**

1. <u>זיהוי מתקפות חדשות ובלתי צפויות</u>: יכולת לזהות דפוסים שלא נראו בעבר, מה שמסייע בזיהוי מתקפות בלתי מתועדות

2. אין צורך בנתונים מתויגים מראש: המערכת פועלת גם ללא דוגמאות מתויגות, מה שחוסך זמן ומשאבים.

3. יכולת גמישה לשימוש במערכות מגוונות: אלגוריתמים יכולים לפעול על סוגים שונים של נתונים במערכות משתנות.

4. יכולת לזהות חריגות: המודלים מזהים שינויים בתבניות קוד או בהפעלת פונקציות, מה שעשוי להעיד על ניסיונות הזרקת קוד.

**חסרונות של שימוש באשכולות בלמידה לא מפוקחת**

1. אזעקות שווא: זיהוי חריגות עלול להוביל לדיווחים שגויים ולבעיות תפעול.

2. קושי בזיהוי מתקפות מורכבות: האשכולות מתמקדים בדמיון בין תבניות, אך ייתכן שיתקשו לזהות מתקפות מורכבות המשתנות לאורך זמן או כאלה שמכילות דפוסים מורכבים המשתלבים בצורה חלקה בתוך קוד לגיטימי.

3. דרישות גבוהות לכוח מחשוב: ניתוח נתונים רבים בזמן אמת מצריך משאבים חישוביים גדולים ויכול להכביד על הביצועים.

**למידת מכונה מבוססת אשכולות בלמידה מפוקחת לזיהוי הזרקות קוד**

האלגוריתם יוצר אשכולות מנתונים מתויגים ומסווג קוד חדש לאשכולות אלה. לדוגמה, כשיש ניסיון להזרקת SQL, המערכת מזהה דפוסים התואמים לאשכול הזרקות SQL ומתריעה בזמן אמת. הכשרת המודל עם דוגמאות מתויגות מאפשרת זיהוי מהיר ומדויק של מתקפות ידועות.

**דוגמה ליישום אשכולות מפוקחים**

נניח שאנחנו בונים מערכת לזיהוי הזרקות קוד בצד הלקוח. נאמן את המודל על מאגר נתונים שבו מתויגים קטעי קוד כלגיטימיים או זדוניים. במהלך האימון, המערכת תלמד את התבניות המשותפות בקוד לגיטימי ובקוד שמכיל הזרקות קוד, ותיצור אשכולות מתאימים. כאשר קוד חדש יכנס למערכת, האלגוריתם ישייך אותו לאשכול המתאים, ואם הוא שייך לאשכול זדוני – הוא יופעל כגורם אזהרה.

**יתרונות של שימוש באשכולות בלמידה מפוקחת**

1. דיוק גבוה בזיהוי מתקפות: המודל לומד תבניות על בסיס דוגמאות מתויגות, מה שמאפשר זיהוי מדויק של מתקפות מוכרות.

2. יכולת לסווג סוגים שונים של מתקפות: האלגוריתם יכול ליצור אשכולות עבור סוגים שונים של מתקפות, כמו SQL Injection ו-XSS, ובכך לזהות כל סוג ביעילות.

3. שיפור ביצועים לאורך זמן: עם יותר נתונים, המודל משתפר בזיהוי ומסתגל למתקפות חדשות.

**חסרונות של שימוש באשכולות בלמידה מפוקחת**

1. דרישה למאגר נתונים מתויגים: אימון המודל מצריך מאגר נתונים רחב ומסודר, תהליך מורכב ולרוב ארוך.

2. יכולת מוגבלת בזיהוי מתקפות חדשות: למרות שהמודל יודע לזהות מתקפות מוכרות בצורה יעילה, הוא עלול להיכשל בזיהוי מתקפות חדשות שלא הופיעו בדוגמאות התיוג שעליהן אומן.

3. פוטנציאל לדרישות חישוב גבוהות: אימון המודלים והפעלתם במערכות גדולות מצריכים משאבים חישוביים משמעותיים.

# פרק 5 - יישומים של למידת מכונה לזיהוי הזרקות קוד

## זיהוי הזרקות קוד באפליקציות רשת

### שימוש בלמידת מכונה לזיהוי התקפות XSS באפליקציות רשת

ההגנות המסורתיות מפני התקפות XSS כוללות חומות אש (WAF) ושיטות ניתוח סטטי ודינמי, אך אלו מסתמכות על חוקים וחתימות. תוקפים יכולים לעקוף אותן בעזרת טשטוש קוד (obfuscation) או מתקפות חדשות שלא תועדו. לכן, פתרונות מסורתיים אינם תמיד יעילים בזיהוי התקפות מוסוות. פתרונות מבוססי למידת מכונה (ML) מציעים גישה אדפטיבית ומתקדמת, לומדים דפוסים מנתונים ומסוגלים לזהות התקפות חדשות או מוסוות, מבלי להסתמך על חוקים קבועים. במאמר [12] נעשה שימוש באלגוריתם SVM שבו השתמשו בשני מסווגים: לינארי ולא לינארי. הדיוק והביצועים של המסווגים תלויים בהיפר-פרמטרים של האלגוריתם אשר שולטים בתהליך הלמידה. לא ניתן לגזור אותם באמצעות אימון ולכן יש לציין אותם על ידי המשתמש באלגוריתם ה-ML. Hyperparameter Optimization הוא תהליך של מציאת קבוצה של היפר-פרמטרים אופטימליים עבור אלגוריתם ML נתון. במאמר [12] השתמשו ב-Grid search כאופטימיזציה של ההיפר-פרמטרים.

### איסוף והכנת נתונים

שלב קריטי באימון מודל למידת מכונה הוא איסוף מערך נתונים מגוון שמכיל דוגמאות חיוביות (קוד התקפה) ושליליות (קלטים לגיטימיים). יש צורך במאגר גדול של דוגמאות התקפה, לדוגמה, באמצעות כלים כמו XSSTIKE ו- XSSER לאיסוף מעל 15,000 דוגמאות של מתקפות XSS, לצד יצירת דוגמאות רגילות בעזרת כלי אוטומטי [12].

לאחר איסוף הנתונים, יש להפרידם לשני סטים: 70% עבור אימון המודל ו-30% לבדיקה. חלוקה זו מבטיחה שהמודל למד בצורה מאוזנת ויכול להתמודד עם נתונים חדשים. הרבה מודלים של למידת מכונה סובלים מהתאמת יתר (overfitting) ובמיוחד אלגוריתמי ML לא לינאריים, כדי למנוע מצב כזה בSVM נעשה שימוש בטכניקת אימות מוצלב K-Fold, טכניקה זו היא דגימה מחדש סטטיסטית שבה מודל ה-ML מאומן ונבדק על k תת-קבוצות של נתוני אימון, במקרה שלנו K נבחר להיות 10.

### חילוץ מאפיינים בעזרת TF-IDF

לאחר איסוף הנתונים, יש להמיר את המידע הגולמי למאפיינים מספריים שניתן לעבד על ידי מודלים של למידת מכונה. תהליך זה מתבצע באמצעות שיטת TF-IDF (תדירות מונח-הפוך תדירות מסמכים), המודדת את חשיבות מילה במסמך ביחס למאגר הנתונים.

באמצעות TF-IDF, מופקת מטריצה של משקלי מונחים עבור כל מסמך, כאשר כל משקל מייצג את תרומת המונח להבנת תוכן המסמך. משקלים אלו משמשים כקלט ישיר למודלים, והמרת הטקסט למספרים היא קריטית להצלחת אימון המודל [12].

### אימון המודלים

במאמר [12] נעשה שימוש במודל Support Vector Machine (SVM), אלגוריתם סיווג נפוץ בעיבוד טקסט. SVM מפריד בין דוגמאות חיוביות ושליליות על ידי יצירת "היפר-מישור" שמפריד בין הדוגמאות בקבוצת האימון.

לזיהוי התקפות XSS, המטרה היא להפריד בין קלטים זדוניים ללגיטימיים. כדי למנוע התאמת יתר (overfitting), נעשה שימוש באימות מוצלב (cross-validation) עם 10 חלקים, שבו הנתונים מחולקים ל-10 קבוצות. בכל סבב, אחת הקבוצות משמשת כסט בדיקה והיתר לאימון, והליך זה חוזר 10 פעמים, מה שמשפר את יציבות המודל.

**הערכת ביצועים**

לאחר אימון המודל, יש לבצע הערכה מדוקדקת של ביצועיו על סט הבדיקה. מדדי ביצועים חשובים כוללים דיוק (Precision), שליפה (Recall), וציון F1, שמודדים את איכות הזיהוי של התקפות. הדיוק מודד את אחוז הקלטים שזוהו כתקיפות והם אכן תקיפות, והשליפה מודדת את אחוז התקיפות שזוהו בהצלחה. ציון F1 הוא ממוצע המשוקלל של דיוק ושליפה.

במאמר, המודלים SVM לינארי ולא-לינארי השיגו תוצאות מרשימות, כאשר המודל הלא-לינארי הציג ביצועים מעט טובים יותר. ההערכה התבססה על מטריצות בלבול (confusion matrices) שהראו רמות נמוכות של זיהויי שווא והחמצות, מה שמעיד על הצלחה בזיהוי התקפות XSS לעומת קלטים לגיטימיים [12].

**יתרונות למידת מכונה בזיהוי התקפות XSS**

למידת מכונה מציעה כמה יתרונות משמעותיים על פני השיטות המסורתיות לזיהוי התקפות XSS [12]:

1. אדפטיביות: המערכות לומדות דפוסים מהנתונים, מה שמאפשר להן לזהות התקפות חדשות או מוסוות, בניגוד לשיטות מבוססות חוקים שמתקשות בזיהוי מתקפות לא תועדו.
2. חילוץ תכונות אוטומטי: מודלים של למידת מכונה יכולים לחלץ תכונות מהנתונים באופן אוטומטי, ללא צורך בהגדרה מראש של חוקים מורכבים.
3. יכולת הרחבה: המודלים מתמודדים עם כמויות נתונים גדולות, והשיפור בביצועים מתאפשר עם אספת נתונים על התקפות חדשות.
4. הפחתת זיהויי שווא: מודלים מצליחים להפחית את אחוז הזיהויים השגויים, מה שמגביר את האמינות בזיהוי התקפות.
5. שילוב שיטות שונות: ניתן לשלב מודלים עם מערכות מסורתיות כמו WAF ליצירת גישה מרובת שכבות להגנה על היישום, תוך זיהוי מתקפות שלא יזוהו על ידי חוקים רגילים.

**אתגרים וכיוונים לעתיד**

למרות היתרונות של למידת מכונה בזיהוי התקפות XSS, קיימים אתגרים משמעותיים:

1. דרישה לנתונים מגוונים: יש צורך במערך נתונים מייצג כדי למנוע הטיות במודלים.
2. משאבי חישוב: אימון מודלים מורכבים, כמו רשתות נוירונים עמוקות, דורש משאבים חישוביים גדולים [12].

בעתיד, צפוי שילוב שיטות למידת מכונה עם טכניקות מסורתיות כמו ניתוח סטטי ודינמי. גישות היברידיות אלו ישלבו את היתרונות של שתי השיטות ויספקו הגנה מקיפה יותר על יישומי רשת [12].

**זיהוי הזרקת קוד בקוד מקור**

בהרצת קוד שעדיין לא עבר הידור מתבצע תהליך בעל 3 חלקים: לקסיקלי (lexer) שממיר טקסט גולמי לטוקנים, פרסר שמארגן את הטוקנים (token) במבנה היררכי וקומפיילר/פרשן שמבצע את הקוד או ממיר אותו לקוד מכונה. המחקר [11] מציג גישה מתקדמת לזיהוי פגיעויות בקוד מקור באמצעות למידת מכונה, עם דגש על טכניקות למידה עמוקה וייצוג פיצ'רים (ביניהם לקסיקליים) לזיהוי מתקפות כמו הזרקות קוד. הוא מתמקד בשפות C ו- C++ ומנצל מודלים עמוקים של רשתות עצביות כדי לזהות דפוסי פגיעות בתוך פונקציות הקוד, תוך שימוש במאגרים גדולים של קוד פתוח. בין הפגיעויות שנחקרו היו Buffer Overflow ופגיעויות מבוססות קלט שגוי.

**נתונים ושיטות ליצירת המודל**

המחקר החל בבניית מערך נתונים רחב, הכולל מעל 12 מיליון פונקציות קוד ממאגרים כמו GitHub ו-Debian Linux. פונקציות רבות תויגו כפגיעות או לא פגיעות באמצעות שלושה כלים עיקריים: Clang , Cppcheck ו-Flawfinder, שסייעו בזיהוי תוויות לקוד עם פגיעויות ידועות כגון CWE-120 (Buffer Overflow) ו- CWE-119 ( Improper Restriction of Operations within the Bounds of a Memory Buffer). לצורך הפקת מאפיינים שימושיים מקוד המקור הגולמי של כל פונקציה, נוצר לקסיקלי מותאם אישית עבור שפות C/C++ שתוכנן לתפוס את המשמעות הרלוונטית של טוקנים קריטיים, תוך שמירה על ייצוג גנרי והקטנה של גודל אוצר המילים הכולל של הטוקנים. הפיכת הייצוג של הקוד שהלקסיקלי יוצר ייצוג אחיד ככל האפשר עבור מאגרי תוכנה שונים ומאפשרת יישום של (transfer learning) על פני כל מערך הנתונים.

שימוש בלקסיקליים מותאמים אישית לפישוט מבנה הקוד וכוללים הסרה של פריטים לא רלוונטיים, כמו שמות משתנים וקריאות למתודות ספרייה, אפשר למודל למידת המכונה ללמוד דפוסים כלליים יותר ולא להיצמד למבנים ספציפיים. לקסיקליים סטנדרטיים, המיועדים בפועל להידור קוד, לוכדים יותר מדי פרטים שעלולים לגרום להתאמת יתר (overfitting) בגישות של למידת מכונה. הלקסיקלי המתואם הצליח לצמצם את הקוד בשפות C/C++ לייצוגים המשתמשים באוצר מילים כולל של 156 טוקנים בלבד (בדרך כלל עשוי להכיל 200-300 טוקנים). בנוסף, ההטמעה הייתה בהשראת עיבוד שפה טבעית (NLP) בשימוש ב-CNN וב-RNN לזיהוי פגיעויות בקוד. הטוקנים של הקוד מומשו להטמעה בעלת k=13 ממדים (אופטימלי מניסויים קודמים), תוך הוספת רעש גאוסיאני קטן לשיפור עמידות להתאמת-יתר.

על מנת לחלץ את התכונות ב-CNN היה שימוש ב-512 מסננים בגודל m=9 וב-RNN בשתי שכבות GRU עם מצב נסתר בגודל n=256. מאחר ואורך הפונקציות ב-c/c++ משתנה מאוד, תכונות קונבולוציוניות וחוזרות עברו איגום מקסימלי לאורך רצף הטוקנים ליצירת ייצוג בגודל קבוע. אותן תכונות הוזנו למסווג Fully Connected עם Dropout ואיבוד Cross Entropy, תוך איזון משקל הפונקציות הפגיעות. האימון כלל רק פונקציות עם גודל הטוקן בין 10 ל500.

**ניתוח הפגיעויות באמצעות רשתות עצביות קונבולוציוניות (CNN)**

המחקר השתמש ברשתות עצביות קונבולוציוניות (CNN) המותאמות לניתוח טוקנים מקוד מקור, כדי לזהות דפוסים המצביעים על פגיעויות. CNN פועלות באמצעות פילטרים המבצעים קונבולוציה על רצפי הטוקנים ומחפשים דפוסי קוד פגיעים.

שיטה זו אפשרה לחוקרים לזהות פגיעויות כמו Buffer Overflow ו- NULL Pointer Dereference והדגימה כיצד CNN מסייעות באיתור דפוסים מורכבים בקוד, כמו שימוש לא תקין במשתנים או קריאות בלתי תקינות לפונקציות. זיהוי הפגיעויות התבסס על שילוב CNN עם אלגוריתם Random Forest [11].

**למידת מכונה מבוססת יערות החלטה (Random Forest)**

המערכת כללה גם יערות החלטה (Random Forest) לשיפור דיוק המודלים ולמניעת התאמת יתר (overfitting). יערות החלטה מבוססים על יצירת מספר רב של "עצים" שחוזים אם פונקציה מכילה פגיעות, והתוצאה נקבעת על פי הצבעה משותפת של העצים. שילוב Random Forest שיפר את יכולת המערכת להבחין בין פונקציות פגיעות לבין פונקציות לא פגיעות, במיוחד במקרים של מידע לא אחיד [11]. בשלב הבא, החוקרים גילו כי השימוש בתכונות נויראליות (פלט מאיגום רצפי ב-CNN וב-RNN) כקלט למסווג אנסמבל, כגון Random Forest או Extremely Randomized Trees, הניב את התוצאות הטובות ביותר. כך, השילוב של אופטימיזציית התכונות והמסווג אפשר אימון מהיר יותר ועמידות גבוהה יותר נגד התאמת-יתר.

**ניתוח באמצעות עצים סינטקטיים (AST)**

בנוסף ל- CNN ול-Random Forest, המחקר עשה שימוש בעצים סינטקטיים (AST) לניתוח הקוד. עצים סינטקטיים מציגים את מבנה הקוד באופן היררכי ומסייעים למודלים להבין את הלוגיקה של הקוד בצורה מעמיקה יותר. השימוש ב- TBCNN( Tree-Based Convolutional

Neural Networks) המותאם לעצי סינטקס שיפר את היכולת לזהות פגיעויות שנובעות מטעויות סינטקטיות, כמו כשלים באבטחה שנובעים ממבני קלט לא תקינים בקוד [11].

**תהליך יצירת תוויות לזיהוי פגיעות**

תהליך יצירת התוויות כלל שימוש בכלי ניתוח סטטיים כמו Clang, Cppcheck ו-Flawfinder, שסייעו בזיהוי פגיעויות בקוד. הממצאים מכלי הניתוח שימשו להפקת תוויות של "פגיעות" או "לא פגיעות" עבור הפונקציות, בהתאם לזיהוי פגיעויות ידועות כגון Buffer Overflow, שימוש שגוי במצביעים, ופגיעויות הקשורות לקלט [11]. באמצעות שיטה זו הצליחו החוקרים לזהות פגיעויות כמו Buffer Overflow ו-NULL Pointer Dereference. כלומר, המחקר הדגים כיצד CNN מסייעות באיתור דפוסים מורכבים בקוד, כמו שימוש לא תקין במשתנים או קריאות בלתי תקינות לפונקציות.

**תוצאות הניסויים**

המערכת נבחנה על מאגרי קוד גדולים מ- GitHub ו-Debian Linux, והשוואה לכלי ניתוח מסורתיים הראתה שמודלים כמו CNN ו- Random Forest שיפרו משמעותית את דיוק זיהוי הפגיעויות לעומת Clang. במקרים מסוימים, הדיוק הגיע לכ-90%, בעיקר בזיהוי Buffer Overflow. המחקר הראה כיצד שילוב למידה עמוקה ו- Lexers מותאמים אישית מפחית את הסיכון ל- overfitting ומאפשר למערכת ללמוד דפוסים כלליים יותר. השילוב של CNN ו- Random Forest יצר תוצאות מצוינות בזיהוי פגיעויות מורכבות באזורים שונים בקוד [11].

**יתרונות למידת מכונה לזיהוי פגיעויות בקוד**

השימוש בלמידת מכונה מציע יתרונות רבים בזיהוי פגיעויות, במיוחד כאשר מדובר בניתוח של מאגרי קוד פתוח גדולים ומורכבים. בין היתרונות המרכזיים:

1. <u>מהירות ואוטומציה</u>: מאפשרת זיהוי אוטומטי ומהיר של פגיעויות, ללא צורך בהגדרת חוקים ידניים, מה שמקל על זיהוי בעיות בפרויקטים גדולים.
2. <u>ניתוח על בסיס נתונים גדולים</u>: מספקת ניתוח מדויק יותר ככל שהמודל נחשף ליותר נתונים, בניגוד לשיטות סטטיות מסורתיות.
3. <u>יכולת להסתגל לדפוסים חדשים</u>: מתאימה עצמה לדפוסי קוד חדשים ושינויים במבני קוד, כולל פתרונות עדכניים במאגרי קוד פתוח כמו GitHub.

**אתגרים ותוצאות שגויות**

עם זאת, ישנם אתגרים בשימוש בלמידת מכונה לזיהוי פגיעויות, בעיקר ביצירת תוויות מדויקות. כלי ניתוח סטטי מסייעים, אך הם לא מספקים, ולעיתים מופיעות תוצאות חיוביות שגויות (False Positives) שמובילות לבזבוז זמן ומשאבים על תיקון בעיות שאינן קיימות. המחקר מציע לשלב ניתוחים דינמיים בנוסף לסטטיים ולהשתמש בתיקוני אבטחה קודמים כדי לשפר את איכות התוויות. גישה זו עשויה לצמצם תוצאות שגויות ולשפר את ביצועי המערכת.

**<u>יישום של למידת מכונה לזיהוי הזרקת קוד בזמן אמת</u>**

למידת מכונה מהווה אבן יסוד בזיהוי מתקפות סייבר מתקדמות כמו הזרקת קוד בזמן אמת. באמצעות מודלים של למידת מכונה, ניתן לנתח תעבורת רשת ולזהות תבניות מתקפה שלא ניתן לזהות בשיטות מסורתיות, המבוססות לרוב על חתימות קוד ידועות. כך למידת המכונה מספקת פתרונות מתקדמים למציאות הדינמית של תעבורת רשת. מערכת AI-IDS המתוארת במאמר [10] מבוססת על למידת מכונה ומשתמשת ברשתות עצביות קונבולוציוניות (CNN) וב- LSTM (זיכרון ארוך-קצר טווח) כדי לאתר מתקפות בזמן אמת.

**ארכיטקטורה ועקרונות הפעולה של המערכת AI-IDS**

מערכת AI-IDS משמשת ככלי לזיהוי בזמן אמת של מתקפות הזרקת קוד מבוססות HTTP . המערכת המוצעת מורכבת ממספר רכיבים עיקריים[10]:

1. קידוד UTF-8 לנירמול נתונים: לפני העיבוד, הנתונים מקודדים ב UTF-8-להבטחת תאימות ועיבוד מהיר, המאפשר התמודדות עם מגוון תווים וסימנים.

2. מודל CNN-LSTM: שילוב של רשת קונבולוציונית ורשת LSTM מאפשר זיהוי מאפיינים מרחביים וזמנים בתעבורת הרשת. ה- CNN מזהה תבניות קבועות במידע, בעוד שה- LSTM מנתח את הרצף והקונטקסט של הנתונים.

3. מערכת מבוססת Docker: המערכת פועלת באמצעות Docker images, מה שמבטיח גמישות והרחבה קלה עם ביצועים גבוהים.

4. כלי תיוג וניתוח: המערכת משלבת כלי תיוג המאפשרים לאנליסטים לסווג אירועים ל"חיוביים" או "שליליים", ושיפור דיוק המערכת באמצעות למידה מתמשכת.

**יתרונות השימוש במערכת מבוססת למידת מכונה** [10]

1. זיהוי מתקפות חדשות: בניגוד למערכות חתימות, המערכת מזהה מתקפות חדשות ולא מוכרות על ידי לימוד תבניות מתעבורת הרשת, כמו CNN-LSTM .

2. הפחתת אזעקות שווא: למידת מכונה מאפשרת למערכת לשפר את הזיהוי ולהפחית את שיעור האזעקות השווא, אשר נפוצות במערכות גילוי מבוססות חתימות בלבד.

3. גמישות והרחבה: הארכיטקטורה מבוססת Docker מאפשרת גמישות והתאמה לסוגי תעבורה חדשים, מה שמבטיח ביצועים טובים גם בתנאי עומס.

4. אופטימיזציה לזיהוי מתקפות מתקדמות: המערכת מסוגלת לנתח תעבורת HTTP ולגלות מתקפות מתקדמות כמו SQL Injection והתקפות מבוססות JavaScript , תוך ניצול יכולות המודל לזיהוי דפוסים במידע מוצפן או מקודד.

5. שיפור חוקי זיהוי: המערכת מספקת כלים לשיפור וחידוש חוקי זיהוי (Snort rules), מה שמאפשר התאמה מהירה להתפתחויות חדשות במתקפות, מכיוון שהמערכת עובדת במקביל למערכות NIDS מסורתיות כמו Snort.

**האתגרים בזיהוי הזרקות קוד ומתקפות בזמן אמת**

מתקפות הזרקת קוד, כגון SQL Injection, XSS ו-Malware Injection, כוללות הזרקת קוד זדוני לבקשות HTTP. התוקפים משתמשים בטכניקות כמו קידוד וטשטוש (Obfuscation) כדי לעקוף מנגנוני אבטחה מסורתיים. מערכות חתימות כמו Snort מתקשות לזהות מתקפות חדשות או מוסוות, כיוון שהן נשענות על דפוסי התקפה קבועים, מה שמחייב תחזוקה מתמדת של המודלים והנתונים. כאשר קוד זדוני לא תואם לחתימה קיימת, המערכת לא תזהה את המתקפה, מה שעלול לגרום לפרצות אבטחה. בנוסף, מערכות אלו סובלות משיעור גבוה של אזעקות שווא, במיוחד עם תעבורה לא שגרתית, דבר שמכביד על צוותי האבטחה. הפתרון לכך הוא למידת המשך (retraining) של המערכת על בסיס מידע שנאסף ונבדק על ידי אנליסטים [10].

**ניסויים ותוצאות**

במסגרת המחקר, ביצעו המחברים ניסויים על שתי מערכות נתונים ציבוריות (CSIC-2010 ו-CICIDS2017) ונתונים בזמן אמת. מודלי CNN-LSTM הראו יעילות גבוהה בזיהוי מתקפות, עם דיוק מעל 98% ושיעור זיהוי גבוה של מעל 99%. המערכת הצליחה לזהות מתקפות מורכבות ומטושטשות (obfuscated attacks) שעקפו מערכות IDS מסורתיות, כמו SQL Injection ו-Application Layer Attacks, גם כשהן מוסוות או מקודדות. היא הפחיתה משמעותית את שיעור האזעקות השגויות, והודות ללמידה מתמשכת, המערכת מתאימה את עצמה במהירות להתפתחויות חדשות במתקפות סייבר.

**יישומים מעשיים של מערכת AI-IDS בתעשייה**

יישום המערכת AI-IDS מאפשר לארגונים במגוון רחב של תעשיות לשפר באופן משמעותי את רמת ההגנה על שירותי הרשת שלהם. הנה מספר יישומים מעשיים עם דוגמאות לתעשיות שונות:

1. <u>הגנה על שירותי אינטרנט ציבוריים</u>: חברות טכנולוגיה ושירותי ענן משתמשות ב-AI-IDS כדי לזהות ניסיונות חדירה דרך API-ים ציבוריים ולהגיב במהירות, ובכך למנוע השבתת שירותים או גניבת מידע רגיש.

2. <u>שיפור מערכות הגנה קיימות</u>: השילוב של AI-IDS עם מערכות IDS מסורתיות מספק הגנה מפני מתקפות מורכבות, במיוחד במוסדות פיננסיים, שמגנים על מערכות בנקאיות מקוונות.

3. <u>הפחתת אזעקות שווא וניהול משאבים יעיל</u>: הפחתת שיעור האזעקות השווא מאפשרת לצוותי אבטחת המידע להתמקד במתקפות אמיתיות ולנהל את המשאבים ביעילות, דבר חשוב במוסדות ממשלתיים ותשתיות חיוניות.

4. <u>הגנה על תשתיות קריטיות</u>: AI-IDS מספקת שכבת הגנה מתקדמת לתשתיות קריטיות, עם יכולות זיהוי בזמן אמת ומינימום אזעקות שווא, מה שמבטיח תפעול שוטף של שירותים חיוניים כמו מים וחשמל.

# פרק 6 - עתיד זיהוי הזרקות קוד באמצעות למידת מכונה

## טרנדים חדשים בתחום למידת המכונה

למידת מכונה ממשיכה להתפתח ולשנות את תחום אבטחת המידע. בעידן הדיגיטלי, זיהוי התקפות סייבר, ובמיוחד הזרקות קוד, הוא בעדיפות גבוהה. למידת מכונה מציעה פתרונות לאתגרים אלו, עם מספר טרנדים משמעותיים שמובילים לזיהוי מתקדם של הזרקות קוד ואיומים נוספים.

**ארכיטקטורות מתקדמות של למידה עמוקה**

למידה עמוקה (Deep Learning), כתחום מרכזי בלמידת מכונה, עושה שימוש ברשתות נוירונים מתקדמות לזיהוי דפוסים מורכבים. בשנים האחרונות, רשתות אלו הפכו לכלי חיוני בזיהוי הזרקות קוד, בזכות יכולתן לנתח באופן אוטומטי דפוסים מורכבים בקוד ובנתונים, ולהפיק תובנות המסייעות באיתור איומים מתוחכמים.

**למידה עצמית (Self-supervised Learning)**

אחד הטרנדים המרכזיים בלמידה עמוקה הוא למידה עצמית, המאפשרת למודלים ללמוד מהנתונים עצמם ללא צורך בתגיות. בזיהוי הזרקות קוד, זהו יתרון גדול שכן המודלים מתמקדים בזיהוי חריגות בהתנהגות הקוד במקום להסתמך על דוגמאות מתויגות.

**הסבריות ושקיפות במערכות לזיהוי מתקפות**

ההסבריות של תוצאות מודלי למידת מכונה היא טרנד מתפתח, במיוחד באבטחת מידע, שבה קבלת החלטות קריטיות דורשת שקיפות. Explainable AI (XAI) עוסק בפיתוח כלים שמאפשרים להבין כיצד מודלים מקבלים החלטות.

**הסבריות בלמידת מכונה לאבטחת מידע**

מערכות לזיהוי הזרקות קוד, מודלים בלמידת מכונה יכולים להיות מורכבים, מה שמקשה להבין מדוע קטע קוד מסווג כחשוד. הסבריות מסייעת למפתחים ולצוותי אבטחה להבין את ההחלטות, לחזק את האמון במודלים, ולשפר את מניעת הפגיעויות. כלים כמו SHAP ו- LIME מציגים את התכונות החשובות שהשפיעו על החלטת המודל, ומקלים על ניתוח התוצאות.

**מודלים אינטראקטיביים**

הדרישה לשקיפות מובילה לפיתוח כלים לאינטראקציה עם מודלי למידת מכונה. בעתיד, מפתחים יוכלו לשאול את המודל "למה הקוד הזה זוהה כפגיע?" ולקבל תשובות שיסייעו בשיפור הקוד ובהפחתת הסיכון להזרקות קוד.

**למידה מבוזרת (Federated Learning)**

למידה מבוזרת מאפשרת לאמן מודלים על נתונים מפוזרים בין מכשירים או ארגונים, בלי לשתף את הנתונים עצמם, וכך לשמור על פרטיות. זהו פתרון חשוב לארגונים המעוניינים לזהות מתקפות כמו הזרקות קוד על ידי ניתוח נתונים משותף. כל ארגון מאמן מודלים מקומיים ומשתף את המודלים (ולא את הנתונים), כדי ליצור מודל גלובלי שמזהה דפוסי תקיפה רחבים יותר.

**הגנה בזמן אמת באמצעות למידת חיזוק (Reinforcement Learning) ומערכות דינמיות**

למידת חיזוק היא טכניקה שבה אלגוריתם לומד מניסוי וטעיה, תוך התאמה על בסיס תגמולים או עונשים. בתחום אבטחת המידע, היא מאפשרת לפתח מערכות הגנה דינמיות שמתאימות את עצמן בזמן אמת להתקפות חדשות, ללא תלות בנתוני אימון סטטיים. המערכות יכולות לזהות ולחסום תקיפות בזמן אמת, מה שמספק הגנה אוטונומית מפני איומים משתנים, במיוחד נגד הזרקות קוד.

## יישומים מתקדמים של למידת מכונה לזיהוי הזרקות קוד

יישומים מתקדמים של למידת מכונה לזיהוי הזרקות קוד מדגישים את השילוב בין טכנולוגיות חדשניות לצורך בהגנה על מערכות מחשוב מפני התקפות סייבר מתוחכמות. בחלק זה נפרט על יישומים מתקדמים של למידת מכונה בזיהוי הזרקות קוד, עם דגש על הטכניקות החדשניות ביותר.

### שילוב למידה מכונה עם ניתוח גרפים

שילוב למידת מכונה עם ניתוח גרפים הוא יישום מתקדם בזיהוי הזרקות קוד, באמצעות רשתות נוירונים על גרפים (**Graph Neural Networks - GNN**). גרפים מייצגים את היחסים בין רכיבי הקוד, כמו פונקציות ומבני תלות, ומאפשרים ניתוח יעיל של הקוד. GNN מנתחות עצי תחביר מופשטים ( Abstract Syntax Trees-AST ) וגרפי זרימת שליטה ( CFG - Control Flow Graphs) כדי לזהות דפוסים בעייתיים שלא נראים בניתוח שטחי.

באמצעות GNN, ניתן לזהות הזרקות קוד על ידי זיהוי סטיות מזרימת התוכנה הרגילה או ניתוח התלות בין משתנים. לדוגמה, ניתן לבדוק אם קלט ממשתמש עובר סינון (Sanitization) ואימות (Validation) לפני שמירתו בבסיס הנתונים. אם הקלט זורם ישירות ללא ניקוי, המודל יכול לסמן את הקוד כחשוד ולשלוח התראה.

### שילוב היברידי של ניתוח סטטי ודינמי מתקדמים

שילוב ניתוח סטטי (Static Analysis) ודינמי (Dynamic Analysis) הוא גישה מרכזית באבטחת הקוד, המשלבת טכניקות לזיהוי פגיעויות. ניתוח סטטי מזהה בעיות מבניות בקוד מבלי להריץ אותו, כמו זריקות SQL, ומאפשר גילוי בעיות בשלב הפיתוח באמצעות טכניקות כמו Random Forest ו-Support Vector Machines (SVM).

במקביל, ניתוח דינמי עוקב אחרי פעולות התוכנה בזמן אמת ומזהה חריגות כמו קלטים לא תקינים. Deep Reinforcement Learning משפר את היכולות בתחום זה על ידי למידה מניסיונות קודמים.

השילוב של שתי הגישות באמצעות למידת מכונה מציע גישה מקיפה, המפחיתה התרעות שגויות (False Positives) ומגלה בעיות שיכולות להתגלות רק בזמן הריצה, ובכך משפרת את ההגנה על התוכנה.

### שימוש במודלים מוכללים מראש (Pretrained Models)

גישה למאגרי קוד פתוח גדולים, כמו GitHub,מאפשרת הכשרת מודלים של למידת מכונה על כמות עצומה של דוגמאות קוד. מודלים אלו, המכונים "מודלים מוכללים מראש", מאפשרים זיהוי פגיעויות בצורה יעילה יותר לעומת מודלים שמתחילים את הלמידה מאפס.

### מודלים טרנספורמרים ומודלים מוכללים מראש

רשתות נוירונים עמוקות, במיוחד טרנספורמרים (**Transformers**) כמו GPT ו-BERT, מותאמות לניתוח שפות תכנות וזיהוי פגיעויות בקוד. מודלים אלו מזהים דפוסים סינתקטיים וסמנטיים, גם בהתקפות מוסוות.

מודלים כמו CodeBERT ו- GPT-3 הותאמו לניתוח קוד ומספקים בסיס לזיהוי אוטומטי של פגיעויות, כולל הזרקות קוד. הם מתמחים בזיהוי חולשות, כמו קלט לא מבוקר שיכול להוביל להזרקה זדונית. יתרונם הוא ביכולת להתאים לשפות תכנות שונות ולסביבות פיתוח מגוונות, מה שמאפשר יישום במגוון מערכות.

### יישום במערכות שונות

מודלים מוכללים מראש מאפשרים לארגונים לאמן מערכות לזיהוי הזרקות קוד במהירות ובעלות נמוכה. הם מתאימים במיוחד לפרויקטים בקוד פתוח, הודות לגישה למאגרי קוד גדולים. ניתן לטעון מודלים אלו ביישומים כמו IDEs, שרתים ומערכות מבוססות ענן.

**אבטחה בקצה הרשת (Edge Computing)**

בעידן שבו מכשירים חכמים ו- IoT נפוצים, זיהוי הזרקות קוד במכשירים בקצה הרשת הוא קריטי. Edge Computing מאפשר העברת כוח עיבוד למכשירים עצמם, כך שלמידת מכונה יכולה לזהות מתקפות, כולל הזרקות קוד, ישירות במכשירים, ללא צורך בהמתנה למרכזי נתונים רחוקים.

**מערכות זיהוי בקצה**

מכשירי IoT, כמו חיישנים תעשייתיים ומצלמות חכמות, מוגבלים בעיבוד ואנרגיה. לכן, פיתוח מודלים קלים ויעילים לזיהוי פגיעויות הוא אתגר חשוב. מודלים כמו TinyML מאפשרים להריץ אלגוריתמים של למידת מכונה ישירות על מכשירים בקצה הרשת, תוך חיסכון במשאבים, ובכך לזהות ולהגיב לזריקות קוד בזמן אמת.

**אבטחת IoT**

מכשירי IoT חשופים להתקפות זדוניות בגלל חוסר אבטחה. זיהוי מתקפות באמצעות למידת מכונה בקצה הרשת הוא פתרון חיוני ליישומים כמו בתים חכמים ותחבורה חכמה. הטמעת למידת מכונה במכשירים מאפשרת לנתח התנהגות חשודה ולמנוע התקפות כמו הזרקות קוד לפני נזק.

**<u>אתגרים עתידיים בתחום זיהוי הזרקות קוד באמצעות למידת מכונה</u>**

תחום למידת המכונה מתפתח במהירות, יחד עם הכלים והטכניקות לזיהוי מתקפות סייבר כמו הזרקות קוד. עם זאת, התפתחויות אלו מביאות גם אתגרים חדשים שדורשים פתרונות מתקדמים. במאמר זה נתמקד בחמישה אתגרים מרכזיים שמשפיעים על זיהוי הזרקות קוד בעזרת למידת מכונה: עמידות בפני התקפות אדברסריות, הסבריות ושקיפות, אינטגרציה במערכות פיתוח וזיהוי מתקפות Zero-day.

**1. שיפור עמידות בפני התקפות אדברסריות (Adversarial Attacks)**

מודלי למידת מכונה רגישים למתקפות אדברסריות (Adversarial Attacks) , בהן תוקפים מבצעים שינויים מזעריים בקלט כדי להטעות את המודל ולגרום לו לקבל החלטות שגויות. במתקפות אלה, שינויים קטנים, לעיתים בלתי נראים לעין האנושית, יכולים להוביל לתוצאות שגויות מצד המערכת. לדוגמה, תוקף יכול לשנות שם של משתנה או להוסיף פרגמנט קטן לקוד שאינו משפיע על הפעולה אך מצליח לבלבל את המודל ולגרום לו לחשוב שהקוד תקין.

<u>דרכים להתמודד עם התקפות אדברסריות</u>

אחת הדרכים להתמודד עם התקפות אדברסריות היא אימון אדברסרי (Adversarial Training), שבו המודל לומד לזהות ולהתגונן מקלטים מתוחכמים שמנסים להטעות אותו. תהליך זה דורש משאבים רבים ומעלה את מורכבות המודל.

בנוסף, פיתוחים בתחום דיטקטור אנטי-אדברסרי (Anti-Adversarial Detectors) מזהים ניסיונות התקפה באופן עצמאי, על ידי סריקת קלטים וזיהוי חריגות לפני עיבוד הקלטים על ידי המודל הראשי.

כמו כן, אנליזה פורמלית (Formal Verification) בלמידת מכונה מתפתחת ומאפשרת לאמת את יציבות המודל באמצעות כלים מתמטיים פורמליים, המבטיחים עמידות בפני שינויים זעירים בקלט.

**2. הסבריות ושקיפות (Explainability and Transparency)**

עם עליית מורכבות מודלי למידת המכונה, עולה הצורך בהסבריות (Explainability) ושקיפות בתהליך קבלת ההחלטות. במודלים מסורתיים, כמו עץ החלטות, ניתן לעקוב בקלות אחרי החוקים שהובילו להחלטה. לעומת זאת, במודלים מתקדמים כמו רשתות נוירונים עמוקות וטרנספורמרים, הבנת תהליך קבלת ההחלטה הופכת למורכבת הרבה יותר.

חשיבות השקיפות בלמידת מכונה

הסבריות היא קריטית באבטחת סייבר, שכן מנהלי אבטחה ומתכנתים חייבים להבין מדוע מודל למידת מכונה סיווג קוד כזדוני או תקין. חוסר בהבנה עשוי להוביל לשגיאות חמורות. אחת הגישות היא פיתוח מודלים פרשניים (Interpretable Models), שמשתמשים בכלים כמו LIME (Local Interpretable Model-agnostic Explanations) ו- SHAP ( Shapley Additive Explanations) כדי להסביר את ההחלטות ולהנגיש את תהליך קבלת ההחלטות. אתגר נוסף הוא יצירת ממשקי משתמש שמספקים הסברים ויזואליים בזמן אמת, מה שיגביר את האמון במודלים ככל שהטכנולוגיה תתקדם.

**3. אינטגרציה של למידת מכונה במערכות פיתוח (IDEs)**

מטרת זיהוי הזרקות קוד היא להטמיע טכנולוגיות ישירות בסביבות הפיתוח (IDEs) כמו Visual Studio ו-IntelliJ. מתכנתים משתמשים בכלים לניתוח קוד, והשדרוג הבא הוא להוסיף כלי למידת מכונה שיזהו בזמן אמת קוד פגיע ויתריעו על פוטנציאל לזריקות קוד במהלך הכתיבה.

האתגרים באינטגרציה של למידת מכונה בסביבות פיתוח

האתגר המרכזי הוא לפתח כלים שיתממשקו עם סביבות פיתוח מבלי להפריע למתכנתים או להשפיע על ביצועים. יש ליצור פתרונות שיפעלו ברקע ויסרקו את הקוד בזמן אמת, תוך התרעה רק במקרה של סיכון ברור. התראות שגויות עלולות לגרום למפתחים להתעלם מהן ולהפחית את היעילות.

לכך, יש לפתח מודלים קלים ומהירים לניתוח דינמי של קוד בזמן ריצה, וכלים מודולריים שיתממשקו עם מערכות קיימות מבלי לדרוש שינויים משמעותיים בקוד.

**4. זיהוי תקיפות אפס-יום (Zero-Day Attacks)**

תקיפות אפס-יום הן מתקפות חדשות שעדיין לא קיימים עבורן פתרונות מוכנים או דפוסי זיהוי ידועים. למידת מכונה מבוססת על נתוני אימון, ולכן אחד האתגרים הגדולים ביותר הוא היכולת לזהות תקיפות חדשות לגמרי, שלא היו קיימות קודם לכן. מודלים המסתמכים על דפוסים ישנים עלולים שלא לזהות תקיפות מסוג זה, ובכך להוות נקודת תורפה משמעותית.

פתרונות לזיהוי תקיפות אפס-יום

אחד הפתרונות המתקדמים לזיהוי תקיפות חדשות הוא למידה בלתי מונחת ( Unsupervised Learning), המאפשרת למודלים לזהות אנומליות מבלי להסתמך על דפוסים ידועים. המודל לומד את ההתנהגות הרגילה ומזהה חריגות שעשויות להעיד על תקיפה חדשה. טכניקות כמו-Meta Learning יכולות לסייע למודלים להגיב במהירות למצבים חדשים, תוך שימוש בידע קודם. בנוסף, זיהוי דפוסים דינמיים בזמן אמת באמצעות מערכות ניתוח דינמיות מסייע בזיהוי התקפות שאינן ניתנות לזיהוי בשלב הניתוח הסטטי.

**5. הבטחת פרטיות ובטיחות בלמידה מבוזרת (Federated Learning)**

למידה מבוזרת (Federated Learning) היא גישה חדשה בלמידת מכונה, המאפשרת לאמן מודלים על נתונים המפוזרים בין מספר מכשירים או אתרים מבלי לשתף את המידע עצמו. שיטה זו מציעה יתרונות משמעותיים בתחום אבטחת מידע ופרטיות, במיוחד כשמדובר ביישומים רגישים כמו בריאות, בנקאות ואבטחת מידע.

עקרונות הלמידה המבוזרת

בלמידה מבוזרת, המודל נשאר על מכשירים מקומיים, והנתונים לא עוזבים אותם. המכשירים מבצעים עדכונים על המודל המקומי שלהם, והעדכונים בלבד נשלחים לשרת המרכזי, המעדכן את המודל הגלובלי מבלי לחשוף נתונים. טכניקות לשמירה על פרטיות:

1. Diffential Privacy: מבטיחה שהמידע הנחשף לא יחשוף פרטים אישיים.
2. Aggregation Protocols: איגוד נתונים כדי לשמור על חסיון המידע.

3. Secure Multi-Party Computation (SMPC): מאפשר חישובים משותפים ללא חשיפת נתונים אישיים.

יתרונות הלמידה המבוזרת

יתרונותיה הם: שיפור פרטיות הנתונים, הפחתת עלויות הנתונים, אימון מודלים באיכות גבוהה, ועמידה ברגולציות פרטיות.

אתגרים בלמידה מבוזרת

על אף היתרונות הרבים, למידה מבוזרת נתקלת גם באתגרים: קושי בסנכרון עדכונים בין מכשירים, איכות הנתונים המקומיים עשויה להיות לא מדויקת, חשיפה למתקפות אדברסריות, ניהול משאבים במכשירים מוגבלים, ומורכבות פיתוח מערכת למידה מבוזרת.

# פרק 7 - סיכום ומסקנות

בפרק זה נסכם את הפרקים המרכזיים שדנו בהם בעבודה אשר התמקדה בטכניקות למידת מכונה לזיהוי הזרקת קוד, לאחר מכן נבצע הערכה של היתרונות והחסרונות של הגישות השונות והתפיסות לעתיד בתחום.

לאורך העבודה דנו בסוגי מתקפות הזרקת קוד כגון SQL Injection ו-XSS, הנחשבות למתקפות נפוצות ומסוכנות בעולם אבטחת המידע. מתקפות אלו מנצלות חולשות בקלטי המשתמש כדי להחדיר קוד זדוני העלול לפגוע באבטחת המידע במערכת, כולל גניבת מידע רגיש, שינוי והשחתת נתונים, ואף השתלטות על השרת. הסברנו כי השיטות המסורתיות לזיהוי מתקפות אלה, כגון סינון קלט, קידוד וניתוח דינמי, אמנם אפקטיביות במידה מסוימת אך הן מתקשות בזיהוי מתקפות מוסוות או בלתי ידועות.

על מנת להתמודד עם אתגרים אלו, בחנו בעבודה את השימוש בלמידת מכונה כפתרון חדשני ומתקדם בתחום אבטחת המידע. דנו בשיטות שונות של למידת מכונה, כולל עצי החלטה, מכונות וקטורים תומכים (SVM), רשתות נוירונים מלאכותיות (ANN), ורשתות קונבולוציוניות (CNN). לכל שיטה הצגנו את ההבדל בין למידה מפוקחת ללא מפוקחת וכן היתרונות והחסרונות שלהן. למשל, עצי החלטה הרבה יותר קלים להבנה ולפרשנות אך עלולים לסבול מהתאמת יתר, בעוד ש-SVM מספקים הפרדה מדויקת אך הם דורשים משאבי חישוב רבים. רשתות נוירונים עמוקות מאפשרות זיהוי של דפוסים מורכבים וקשרים נסתרים בנתונים, אך הן דורשות כמויות גדולות של נתונים מסומנים לאימון. CNN מספקות ביצועים טובים בזיהוי מבוסס רצפים וניתוח מבני קלט מורכבים.

בתפיסה לעתיד, הרצון הוא לשלב טכניקות למידת מכונה בצורה עמוקה יותר במערכות אבטחה קיימות, תוך פיתוח גישות היברידיות המשלבות מספר שיטות יחד. כמו כן, יש צורך בהשמת דגש על פיתוח מודלים המספקים שקיפות והסבריות גבוהה, כדי להקל על הבנת תהליכי קבלת ההחלטות. נושאים נוספים לעתיד יכולים לכלול שיפור בניהול נתוני האימון, יצירת טכניקות מתקדמות למניעת התאמת יתר ,התמודדות עם מתקפות חדשות ומתקפות המתפתחות במהירות.

לסיכום, למידת מכונה מציעה פתרונות מתקדמים וחזקים לאבטחת מידע, אך דורשת המשך פיתוח והתאמות כדי לעמוד באתגרים המורכבים שמציבה הזירה המודרנית של אבטחת הסייבר.

# פרק 8 - ביבליוגרפיה


[1] Abaimov, S., & Bianchi, G. (2021). A survey on the application of deep learning for code injection detection. *Array*.

[2] Shalev-Shwartz, S., & Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge University Press

[3] Abaimov, S., & Bianchi, G. (2019). CODDLE: Code-injection detection with deep learning. *IEEE Access*, *7*, 128617–128627.

[4] Singhal, S., & Yadav, P. (2022). Machine learning techniques for intrusion detection system: A survey. In S. Sharma, S.-L. Peng, J. Agrawal, R. K. Shukla, & D. N. Le (Eds.), *Data, engineering and applications* (Vol. 907, pp. 1-14). Springer.

[5] Kuppa, K., Dayal, A., Gupta, S., Dua, A., Chaudhary, P., & Rathore, S. (2022). ConvXSS: A deep learning-based smart ICT framework against code injection attacks for HTML5 web applications in sustainable smart city infrastructure. *Sustainable Cities and Society*, *80*, Article 103601.

[6] O'Shea, K., & Nash, R. (2015). An Introduction to Convolutional Neural Networks. *arXiv preprint arXiv: 1511.08458*.

[7] Haykin S. (1998). Neural Networks: A Comprehensive Foundation. Prentice Hall New Jersey, NY.

[8] Kakisim, A. G., Kaur, P., & Jain, R. (2023). A deep learning approach based on multi-view consensus for SQL injection detection. *International Journal of Information Security*, *23*(2), 1541-1556.

[9] Liu, H., & Lang, B. (2019). Machine learning and deep learning methods for intrusion detection systems: A survey. *Applied Sciences*, *9*(20), 4396.

[10] Kim, H., Kim, Y., & Park, H. (2021). AI-IDS: Application of deep learning to real-time web intrusion detection. *IEEE Access*, *10*, 10935-10951.

[11] Russell, R., Kim, L., Hamilton, L., Lazovich, T., Harer, J., Ozdemir, O., Ellingwood, P., & McConley, M. (2018). Automated vulnerability detection in source code using deep representation learning. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)* (pp. 757–762). IEEE.

[12] Gogoi, B., Ahmed, T., & Saikia, H. K. (2021). Detection of XSS attacks in web applications: A machine learning approach. *International Journal of Innovative Research in Computer Science and Technology*, *9*(1), 1–10.

[13] Stallings, W., & Brown, L. (2018). *Computer security: Principles and practice* (4th ed.). Pearson.

[14] Wikipedia, The free encyclopedia. http://en.wikipedia.org/wiki/Main_Page