**PDF malware detection - preliminary research**

# הקדמה:

# מבנה כללי של PDF:

מבנה PDF מחולק ל2 חלקים עיקריים:

1. **General structure** - header, body, Cross-Reference (X-Ref) Table, Trailer
2. **Objects-** כפי שצוין קודם לכן, אובייקטים מזוהים בדרך כלל על ידי מספר, והם יותר  
   המכונה באופן רשמי אובייקטים עקיפים. עם זאת, כל אלמנט בתוך הגוף נחשב בדרך כלל  
   כאובייקט, גם אם מספר אינו מזהה אותו. כאשר חפץ אינו מזוהה במספר  
   (כלומר, כאשר הוא חלק מאובייקטים אחרים), זה נקרא ישיר.  
   אובייקטים עקיפים מורכבים בדרך כלל משילוב של אובייקטים ישירים

.

נסביר על כל אחד מהחלקים של **General structure** :

* **Header** הצגת שורת טקסט אחת המכילה מידע על גרסת קובץ הPDF
* **Body-** - רצף של אובייקטים המגדירים את הפעולות שמבצע הקובץ  
  יכול להכיל גם נתונים מוטעים דחוסים או לא דחוסים (לדוג' טקסט, תמונות, קוד)לכל אובייקט יש מספר סימוכין ייחודי
* **טבלת הפניות מוצלבות (X-Ref).** רשימה של קיזוזים המציינים את המיקום של כל אובייקט ב

הקובץ. רשימה כזו נותנת לקוראי PDF אינדיקציות מדויקות היכן להתחיל לנתח כל אחד מהם

לְהִתְנַגֵד. טבלת ההפניה ההצלבה מוצגת על ידי הסמן xref, ואחריו רצף

של מספרים, שהאחרון שלהם מציין את המספר הכולל של אובייקטים בקובץ. כל שורה בטבלה

מתאים לאובייקט מסוים, אבל רק קווים המסתיימים ב-n קשורים לאובייקטים שכן

מאוחסן באופן קונקרטי בקובץ. ראוי לציין שכל קוראי ה-PDF מנתחים רק את האובייקטים

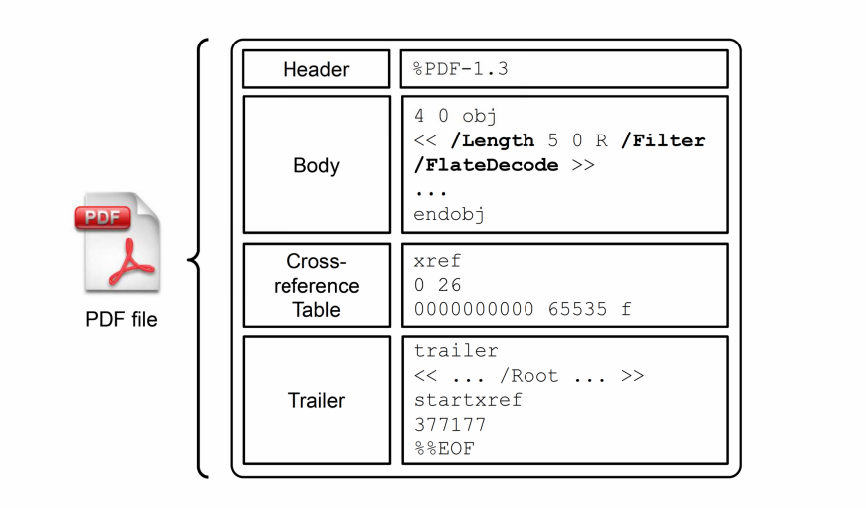
שאליהם מתייחסים טבלת X-Ref. לכן, אפשר למצוא חפצים שמאוחסנים

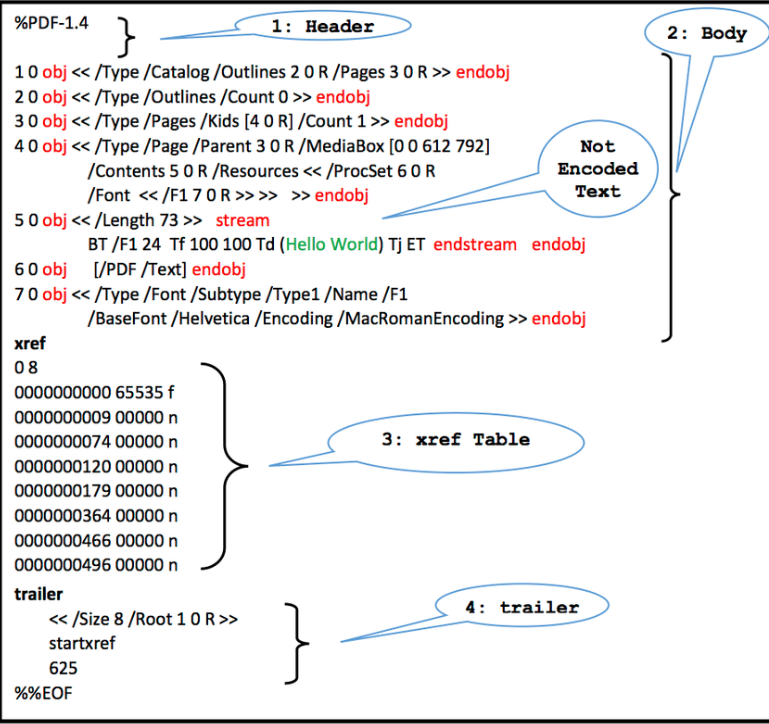
בקובץ, אך אין לכך התייחסות בטבלה.

* **טריילר**- אובייקט מיוחד המתאר כמה אלמנטים בסיסיים של הקובץ, כמו האובייקט הראשון

של הגרף (כלומר, היכן קוראי ה-PDF צריכים להתחיל לנתח את פרטי הקובץ). יתר על כך,

הוא מכיל הפניות למטא נתונים של הקובץ, המאוחסנים בדרך כלל באובייקט בודד אחד.

מילת המפתח טריילר תמיד מציגה את אובייקט הטריילר.

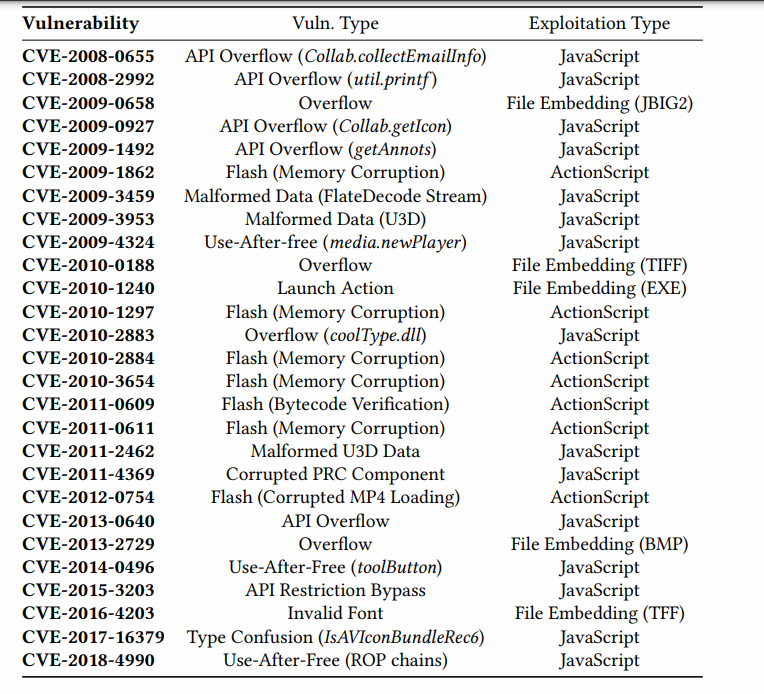


## קבצי PDF מנותחים באופן הבא:

ראשית אובייקט הטריילר מנותח ראשון.  
ולאחר מכן כל אובייקט בגוף נחקר על ידי המיקום שהוא צריך להיות לפי קרוס רפרנס  
כל קובץ פי די אף מסתיים בEOF

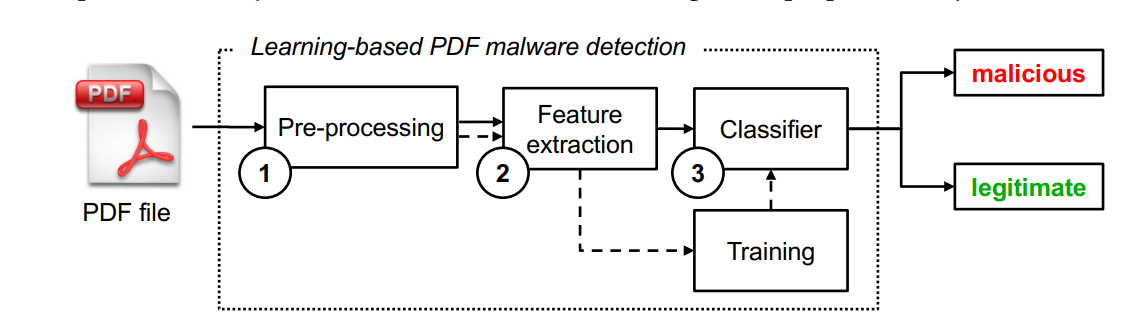
נדבר על 3 מתקפות עיקריות בPDF:

* **JavaScript** מתקפות בהן בתוך אובייקט או כמה אובייקטים מוכל סקריפט שיפעיל קטע קוד זדוני.
* **ActionScript** התקפות בהן יש קוד זדוני שרץ ברקע ברגע שקובץ הPDF נפתח
* **File Embedding** טכניקת ניצול המשתמשת בקבצים חיצוניים כגון exe על מנת לנתח את הקובץ ולהטמיע בו ניצול (למשל בתוך תמונה וכו')



# **למידת מכונה בשביל גילוי נוזקות בקבצי PDF**

למידת המכונה כאן עובדת באופן דומה של למידת מכונה כללית.



יש לנו 3 שלבים בלמידת מכונה בכדי לסווג בסוף האם הקובץ זדוני או לגיטימי.

1. Pre-processing – עיבוד מוקדם ניתוח קבצי PDF על ידי בידוד נתונים חיוניים למשל : ג'אבה סקריפט, מילות מפתח, ActionScript וכו'.
2. Feature extraction – חילוץ פיצ'רים, רכיב זה פועל על המידע שחולץ לנו ע"י תהליך מס' 1, וליצור ממידע זה ווקטור מספרים. ווקטור זה יכול לייצג מילות מפתח, שימוש ב API וכו'.
3. Classifier – שלב הסיווג, לא ניתן את כל המידע רק לאימון אלה גם לבדיקות, וגם בשלב זה נבדוק שאכן אנו לא בover fitting , ומשם נסווג האם זדוני או לגיטימי.

כעת נסביר כיצד ניתן לאפיין כל מרכיב ומהם החוזקות והחולשות שלו.

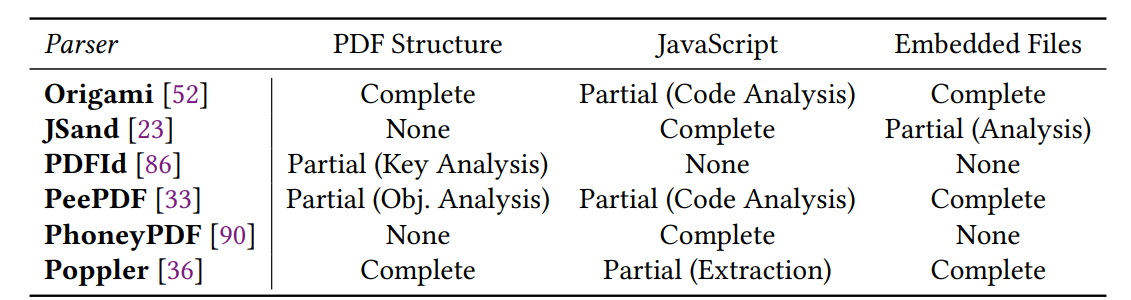
Pre-processing – תהליך זה חיוני מאוד לבחירת הנתונים בכדי שנוכל לזהות.   
ישנם 2 סוגי תהליכי עיבוד:

1. סטטי – לא נריץ את הקובץ, אלא ננסה לנתח את התוכן שלו, הבנה כמות מספר האובייקטים, קבצי JavaScript אשר בקובץ כמה מקודדים. למשל ע"י pdf-parser, peePdf ניתוח זה משמעותית לוקח פחות זמן ופחות יקר, אך חסרונו הוא עירפול מידע, כלומר יכול להיות שבסופו של דבר לא נדע בוודאות האם הקובץ זדוני או לא.
2. דינאמי – הרצה של הקובץ בתוך סאנד-בוקס וכדומה. החיסרון הוא שזה פעולה יקרה וצורכת הרבה זמן, אך בסופו של דבר תביא לתוצאות חד משמעיות.

כאשר אנו מדברים על כלים ל pre-processing ישנם 2 קטגוריות עיקריות:

1. על ידי גורמי צד שלישי – יתרון: אתה לא צריך להמציא בעצמך את הכלי לניתוח וכבר מקבל פונקציות מובנות. חיסרון: בגלל שזה גינארי יכול להיות שישנם הרבה פונקציות שאתה בכלל לא צריך וסתם מעמיס על המערכת וכמו כן יכולים להיות באגים ובגלל שזה צד שלישי לא נדע עליהם.
2. ניתוח מותאם אישית – ניתוח שאתה עושה בעצמך, יתרון: מותאם אישית לבעיה שלך. חיסרון: לבנות את הכל מאפס וכמו לכל מתכנת גם לך יכול להיות באגים וגם לוקח הרבה זמן לבנות.

טבלה שמראה אלו חברות נותנות לנו שירות של צד שלישי(חלק מהם זה קוד פתוח לא הכל בתשלום). ואפשר לראות גם במה הם מתמחות יותר ובמה פחות עוזרות לגלות.



ישנם 3 כיווני חקירה עיקריים כמו שאפשר לראות בטבלה כמו שאנו יודעים אלו הפרטים בהם בד"כ שותלים את הנוזקה:

* PDF Stracture – חלק זה מתייחס לכל המרכיבים של מבנה קובץ PDF ואינם קשורים

ל Embedded Files ,למשל לניתוח של כמה אובייקטים יש לנו כמה מהם מקודדים וכו'.

* JavaScript – מתייחס לקוד הג'אבה סקריפט המוטבע בתוך הקובץ, כלומר כאן אנו מנתחים את הקוד האם הוא זדוני או לא, וכאן כמו מקודם אפשר לעשות זאת בשני אפשריות ניתוח סטטי או דינאמי שהסברנו למעלה.
* Embedded Files – כאן רעיון הניתוח הוא לחלץ קבצים אשר ניסו להחדיר למשל קיבצי exe שניסו להחדיר לקובץ אנו רוצים לנתח אותו. חילוץ של קבצים מוטבעים.

כמו שאפשר להבין מטבלה אין לנו שירות אשר יכול לטפל ב3 כיווני החקירה. לכן כדי להבין מה כל נותן שירות מספק לנו כדי שנדע האם אנו זקוקים לו או לא.

* **Origami**- מנתח זה, שנכתב כולו ב Ruby , מאפשר למשתמשים לנווט באובייקט

מבנה של קבצי PDF, ליצירת קבצים זדוניים על ידי הזרקת קוד או אובייקטים אחרים, כדי לשחרר דחיסה ומפענח זרמים וכו'. יתר על כן, הוא מטמיע מידע פופולרי לזיהוי

פגיעויות מבוססות JavaScript API.

* **JSand** - מנתח זה היה חלק ממנוע Wepawet לביצוע ניתוח דינמי של קבצי PDF. זה יכול להפעיל את קוד ה-JavaScript המוטבע כדי לחלץ קריאות API ולבטל את הערפול קוד. יתר על כן, זה יכול לבדוק קובצי הפעלה משובצים(Embed) כדי לחשוף את נוכחותם של נוספים

התקפות. לרוע המזל, שירות Wepawet אינו זמין כעת; ולכן זה לא אפשרי

כדי לבדוק את JSand יותר.

* **PDFId** - מנתח זה פותח כדי לחלץ את אובייקטי שם ה-PDF הוא אינו מבצע ניתוח נוסף על קוד או קבצים מוטבעים.
* **PeePDF** - מנתח זה, שנכתב כולו ב-Python, יכול לבצע ניתוח מלא של מבנה קובץ ה-PDF (ללא יכולת להחדיר אובייקטים). זה מאפשר להזריק ולחלץ קבצים משובצים(Embed), והוא מספק ניתוח סטטי בסיסי של קוד JavaScript.
* **PhoneyPDF** - מנתח זה, שנכתב כולו ב-Python, מבצע ניתוח דינמי של קוד JavaScript מוטבע באמצעות מכשור. ליתר דיוק, זה מחקה את ביצוע של קוד JavaScript המוטבע בקובץ PDF, על מנת לחלץ קריאות API זה קשורים אך ורק לביצוע PDF. מנתח זה אינו מבצע ניתוח מבני כלשהו או הטמעת חילוץ קבצים.
* **Popler** - היא ספריית C++ המשמשת תוכנות פופולריות בקוד פתוח כגון X-Pdf כדי להציג את תוכן הקובץ. מסיבה זו, תכונות הספרייה הושלמו תמיכה בניתוח וניהול מבנה PDF, כמו גם חילוץ קוד JavaScript והזרקת קבצים משובצים(Embed).

תמונה שמכילה שולחן

התיאור נוצר באופן אוטומטיFeature extraction  
 בשלב זה נחלץ מתוך הניתוח של השלב הקודם את התכונות הרלוונטיות לקובץ שלנו וניצור ממנו וקטור שיכנס ללמידת המכונה.  
התכונות בוקטור מחולקות ל3 קטגוריות:  
- מבניות הקובץ: תכונות הקשורות למבנה קובץ הpdf לרוב נוגעות במילות מפתח ספציפיות בקובץ או אובייקטים\זרמים עקיפים.  
- java script : תכונות הקשורות למבנה קוד ומתאפיינות באופרטורים ספציפיים וכמותם  
- raw bytes : נוגעות בתכונות ספציפיות הנוגעות לרצפים של בתים שנלקחו כn-grams

כפי שניתן לראות בטבלה לעיל, רוב הפתרונות בנושא התקפות על Pdf מיישמות מתודות חילוץ הכוללות רק סוג אחד של תכונה.   
הדבר נעשה בעקבות הרצון להיות יותר אפקטיביים בנושא ספציפי אחד ולא ללכת לאיבוד בחיפוש במתודה שתכלול באופן כולל את כל 3 הסוגים.

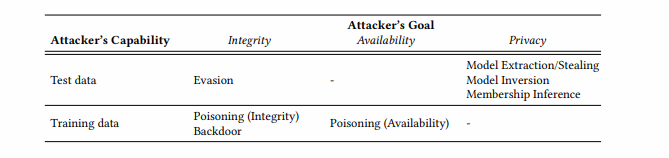
Learning and Classification – את הוקטור תכונות שמגיע מתוך חילוץ התכונות בשלב הקודם נכניס למכונה הלומדת וממנה נקבל את הסיווג עבור הקובץ

# Related works:

טכנולוגיות הזיהוי מתחלקות ל3 : סטטי, דינאמי, ומשולב (סטטי ודינאמי יחד).  
**ניתוח סטטי**  
בשנת 2008 לראשונה זוהו קבצים זדוניים באמצעות שיטת Markov n-grams שהייתה מודל סטטי באמצעות למידת מכונה. הרעיון הכללי שלו היה לאתר קודים זדוניים ולהשוותם אל מול הטכנולוגיה הקודמת שהייתה קיימת.   
זמן קצר לאחר מכן בשנת 2009 הוצגה שיטת זיהוי המבוססת על עצי החלטה ובה מיצוי הפיצ'רים של הקובץ היה גם לרמת הבתים והאלגוריתם המשומש שם היה אלגוריתם סטנדרטי של כריית נתונים שניתח כל בלוק בקובץ, הוא התבסס על חוסר חתימה ויכולת לזהות תוכנות זדוניות של zero-day בצורה מדויקת.  
 לאחר מכן כבר בשנת 2011 פותחה PJscan טכנולוגיית זיהוי למסמכי PDF זדוניים, שהייתה שיטת ניתוח סטטי מבוססת javaScript ויכלה להשיג כבר כ85 אחוזי דיוק בזיהוי. בהשוואה לדגמים דינאמיים טכנולוגיה זו נפלה לא מעט פעמים בזיהוי קבצים תקינים כזדוניים.  
במהלך 2012 הוצעה שיטה אשר השתמשה בשיטת זיהוי תבניות כדי לחלץ תכונות ממילות מפתח בקבצי PDF וסיפקה באמצעות אלגוריתם יער אקראי. שיטה זו בדקה כל התקפה על קובץ PDF ולאו דווקא מתקפת js .  
באותה שנה הוצגה PDF-rate גלאי PDF זדוני המבוסס על מטא נתונים ותכונות מבנה הקובץ. שיטה זו יכלה להשיג את התוצאות המיטביות ביותר באותה תקופה גם כנגד תכונות זדוניות לא ידועות עם שיעור סיווג גבוה של 99 אחוז דיוק, והייתה יעילה גם כנגד מתקפות חיקוי.  
בשנת 2013 נבנה גלאי שהתבסס על נתיב מובנה בקבצי PDF והשתמש גם הוא בעץ החלטות ובנוסף במכשיר וקטור מובנה (SVM) כאלגוריתם הלמידה. למרות שהיכולות שלו כנגד פרצות אבטחה לא ידועות הייתה חזקה, החוסן שלו היה חלש באופן יחסי.  
כשנתיים לאחר מכן בשנת 2015 הוצע מודל זיהוי אוטומטי שחילץ את התכונות מהמבנה וגם מהתוכן של הקובץ כדי למנוע התחמקות על בסיס זיהוי מבנה. הדיוק שלו היה טוב יותר ממערכות אחרות המשתמשות בטכנולוגיית ניתוח סטטי, אך היה רגיש מדי למערך הtrain .  
בשנת 2016 עוד חידשו את המודל הנ''ל בעזרת סטטוס הצבעה של מסווגי משנה כדי לשפוט האם החיזוי אמין. במהלך אותה שונה עוד הרחיבו את המודל על מגוון שונה של פורמטים.  
**ניתוח דינאמי**את טכנולוגיות זיהוי אלו יש להפעיל בתוך ארגז חול או מכונה וירטואלית כדי להימנע מחילול המחשב האישי. כאשר התוכנית פועלת הנתונים נבנים על ידי ניתוח כלים דינמיים המאפשרים לקבל ניתוח מדויק יותר, אך זה לוקח הרבה יותר זמן.  
בשנת 2010 הציעו גלאי תוכנות זדוניות מקוונות והשתמשו במסווג בייסיאני כדי לזהות קוד JS זדוני. בשנת 2011 ראו כי מיותר להשתמש בטכנולוגיית סימולציה המבוססת תוכנה והציעו מסגרת חדשה לאיתור מהיר, מדויק של הזרקות קוד.  
**טכנולוגיות זיהוי משולבות סטטי ודינאמי**מצד אחד טכנולוגיה זו לוקחת את היתרונות של שתי הטכנולוגיות, אך מנגד הופך את המודל למורכב יותר.  
בשנת 2011 פיתחו סורק מסמכים זדוני עצמאי המשלב ניתוח מסמכים סטטי וביצוע קוד דינמי כדי לזהות איומי PDF שלא היו ידועים בעבר.  
בשנת 2014 פותח מודל זיהוי תוכנות זדוניות של JS קל משקל באמצעות הפניות של API והשיג ביצועים מצוינים במונחים של דיוק, תפוקה והכללה**.  
באופן כללי בכדי ליצור גלאי בעל ביצועים גבוהים נצטרך מערך נתונים בקנה מידה גדול ועלות של תיוג דגימות אלה עדיין מהווה בעיה.  
(בטבלה לעיל ניתן לראות ציר זמן מסודר של הכלים וההתקדמות שלהם בנושא)**

# התקפות כנגד גלאי תוכנות זדוניות בPDF:

בחלק זה נציג כיצד ההתקפה "מתגוננת מפני ההגנה נגדה". (כלומר כיצד התוקף נמנע מגילוי התקיפה). ובכך נוכל ללמוד עוד גם על דרכיו של התוקף.  
מטרת התוקף- מוגדרת על סמך הפרת האבטחה הרצויה לו. (שלמות, זמינות או הפרת פרטיות)  
באופן כללי, התוקף ירצה לבלבל את המכונה שלנו בעזרת קבצים שיסווגו להפך ממה שהם- וכך בעצם לגרום לבלבול בזיהוי התכונות וסיווג המכונה הסופי.  
דוגמא לכך היא סקריפטים כללים שרצים ברקע, שיוני גופנים, הוספת טקסט וכו')  
הידע של התוקף- יכול להשתנות בין תוקף לתוקף ברמת הידע שלו על היעד, כלומר התוקף עשוי לדעת באיזה אלגוריתם הגלאי משתמש (אלגוריתם עיבוד מקדים וחילוץ תכונות) בפוטנציאל התוקף ישאף לדעת גם את הפרמטרים המסווגים שנלמדו לאחר האימון.  
רמת הידע של התוקף לגבי המכונה מחולקת ל3 תתי קטגוריות:  
-ידע מושלם שנקרא גם white box בתרחיש כזה אנו מניחים כי התוקף יודע הכל על המערכת שלנו למרות שזה ,קורה לעתים רחוקות, מעשית זה אפשרי וכך אפשר להיערך למקרה הקיצוני ביותר.  
-ידע בינוני שנקרא גם gray box בתרחיש כזה אנחנו מניחים שלתוקף ידע מוגבל, הוא מכיר חלק מהתכונות אך אין לו את נתוני האימון ופונקציית הסיווג.  
-אפס ידע שנקרא גם black box בתרחיש כזה אנו יוצאים מנקודת הנחה כי לתוקף ידע מינימלי, אם בכלל.

היכולת של התוקף- לנהל את נתוני הקלט שיוצאו מהpdf לתוך הוקטור שבהמשך יוכנס למכונה. האם לתוקף יש יכולת להשפיע באופן שונה על קלטים שלו שיוכנסו לtrain או לtest  
האם הוא יוכל להשפיע על הקבצים באופן כזה שישבשו את למידת המכונה בזמן שההתקפה שלו תישאר בשלמותה.

בטבלה לעיל מפורטות התקפות שונות והשפעתן אל מול מטרת התוקף, אנו נפרט כעת על כל אחת מהן:  
**התקפות התחמקות evasion attacks**- בהגדרה זו התוקף מצליח לבלבל את המכונה כך שתסווג קובץ זדוני כקובץ נקי.  
**התקפות הרעלה poisoning attacks –** התקפות אלו מכוונות לשלב האימון של המכונה, התוקף מזריק בכוונה דגימות עם תווית שגויה במטרה להפחית את אחוזי הזיהוי ובכך בקצה יגרום לסיווג כללי לא נכון.  
**התקפות פרטיות – privacy attacks –** מטרת התקפות אלו היא לגנוב מידע על המכונה הלומדת באמצעות שליחת דגימות ספציפיות כנגד המכונה.

לסיכום:  
הצגנו באופן כללי את מבנה של קובץ pdf   
לאחר מכן פירטנו על חלקיו ובאילו מהם ניתן להחדיר נוסקה לקובץ.  
כמו כן הצגנו את הנושא של למידת מכונה בהקשר זיהוי קבצי pdf זדוניים והכלים הקיימים בשוק כדי ליצור מכונה שכזו.  
לבסוף הצגנו את הצד התוקף, את מטרותיו וכליו כדי להימנע מהמכונה הלומדת.