האוניברסיטה הפתוחה
המחלקה למתמטיקה ולמדעי המחשב

אנאליזות סטאטיות לחשיפת טבעו של קוד

עבודת תזה זו הוגשה כחלק מהדרישות לקבלת תואר
"מוסמך למדעים" .M.Sc במדעי המחשב
באוניברסיטה הפתוחה
החטיבה למדעי המחשב

על-ידי
שרון רעם

העבודה הוכנה בהדרכתו של **פרופ' אהוד גודס**

אוגוסט, 2011

תודות:

ברצוני להודות לפרופסור אהוד גודס על עזרתו הרבה בהכנת תזה זו ועל הערותיו והצעותיו שתרמו תרומה מכרעת להשלמתה.

ברצוני להודות לבני משפחתי על השלמתם עם השעות הרבות שהשקעתי בתזה ותמיכתם לכל אורך הדרך.

תוכן העניינים

רשימת תרשימים

רשימת טבלאות

תקציר

בסביבה המקוונת והמקושרת המאפיינת עסקים כיום, הן ארגונים גדולים וכן עסקים קטנים מפתחים תלות הולכת וגוברת באינטרנט כבסיס להשגת מידע, תקשורת הודעות אלקטרוניות (e-mails), מסחר אלקטרוני וכדומה.

במקביל להפיכתה של תשתית האינטרנט לנדבך חשוב להעלאת הפריון והתבססותה כמרכיב מרכזי וחיוני לעסקים, היא חושפת את העסק לסוג איום חדש שלא היה מוכר בעולם העסקים שלפני תקופת האינטרנט, איום ההתקפה כנגד תשתיות הארגון תוך שימוש בערוצי התקשורת האלקטרונית כתווך.

אחד המנועים העיקריים בהאצת השימוש ברשת הוא היכולת להשתמש בטכנולוגיות המבוססות על תוכן פעיל לצורך מימוש יישומים מבוססי רשת (Web Based Applications). במקביל לתועלת המתקבלת מטכנולוגיות מסוג זה (Active-x, Java applets, Java Script, Executable files וכדומה), הן עשויות לשמש כנשאיות זדונה (Malware) שמטרתה לפגוע בארגון על ידי פגיעה בתשתיות, גניבת מידע או ביצוע פעולות בלתי חוקיות אלו אחרות.

ההגנות המסורתיות כגון אנטי ווירוסים למיניהם, הגנה בפני חדירה וסווג כתובות באינטרנט, הנמצאות בשימוש נרחב, אינן מספקות הגנה מספיקה כנגד האיומים המודרניים הידועים כבר היום. תוכנות אלו מתבססות על צורת הגנה מגיבה (Reactive), קרי, ברגע שמתגלה איום כלשהו, לדוגמה ווירוס, מעדכנות ספקיות האנטי ווירוס את תוכנת ההגנה הנמצאת במחשבי הלקוח. הבעיה בגישה זו היא בהשארת העסק חשוף לפגיעה החל מהשלב בו מופץ הווירוס ועד הרגע שבו תוכנת ההגנה מקבלת את העדכון. הגנות אלו אינן מספקות מענה מספיק להתקפות לא מוכרות מראש או התקפות מורכבות.

גם גישה החוסמת כל אפשרות להפעלת קוד פעיל במחשב אינה ברת ביצוע היות ורוב התשתיות הנמצאות בשימוש ביום יום כגון מערכות ניהול קשרי לקוחות (CRM), ניהול העסק (ERP) מסחר אלקטרוני, וועידת רשת ודואר אלקטרוני מבוססות על קוד פעיל.

מסיבות אלו ואחרות, גורסת הגישה המודרנית כי פעולת ההגנה חייבת להיות בגישה מונעת מראש (Proactive). על התשתית לאפשר הפעלת קוד פעיל אולם עליה לבצע ניתוח של הקוד או הפעילות שהוא עומד לבצע ולמנוע את פעולתו במקרה שקיים חשד שמטרתו זדונית. קיימים מימושים שונים של צורת הגנה מסוג זה, אם על ידי ניתוח סמנטי של התנהגות הקוד וכלה בהרצת הקוד במעין "ארגז חול" על מנת לבחון את התנהגותו במקום בו לא יגרום כל נזק.

גם השיטה אחרונה המוזכרת אינה מושלמת וקיימות דרכים שונות לעקוף אותה. לדוגמה, הקוד יכול להפעיל את פעולתו הזדונית בצורה אקראית בתקווה שבזמן בו תיבחן פעולתו, יפעל בצורה תמימה לחלוטין. כמו כן, קוד יכול להתנהג בצורה תמימה ונורמאלית לחלוטין כאשר הוא נמצא בסביבה סטרילית אולם יתנהג בצורה שונה כאשר ימצא במחיצת קובץ מסוים, מוגדר מראש.

בשיטת הפעולה האחרונה שהוזכרה, קיימת שיטת ההתקפה המורכבת ממספר שלבים ומרכיבים שונים (קבצים שונים). רק השילוב של כולם ביחד מפעיל את ההתקפה. לדוגמה קובץ אחד יכול להכיל "מנוע" פקודות שלכשעצמו אינו מבצע פעולות חשודות אולם אם יחוש בקובץ מוגדר מראש המכיל סט פקודות מוכר, יריץ פקודות אלו.

לרוב, השלב הקשה בהתקפה הוא להחדיר קבצים אלו למחשב. כותבי זדונה עוסקים בפיתוח שיטות שונות מגוונות לצורך הסוואת תוכנם האמיתי של הקבצים, אם על ידי השיטה הנאיבית של שינוי מזהה הקובץ בשמו (File Extension הנמצא בשימוש בעיקר ב - Windows) או "שתילת" הקוד שלו כחלק מקובץ תמים למראה כגון מסמך כלשהו, תמונה, קובץ מוסיקה (mp3) וכדומה.

לאחרונה נעשו מחקרים רבים במטרה למצוא דרכים יעילות לזיהוי אופיו של הקוד הכתוב בקובץ כלשהו ללא הסתמכות על מאפייניו החיצוניים. בתזה זו אני סוקר את ההתקדמות בתחום תוך ניתוח מספר שיטות וכוונים שונים שנעשו בתחום. כמו כן אני בוחן הרחבה של הדרך לאפיון הקוד על ידי ניתוח התפלגות הבתים (Byte Distribution) שלו.

תרומתה העיקרית של תזה זו היא בהצגת תשתית באמצעותה ניתן, בצורה נוחה ויעילה, לחקור ולאתר את הפרמטרים המתאימים ביותר כאורך N-Gram, בדיקה סטטיסטית ומאפיינים נוספים, ליצירת חתימה מאפיינת לטיפוסים שונים של קובץ. התשתית מאפשרת יצירת חתימה, באופן אוטומטי עבור כל טיפוס קובץ, המתבססת על מדגם מייצג של קבצים מאותו הטיפוס. כמו כן מאפשרת התשתית סריקה של קבצים לא ידועים ואפיונם בהתבסס על החתימה המייצגת.

לשם הוכחת יכולת הורצה התשתית עם מספר פרמטרים בסיסיים עבור קבצים מטיפוס DOCX, PDF, RTF, EXE, DLL, ZIP. תוצאות הריצה שהתקבלו מרשימות ומתקבלת תוצאה כללית של 0.996 במדד ה F-Measure עבור כל הקבצים ועבור כל הטיפוסים שנבחנו, זאת אם הטיפוסים EXE ו – DLL נלקחים בחשבון כקבצים ביצועיים השייכים לאותה קבוצה.

פרק 1 מבוא

בראשית ההיסטוריה של האינטרנט, היו כארבעה שרתים בלבד ברשת. נכון להיום, מספר השרתים עומד על מאות מיליונים. אין זה מפתיע שהתפתחות הווירוסים ושאר המזיקים ברשת קשורה והתרחשה במקביל להתפתחות והצלחת האינטרנט עצמו.

הזדונה המוכרת היום התחילה בתחילה במספר קטן ביותר של ווירוסים. ווירוס הוא למעשה תוכנית מחשב, המוחדרת למחשב ללא ידיעת בעליו וללא רשותו ומבצעת פעילות כלשהי תוך שימוש במשאבי המחשב. עם התפתחות הרשת, קצב ההפצה גדל עשרות מונים ועימו התחכום והמורכבות של הקוד. כיום קיימות שיטות רבות ושונות להתקפת מחשבים. ההבדלים בין השיטות באים לידי ביטוי בדרך כלל לפי שיטת ההפצה, לדוגמה דיסק תוכנה לעומת הרשת, מטרת תכנית הווירוס, גניבת מידע או פגיעה באיכות השרות (QoS), ודרך הפעולה של הקוד, לדוגמה בצורה חשאית בעת גניבת מידע או בצורה בולטת בעת פגיעה באיכות השירות.

בין שיטות הפעולה המוכרות היום ניתן לראות שימוש ב Worms, ווירוס מחשב המשכפל עצמו, סוסים טרויאניים (Trojan Horses), תוכניות זדוניות התוקפות את המחשב מבפנים, רוגלות (Spyware), תוכנות שמטרתן העיקרית היא גניבת נתונים מהמחשב, תוכנות רובוטיות (Bot) המבצעות פעילות מתוך המחשב ביוזמת התוקף, תוכניות Rootkits, המשתמשות בהרשאות Administrator להסתרת עקבותיהם במחשב ושיטות רבות ומגוונות אחרות.

במקביל לפיתוח אמצעי הגנה בפני ההתקפות הידועות כיום, שיטות ההתקפה משתכללות ועושות שימוש נרחב ביכולות המתקדמות של התוכנות המודרניות, אשר במקביל להתקדמותם הטכנולוגית, חושפות חולשות חדשות לבקרים. תכונות חדשות של תוכנות משמשות כר נרחב לאמצעי התקפה, לדוגמה, יכולת הדפדפן לבצע פקודות תסריט (Script) מאפשרת החדרת קוד ביצועי זדוני כחלק מהדף הנטען, התמים לכאורה. התקפות רבות כיום משלבות טכנולוגיות שונות במאמץ משולב, החדרת קובץ קוד במסווה של קובץ תמים לכאורה שאינו עושה דבר, אולם בשילוב קובץ אחר המסוגל להריץ את סט הפקודות שבקובץ הראשון מתקבלת התקפה משולבת, קשה ביותר לניטור ומניעה.

כיום ברור כי אמצעי המניעה המסורתיים כאנטי ווירוס, ניטור אתרי גלישה וכדומה, אינם יעילים מול המערכת המשומנת של תעשיית הזדונה. את תפקיד ההאקרים של פעם, שמטרתם העיקרית הייתה לבחון את יכולתם, תפסו ארגוני פשע בינלאומיים. אלו הבינו שיחס השקעה לתועלת בפשעי המחשב גבוהים לעין ערוך מאשר בפשע רגיל. ארגונים אלו מפעילים מרכזי פיתוח שכל מטרתם לאתר חולשות בתוכנות קיימות ולנצל אותם.

הפער בין השלב בו מופצת הזדונה ולשלב שבו מערכות ההגנה המסורתיות מוגנות בפניה הוא חלון הזדמנויות קריטי לביצוע הנזק ולכן, מערכות ההגנה המודרניות חייבות להתמודד עם שיטות התקפה חדשות ולא מוכרות כבר בשלב ההיתקלות הראשונית. דרישה זו מחייבת מערכות אלו לנתח קוד על פי התנהגותו ו/או מבנהו ללא ידע מוקדם.

חלק חשוב מתהליך ההגנה מסוג זה הוא זיהוי ניסיון להחדיר קוד למחשב במסווה תמים, לדוגמה, מסמך המתיימר להיות טקסט. על ידי שינוי ה Extension שלו, עשוי להיות קובץ המכיל קוד בינארי זדוני אותו ניתן להריץ במחשב, בו הוא נמצא.

מטרתה של תזה זו היא לדון בדרכים לזהות, או לוודא את התאמת תוכנו של מסמך לסוג המסמך אותו הוא מתיימר לייצג. תרומותיה העיקריות הן:

- סקירה כללית של האיומים העומדים בפני מערכות המחשב המודרניות. במסגרת זו נסקרים מספר מאמרים ומחקרים העוסקים בתחומים הקרובים לליבת מחקר זה באופן מפורט.
- הצגת תשתית באמצעותה ניתן, בצורה נוחה ויעילה, לחקור ולאתר את הפרמטרים המתאימים ביותר ליצירת חתימה מאפיינת לכל טיפוס קובץ.
- ביסוס התשתית על מערכת חוקים המאפשרת גמישות רבה בהגדרת מאפייני ומבנה החתימות.
- הוכחת יכולת המסגרת ליצור חתימות עבור טיפוסים כגון pdf, doc, exe, dll ואחרים ושימוש בחתימות אלו לאיתור הקבצים מטיפוסים אלו. במסגרת זו, מתוך קבוצת בת 1400 קבצים לא מזוהים, זוהו בהצלחה מלאה 92% מהקבצים. מתוך הקבצים שלא זוהו בהצלחה מלאה, הקבצים אובחנו בקירוב והטעויות השגורות היו כאבחון קובץ exe כקובץ dll וטעויות דומות. מתוך 1400 קבצים, רק 6 לא זוהו בהצלחה, ללא סיבה נראית לעין, ודורשים בדיקה נוספת.

מבנה התזה הוא כדלקמן:

פרקים 2 ו – 3 הם פרקי רקע. בפרק 2 מוצגת סקירה כללית של תופעת הקוד הזדוני. התפתחותה, מהווירוסים הראשונים ועד התעשייה המפותחת כיום, תוך הסתכלות מזוויות שונות, הן מצד מערכת המחשב, המוטיבציה של האדם מאחורי הקוד וסוגי איומים. בפרק 3 מובאת סקירה של מספר מאמרים, ציון מחקרים שונים העושים שימוש ב N-Grams למטרות שונות, ציון מחקרים בנושא זיהוי טיפוס הקובץ וכמו כן תיאור כללי של מוצר מסחרי. כל אלו העוסקים בשיטות סטאטיות המשמשות לצורך זיהוי אנומליות בתכולת קבצים או דפי רשת. בפרק 4 מתוארת התרומה העיקרית של התזה ומוצגת הארכיטקטורה החדשה לזיהוי טיפוס של קובץ תוך שימוש באנאליזות סטטיסטיות על תכולתו.

פרק 2 סקירה כללית – סוגי איומים

לרבים, נראית תופעת הזדונות כחדשה. המגיפה המאפיינת את השנים האחרונות חשפה את מרבית משתמשי המחשב האישי ברשת לסוגים שונים של תוכנות זדוניות. בפועל, התופעה מוכרת עוד מראשית שנות השבעים. אומנם, לא ידוע על התקפות וירוסים על מערכות מחשב בתקופה שקדמה לשנות השבעים, אולם הדבר נבע בעיקר מהיות תחום המחשוב בחיתוליו, כאשר מספר מועט מאוד של אנשים היו בעלי יכולת מעשית לכתיבת תוכנות לניצול חולשותיהם של מערכות המחשב הראשונות.

הבעיה התחילה לצוף על פני השטח עם הפיכתם של המחשבים למצויים וזמינים יותר. וירוסים החלו להופיע ברשתות ייעודיות כגון הווירוס Creeper ברשת ה ARPANET [1 p. 28] שקדמה לרשת האינטרנט. עם ההתקדמות הטכנולוגית המואצת, אנו עדים לגידול מואץ ברמת הפגיעות של היישומים המתקדמים. טכנולוגיות חדשות מביאות עימם דרכים נוספות, מוכרות פחות או יותר, להתקפה על ארגונים המשתמשים ביישומים אלו. אם בשנות השישים והשבעים, דרכי ההתקפה על הארגון התמקדו בעיקר בהתקפה פיזית על מערכת המחשב [1 p. 5 - 8] ובכתיבת קוד שמטרתו סחיטת הארגון [1 p. 22], ניתן לראות כיום מעבר לשיטות מתוחכמות ביותר לצורך ניצול חולשות מערכות המחשוב למטרות רווח.

העבירות על החוק, המתוארות במסגרת תזה זו, מתוארות תחת הכותרת Cybercrime. לכינוי זה הגדרות רבות ושונות אולם דרך מקובלת במיוחד היא הגדרת כל פשע שמטרתו היא פגיעה במחשב או מערכת מחשבים או שהאמצעי לביצועו הוא מחשב, כ Cybercrime.

2.1 חשיפת מערכת המחשב לזדונה

אחת מדרישות הסף לפגיעות של מערכת מחשב להתקפות זדוניות, היא האפשרות להריץ תוכנות חיצוניות על מערכת המחשב. מערכת מחשב ייעודית שאינה מאפשרת הרצת קוד חיצוני, אינה חשופה לחדירת זדונה ונחשבת מוגנת יחסית, למעט חשיפה לניצול חולשות על ידי גורמים השייכים ליצרן מערכת ההפעלה עצמה.

אולי הדוגמה הבולטת ביותר כיום היא חשיפתם של הטלפונים הסלולאריים להתקפות. כיום, הטלפון הסלולארי הוא למעשה מערכת מחשב מלאה, המאפשרת קריאת מסמכים, האזנה למוסיקה והרצת תכניות המורדות מהרשת לטלפון. לפני פיתוח יכולות אלו, לא היה ידוע על התקפות כלשהם על מכשירי טלפון סלולאריים, בעוד שכיום אנו חוזים ביותר פגיעות במכשירים אלו במגמה שהולכת וגוברת [2].

כיום נכתבו אינספור וירוסים, סוסים טרויאניים, תולעים ורעות אחרות עבור מספר רב של מערכות הפעלה ויישומים שונים בתחומים שונים. למרות זאת, עדיין ניתן לראות מספר מערכות ותוכנות שנחלצו מפגיעת זדונות עד כה. לצורך ההבנה מדוע מערכות אלו נותרו נקיות בעוד שאחרות מותקפות על בסיס קבוע, עלינו לבחון מה גורם ליישום אחד להיות פגיע יותר מיישום אחר.

- נפיצות היישום - אכן, בין היישומים הנתונים ביותר להתקפות, בעת כתיבת נייר זה, ניתן למצוא את דפדפן האינטרנט ו Flash של Adobe, הנפוצים ביותר [3]. כמו כן, אחד התחומים בעלי קצב הגידול המואץ ביותר כיום, הוא המערכות הניידות עליהם מתבססים הטלפונים הסלולאריים החכמים. לפי אנליסטים בענף, מכשירים חכמים יהיו חשופים יותר מבעבר להתקפות, וזאת כתוצאה משכיחות השימוש ההולכת וגוברת בהם [4].
- פתיחות המערכת לריצת תכניות חיצוניות - היכולת של המערכת לאפשר לתכניות, שנכתבו על ידי מקור חיצוני, לפעול במסגרת המערכת. ללא יכולת זו, המקור היחידי להתקפות על המערכת יכול לבוא אך ורק מהארגונים המפתח, או מפעיל, את המערכת. מגבלה זו מצמצמת משמעותית את האפשרות להתקפה על המערכת.

- איכות המערכת המותקפת - במיוחד בהתייחס להיבט הבטיחותי. בעת פיתוח מערכת, נלקח בחשבון גם גורם ההגנה על המערכת, אולם, לעיתים במודע או לא, נשארים אזורים פגיעים יותר מאחרים. הסיבה יכולה להיות תסריט שהמפתח לא חזה מראש בעת הפיתוח או פגיעות מכוונת שמטרתה לשפר את נוחות השימוש במערכת, כגון אי הגבלת הגישה לקבצים מסוימים או אפשור פעולות מערכת, שבכל מקרה אחר, היו מוגבלות משיקולי הגנה. מערכות מסוימות אף מאפשרות הרשאות מורחבות, המתבסס על ההרשאות המקסימאליות של המערכת עצמה, לקוד הרץ תחת חסותן. דוגמה זו אופיינית למשל לדפדפני אינטרנט, בעיקר בגרסאות מוקדמות, הרצים תחת הרשאה של Administrator ומעניקים הרשאה זו לקוד אותו הם מריצים. הרשאה זו מאפשרת לקוד לבצע פעולות מערכת באין מפריע.
- תיעוד מפורט ומדויק של המערכת – למרבה האירוניה, סיבה זו מהווה אף היא, גורם לפגיעות המערכת. כותבי זדונות זקוקים לתיעוד לצורך הבנת הממשקים שבהם יש להשתמש לצורך ניצול פגיעות המערכת. ככל שהתיעוד מדויק ומפורט יותר, כך קלה יותר עבודתם. במערכות בהם רמת התיעוד נמוכה, על התוקף לבצע לעיתים Reverse Engineering במטרה לזהות את הנקודות הפגיעות וכן לאתר ממשקים המאפשרים לנצל נקודות אלו. סיבה זו מסבירה אולי, מדוע מערכות קוד פתוח, כגון Linux או הדפדפן Firefox, נחשבות, לעיתים שלא בצדק, כמוגנות יותר ממקבילם מבית Microsoft. כמובן שקיים כמובן הנימוק הנגדי שמערכות מתועדות היטב מאפשרות גילוי חולשות בקלות רבה יותר ומניעתן בידי כותבי מערכות ההגנה.

סיבות אלו ונוספות יכולות להסביר את מידת פגיעותם של מערכות שונות להתקפות, וכן מדוע מערכת אחת נחשבת כפחות פגיעה מאחרת.

2.2 האדם שמאחורי ההתקפה

בבואנו לחקור דרכים להגנה על מערכת המחשב בפני התקפות, עלינו, כשלב ראשון, להכיר את האדם העומד מאחורי התקפות אלו. מציאת תכונות אופייניות של כותב הזדונה (האקר, בעגה עממית) תקל עלינו במציאת הגנה מתאימה בפני הקוד אותו כתב. המונח האקר (Hacker) מתאר בדרך כלל חובב בעל ידע רב במחשבים אולם בשנים, מאז התפתחות ה Cybercrime, הפך המונח לשם נרדף לפורץ מחשבים או מתכנת זדונות.

מה למעשה גורם לאדם להיהפך להאקר? האם באמת ניתן ליחס תכונות משותפות לאותם אנשים, המשפיעות על הפיכתם לכאלה?

ניתן ליחס להאקרים מניעים שונים היכולים להיות מושפעים מהאדם או מקבוצה אליה הוא שייך. מניעים אלו עשויים להשתנות לאורך הזמן. לדוגמה, שעמום בביה"ס מוזכר כאחת מהסיבות הראשונות להיהפך להאקר [5 p. 52] אולם המניע עשוי להשתנות ואף להיעלם במקרה של מעבר לאוניברסיטה. בין המניעים הנוספים נמצא את הסקרנות, הרצון ללמוד ולחקור. הרצון לעמוד באתגר, תוך שאיבת סיפוק אישי מעצם ביצוע הצעד הלא חוקי והצלחה בביצועו. הרצון לזכות בהכרה, אם של חוג החברים, העמיתים לתחום או שותפים למקצוע. תחושת הכוח המתלווה ליכולת לפרוץ ואו לפגוע באזורים מוגנים כביכול [6 p. 45 - 46]. מניעים אלו מיוחסים בדרך כלל להאקרים חובבים.

מניע שכיח נוסף הוא הרצון לפגוע, למרוד ולערער את היסודות עליהם מושתת "הממסד". מטרתם של האקרים מסוג זה היא ליצור נזק רב ככל האפשר ולעיתים תכופות הם נרתמים לעזרת ארגון טרור זה או אחר או אפילו ארגון ממשלתי, במטרה משותפת לפגוע במתנגדים לארגון. סוג פעילות זה נמצא תחת הכותרת "האקטיביזם" (hacktivism) ואנשים העוסקים בו עושים כך בדרך כלל מתוך תחושת שליחות ובמטרה להסיט את תשומת הלב למטרות פוליטיות או סוציאליות, בהם הם מאמינים.

וכמובן, תמיד יהיה המניע הבסיסי של בצע הכסף. ההאקרים המונעים על ידי מניע זה פועלים מתוך מטרה להשיג רווח בדרכים לא חוקיות כגון גניבת זהות והתחזות, גניבת פרטי אמצעי תשלום, ריגול תעשייתי, פגיעה באיכות השירות של מתחרים ועוד דרכים יצירתיות שונות. ואכן, בשנים האחרונות אנו עדים לחדירה של ארגוני פשע מאורגנים לתחום ה Cybercrime לאחר שזיהו את פוטנציאל הרווח העצום הגלום בו [1 p. 50,4].

את האקרים ניתן לחלק לקטגוריות שונות. חלוקה טבעית ראשונה היא בין אלו הפורצים לארגון מבחוץ לבין אלו הפורצים מתוכו. לפי הבולשת הפדראלית בארה"ב (FBI), 70% אחוז מהפריצות למערכת בארגון מבוצעות על ידי אנשים בתוך הארגון. למעשה, המשימה העומדת בפני ההאקר החובב ברצונו לפרוץ למחשב בארגון אינה כה מסובכת כפי שניתן אולי לחשוב. לדוגמה, חיפוש מהיר באינטרנט יגלה רשימת אדמיניסטראטורים עם סיסמאות המחדל שלהם, כפי שיצאו משערי הספק עבור מחשבים מסוגים שונים, מערכות בסיסי נתונים שונות ומערכות מורכבות אלו ואחרות כמערכות ERP למיניהם. סקר שנערך ב 2007 מגלה כי כ – 20% מהמערכות המותקנות אצל לקוחות עדיין כוללות בתוכן את הרשימות המקוריות, עובדה המאפשרת חדירה מהירה למערכת ללא מאמץ מיוחד [7].

מדוע עובדים מתוך הארגון יבקשו לפגוע בו? מאותן הסיבות שנמנו למעלה. החל מהסיבה הבסיסית, הסקרנות, הרצון של העובד לבדוק את גבולות יכולתו, התקלות מקרית בקובץ בעל כותרת מושכת עין כרשימת שכר וכדומה. הרצון לנקום, העובד לא קיבל עליית שכר כמובטח, זכה לנזיפה ממנהלו או כל סיבה אחרת הגורמת אצלו לתסכול. סיבה שלישית, לא פחות חשובה ולעיתים בעלת יכולת גרימת נזק גדול יותר היא ריגול תעשייתי. העובד בארגון נחשף במהלך עבודתו למידע חסוי שנחשב כנכס לחברה. מידע זה עשוי להיות בעל ערך רב למתחרים והפיתוי הכספי עשוי להיות רב.

הפורצים מחוץ לארגון ניתנים לחלוקה לפי רמת התחכום שלהם. החל מההאקר המתחיל והחובב, script kiddies בעגת ההאקרים, המשך בהאקרים מתוחכמים יותר, העובדים באופן פרטני מנותק מאחרים וכלה בהאקרים מקצועיים המאוגדים תחת ארגון (פשע או ממשלתי) או קבוצה כלשהי בעלת מטרה משותפת. חלוקה נוספת יכולה להיות לפי המטרה בפעילות ההאקר, הגנה, מחקר, יצירת נזק, עשית הון בניגוד לחוק, ריגול תעשייתי ומטרות אחרות.

להאקרים רבים תכונות אופי דומות. ביניהם נוכל למצוא:

- הקפדה על פרטים - ההאקר המתוחכם יקפיד לרוב על הפרטים, יאסוף מידע בדקדקנות על מטרותיו ויכין את ההתקפה בפרוטרוט. לעיתים יעשה ניסיון לקיצורי דרך במטרה לחסוך בזמן אולם לא כשיטה.

- יכולת ההתמדה - ההאקר הממוצע יכול להשקיע זמן ניכר במטרה לחשוף חולשות במטרתו, למצוא מידע על התוכן המאוכסן ואף מערכות נוספות המחוברות למערכת שתחת התקפה. רק האקרים מהרמה הנמוכה ביותר, או כאלו שרוצים להצטייר כהאקרים, ינסו לתקוף מטרה ללא מידע מוקדם.
- ההערכה העצמית - במחקר שנערך על האקרים התגלתה קורלציה (קשר) בין רמת ההאקר לבין ההערכה העצמית שלו. ככל שההאקר מתוחכם ומקצועי יותר, כן רבה ההערכה העצמית שלו. לפי מחקר זה, מקובל להאמין שההאקרים בעלי ההערכה העצמית הגבוהה יותר הם בדרך כלל בעלי מוטיבציות חיוביות יותר במטרתם כגון איתור חולשות במערכת לצורך סגירת פרצות.
- יצירתיות – ההאקר לא יראה במכשול כבעיה אלא כאתגר שתמיד ניתן למצוא דרך לעמוד בו.

השילוב בין תכונות אלו ואחרות מהווה את המפתח להצלחה של ההאקר. התכונות אופייניות לאנשים רבים ולכן לא יוכלו, אולי, להקל באפיון ההאקר, אולם הימצאותם עשויות להשפיע על האדם בהופכו להאקר [8].

2.3 האיומים העומדים בפני מערכת המחשב

בתחילת עידן טכנולוגית המידע, היינו עדים בעיקר לפשעי מחשב שבוצעו על ידי עובדים ממורמרים או לא ישרים של הארגון. פגיעה פיזית בתשתיות המחשוב היווה את אחד האיומים עיקריים באותה התקופה. צורות נוספות של פגיעה היו על ידי שימוש בגישה למערכות, שניתנה להם במסגרת עבודתם, לצורך שינוי נתונים במטרה להשיג רווחים, או פגיעה בנתונים מתוך שאיפת נקם.

עם התפתחות התקשורת, בתחילה דרך קווי הטלפון, החלה הפצה של מידע על טכנולוגיות התקשורת, דבר שהאיץ את התפתחות שיטות החדירה למערכות ורשתות מחוץ לארגון.

בתחילת שנות השמונים, החלו מתכנתים לכתוב זדונות, בתחילה בתור שעשוע ורצון להבליט יכולות ובהמשך מתוך כוונה מוצהרת לפגוע בתשתיות. בתחילה נכתבו וירוסים תמימים למדי אולם בהמשך נכתבו גם תוכנות בעלות יכולת שכפול עצמי.

עם פריצת עידן האינטרנט, שחשף מספר עצום של מערכות מחשב בכל העולם, החל תהליך מואץ של פריצה למערכות מחשב, בעיקר תוך הסתמכות על הגנה מוגבלת ביותר של מערכות מיושנות או הגדרות מחדל שנותרו ללא שינוי במערכות חדשות ומתקדמות יותר וזאת, בשל חוסר מודעות של מנהלי מערכות בארגונים באותה התקופה. המטרה העיקרית בחדירות, באותה תקופה, הייתה למטרות גריפת רווח, פעילות פוליטית או חבלה בזדון [1 p. 4].

שנות התשעים ולאחריהם שנות האלפיים הביאו איתם מערכות חזקות ומוגנות יותר, דבר שגרם לדעיכה עד היעלמות של שיטות פגיעה ישנות, אולם חולשות חדשות בטכנולוגיות חדשות, שרק הפציעו, פתחו אפשרויות נרחבות נוספות להתקפה בפני ההאקרים. עם זאת, המטרות העיקריות לא השתנו, גריפת הון, פעילות פוליטית, אם באופן פרטי או על ידי ממשלות וארגונים, וחבלה.

את שלב הכנסת הזדונה, במהלך מחזור החיים של התוכנה, ניתן לחלק לשתי תקופות. לפני ואחרי שחרור התוכנה מהיצרן. האיש העומד מאחורי הכנסת הקוד לפני השחרור הוא בדרך כלל מתכנת בתוך הארגון של ספק התוכנה. אחרים, פרטיים או ארגונים, העומדים מאחורי הכנסת זדונה, יעשו זאת בדרך כלל לאחר שחרור התוכנה לציבור המשתמשים [9].

נכון להיום, קיימות אינספור דרכים ושיטות לפגיעה בתשתיות המחשב. חוקרים רבים אינם תמימי דעים על ההבדלים בין השיטות השונות אולם החלוקה בסעיפים הבאים מקובלת על הרוב.

2.3.1 דלת אחורית (Backdoor)

דלת אחורית מוגדרת כדרך לא מתועדת לחדירה למערכת מחשב. כיום ניתן לסווגה לפי מבצע החדירה והאופן בו מבוצעת החדירה. בצורתה הטריוויאלית, דלת אחורית מאפשרת מעקף של מערכות הגנה על התוכנה. דוגמאות לכך הם סיסמא קשיחה מוגדרת מראש, הידועה אך ורק למתכנת, או רצף פעולות שיגרום לעקיפת מערכות ההגנה. מממש דלת זו הוא בדרך כלל כותב התוכנה. לעיתים המטרה בכתיבתה תמימה לחלוטין, כגון מתן גישה בעתיד לצרכי תחזוקה ולעיתים במטרה זדונית לחדור בעתיד למערכת לצורך עשיית רווח.

דרך נוספת למימוש דלת אחורית היא בעזרת תוכנה ייעודית. לעיתים פורץ עשוי למצוא פרצה באבטחת המערכת, המאפשרת לו גישה לאזורים מוגבלים. פרצה זו תאפשר לו גישה חופשית לעשות ככל העולה על רוחו וזאת, עד לשלב בו הפרצה מאותרת על ידי יצרן המערכת ונאטמת על ידו. אחת הדרכים להתמודדות הפורץ בהגנה על מנת לאפשר לו גישה בעתיד, היא שימוש בתוכנת דלת אחורית [10]. תוכנה מסוג זה מאפשרת גישה למערכת המחשב ללא שום צורך במעבר דרך תהליך ההזדהות בכניסה למערכת וללא הסתמכות על החולשות שאפשרו את החדירה למערכת מלכתחילה.

תכנית דלת אחורית אופיינית מורכבת משני חלקים [11]:

- תכנית שרת, אשר בעת פעולתה, מאזינה לפקודות המגיעות בערוץ מסוים דרך שכבת ה TCP או ה UDP. בעת קבלת הפקודה, התכנית מבצעת פעולה כל שהיא על המערכת.
- תכנית לקוח אשר תפקידה לשלוח את הפקודות לתכנית השרת.

תוכנת דלת אחורית נחשבת כמסוכנת משמעותית מווירוסים טיפוסיים, זאת כיון שהיא מאפשרת למשתמש שהתקינה לחזור בעתיד ולהשתלט על המחשב למטרת ביצוע פעולות לא חוקיות, וזאת מכל מקום ברשת.

ניתן להתקין דלת בדרכים שונות. דרך נפוצה ביותר כיום, היא לצרף תוכניות Backdoor כקובץ מצורף, בעל שם תמים, לדואר אלקטרוני. המשתמש התמים מפעיל תוכניות אלו, אשר בעת התקנתן, מסוות את פעולתם למנוע את איתורם בעתיד. מסיבה זו נחשבות לעיתים תוכניות דלת אחורית גם כסוס טרויאני (על כך בהמשך).

דלת אחורית "איכותית" מסווה עצמה כאחרת, מממשת בעצמה את שכבות ה TCP/IP, משתמשת בערוצים סודיים (Covert Channels) להתקשרות עם העולם החיצון וממשת בעצמה פקודות Shell בסיסיות כ ls, mkdir, ps, kill, put, get על מנת להסוות פעולותיה ולמנוע תיעודם. תוכנית זו בנויה כתכנית הניתנת להפעלה בכל סביבה, ללא שום תלות בספריות מערכת חיצוניות. תכנית הבנויה בצורה זו מקנה כר פעולה נרחב למשתמש בה ועם זאת, קשה ביותר לאיתור.

בין תוכניות הדלת האחורית המוכרות ביותר נמצא את ה Netbus ואת BackOrifice (הנחשבות גם כסוסים טרויאניים, לאור שיטות הסוואת פעולתם), המאפשרות לפורץ גישה מרחוק לצורך שליטה על השרת הפגוע [10]. בעת האחרונה היינו אף עדים בישראל למקרה של החדרת תכנית דלת אחורית בארגונים מסחריים לפי הזמנתם של ארגונים מתחרים, דבר שהוביל למעצר מספר דמויות מפתח [12].

2.3.2 ווירוס

ווירוס בעולם המחשבים מתואר כקטע קוד, הנכנס כחלק מתוכנית מחשב תמימה. התוכנית הנגועה נחשבת כמארחת של הווירוס. קטע הקוד עצמו אינו מסוגל לפעול בנפרד מהתוכנית המארחת והוא זקוק לה על מנת לחדור למטרתו ולפעול שם. לדוגמה, על מנת לחדור למערכת מחשב, הווירוס יכול להתחבר לתכנית שירות, למשל מעבד תמלילים, אשר בהפעלתה, תפעיל את הקוד של הווירוס אשר יכול לשכפל עצמו לתוכנות אחרות במערכת ולנטרל הגנות של המערכת מבפנים [9].

לווירוס מטרות שונות, החל מהטרדה או הצגת עמדה (פוליטית או אחרת) של המחבר, המשך בגניבת נתונים, מסחריים או אחרים וכלה בגרימת נזק פיזי של מחיקת נתונים ו\או מערכות מחשב שונות. כפי שמוזכר בסעיף הקודם, הווירוס מתוכנן בדרך כלל לבצע שיכפול עצמי, קרי, ליצר עותקים של עצמו ולהצמידם לקוד של קבצים אחרים.

ניתן לסווג ווירוס לפי סיווגים רבים ושונים, אולם קיימת חלוקה טבעית לפי הקוד בו כתוב הווירוס. למשל, ווירוסים בשפת מכונה או ווירוסים הכתובים בקוד המיועד למתרגם, כגון שפות תסריט (Script) למיניהם. ווירוסים הכתובים בשפת מכונה יכולים לרוץ ישירות תחת מערכת ההפעלה. שיטת כתיבת ווירוסים מסוג זה אופיינית בדרך כלל לתקופות מוקדמות יותר באבולוציה של הזדונה וניתן לחלקם בין שלוש קטגוריות שונות [11]:

- ווירוס מדביק – ווירוס שמצרף עצמו כחלק מהקוד של תכנית ביצועית, כאשר בעת הרצתה, מדביק תכניות נוספות במערכת, הנמצאות על התוכנית המארחת בקשר גומלין. ווירוס ירושלים (הידוע גם כיום שישי השלוש עשרה) היה מטיפוס זה [1 p. 20].

- ווירוס סקטור אתחול – כצפוי, ווירוס זה ממקם עצמו בסקטור האתחול של הדיסק הקשיח או המדיה הניידת כדיסקט, דיסק CD או אפילו דיסק און קי. סקטור האתחול הוא מקום מיוחד על הדיסק, בו נמצאים נתונים על מבנה הדיסק, רשימת הקבצים ומיקומם ולעיתים תוכניות התנעה שונות. במקרה של כונן הדיסק הקשיח הראשי של המחשב, כולל סקטור זה את תוכנית ההתנעה של מערכת ההפעלה של המחשב המורצת על ידי ה BIOS בעת הפעלת המחשב. במקרה של מדיה ניידת, יכול סקטור זה להכיל תוכניות התנעה שונות אשר, בהתחשב בהגדרות המחשב, יורצו באופן אוטומטי בעת טעינת המדיה למחשב.
ווירוסים מסוג זה ניתנים להסתרה בקלות ומסוגלים לגרום נזק רב לאור קרבתם למערכת ההפעלה עד להשבתה מלאה של המחשב. בדרך כלל הם יפעלו בזיכרון המחשב, כך שתהיה להם גישה לכל קובץ שיופעל לצורך הדבקתו. בין הסימפטומים המוכרים של הווירוס הם הודעות שגיאה בעת הפעלת המחשב ואי יכולת לעבור את תהליך האתחול של מערכת ההפעלה. בין הווירוסים מסוג זה הידועים לשמצה נמצא את ה Michelangelo מ 1992 שתוכנן לפגוע ולהרוס ספריות בדיסק הקשיח ב 6 במרץ בכל שנה, תכונה המאפיינת אותו כפצצה לוגית (Logical Bomb), קרי, תוכנית שתוכננה לפעול כאשר מתמלאים תנאים מסוימים כגון תאריך, שילוב תהליכים רצים וכדומה [1 p. 20].
- ווירוס מורכב – ווירוס זה כולל תכונות משני סוגי הווירוסים שצוינו לעיל וככזה, פועל גם על סקטור האתחול, נמצא בזיכרון המחשב ולכן גם מצרף עצמו לקוד של קבצים ביצועיים.

קבוצה הווירוסים הכתובים בקוד המיועד למתרגם (Interpreter), מיועדים בדרך כלל לרוץ תחת יישומים ייעודיים המסוגלים לקרוא את קוד הווירוס ולתרגם אותו לפעילות כלשהיא. סוג זה של ווירוסים נפוץ ביותר בתקופה האחרונה וזאת לאור הקלות בה ניתן לכתוב אותם, עובדה המאפשרת גם למחברי ווירוסים חובבניים לקחת ווירוס קיים ולשנות את הקוד שלו במטרה לייצר תמורה נוספת שלו. ריבוי היישומים, המסוגלים להריץ פקודות תסריט או מקרו למיניהם, אף הוא עוזר בקידום. די לציין את הדפדפנים השונים, חבילת אופיס של מיקרוסופט, יישומי דואר למיניהם, נגני מדיה וכדומה, כולם מסוגלים להריץ שפת תסריט זו או אחרת.

גם קבוצת ווירוסים זו ניתנת לחלוקה לפי:

- ווירוס מקרו (Macro Viruses) – ווירוס מסוג זה הוא השכיח והמצליח ביותר. אחת הסיבות לשכיחותו היא הקלות בה ניתן להצמידו למסמכים שונים, כגון מסמך עיבוד תמלילים או גיליון נתונים אלקטרוני. הווירוס מנצל את יכולת היישום המקבל את המסמך לקרוא ולבצע את פקודותיו לצורך ביצוע פעולותיו וכן להתרבות על ידי העתקה למסמכים אחרים.
  לווירוסים מסוג זה יכולת התרבות מהירה במיוחד בגלל השימוש התדיר בהחלפת מסמכים בין משתמשים שונים. ווירוסים אלה גם מדביקים לעיתים תבניות המשמשות ליצירת מסמכים חדשים. מרגע ההדבקה, כל מסמך חדש שייוצר על ידי המערכת הנגועה, יהיה נגוע בעצמו.
  אחד הווירוסים המפורסמים ביותר מסוג זה הוא Melissa משנת 1999, ווירוס שנכתב בקוד מקרו של MS-Word אשר בהפעלתו, פנה למערכת הממשק של חלונות לדואר אלקטרוני (MAPI-standard) במחשב הקורבן ושלח עצמו ל 50 כתובות דואר אלקטרוני שהיו ברשימת הכתובות במחשב תוך הוספת המסמך הנגוע לדואר והוספת כותרת נושא שכללה את שם המשתמש במחשב הנגוע. הקורבן שמקבל את הדואר, פותח את המסמך הנגוע המצורף ותהליך ההדבקה ממשיך. בגלל מקדם ההכפלה (50), ווירוס זה הצליח להתפשט במהירות שטרם נצפתה עד אז. התופעה גרמה לקריסת מערכות מחשב רבות לאור כמות דואר הזבל שנשלח והנזק שנגרם היה רב [1 p. 34].
- ווירוס תסריט (Scripting viruses) – ווירוס תסריט דומה במהותו לווירוס מקרו אולם, בניגוד לווירוס המקרו, המתבסס על קוד אותו מסוגל להריץ יישום מסוים בלבד, ווירוס התסריט כתוב בקוד (לדוגמה Perl, Java Script, VB Script) הניתן להרצה בעזרת תהליכים סטנדרטיים הנמצאים כמעט בכל מחשב. ווירוס ידוע לשמצה המבוסס על שפת תסריט הוא I Love You משנת 2000. ווירוס זה, שהיה מבוסס על שפת VB Script, הגיע למחשב המותקף דרך דואר אלקטרוני ובהפעלתו, הדביק את ספר הכתובות של יישום הדואר ושלח עצמו לכל המכותבים בספר הכתובות. שיטת עבודתו, מהירות פעולתו והפצתו המהירה זיכו אותו בתואר התולעת (Worm, על כך בהמשך) בעלת תהליך ההפצה המהיר ביותר עד אותה תקופה [1 p. 35].

קיימות טכניקות שונות בהם ווירוסים מחביאים את עצמם ו\או את מטרתם. לעיתים ווירוס עושה שימוש ביותר מטכניקה אחת וזאת על מנת לדחות ככל שניתן את שלב הגילוי שלו, כך שיוכל להפיץ עצמו לכמות גדולה יותר של מחשבים. בסעיפים הבאים נסקור חלק מטכניקות ההסתרה הידועות [11]:

- ללא החבאה – שיטה זו היא ללא ספק הפשוטה ביותר ליישום אולם אינה יעילה במיוחד. האבחון וההגנה בפני ווירוס, הנוקט בשיטה זו, פשוטים ביותר [13].

- הצפנה (Encryption)– השיטה הוותיקה והמסורתית לגילוי ווירוסים הייתה זיהוי חתימת הווירוס בתוך קוד התוכנית הנבדקת והשוואה לרשימת חתימות וירוסים ידועים מראש.
שיטת הסתרה זו מערבת תהליך של הצפנה ופענוח של גוף הווירוס. ההצפנה נעשית לעיתים בעזרת מפתח אקראי, כך שכל מופע של הווירוס נראה שונה מרעהו, דבר המקשה על גילויו על ידי השוואה. בשיטה זו קוד ההצפנה והפענוח אינו משתנה ולכן ניתן לאיתור בקלות יחסית. מסיבה זו השיטה אינה מתארת בדיוק את מה שמפענחי צפנים היו מכנים בשם "הצפנה" אלא יותר שיטה לטשטוש המצאות הווירוס [13].

- רב צורתיות (Polymorphism) – רב צורתיות היא למעשה צורה מתקדמת יותר של הצפנה עצמית. בצורת הסתרה זו, קוד הווירוס עובר תהליך של מספר שינויים במקביל , הן בהגדרות ההצפנה, והן בקוד ההצפנה עצמו. גוף הווירוס עצמו אינו משתנה אולם נראה שונה בכל מופע בגלל אופי ההצפנה המשתנה. הווירוס הראשון הידוע שפעל בשיטה זו הוא "1260", שנכתב בארצות הברית ב 1990. הווירוס השתמש בשתי מפתחות לפענח את הגוף שלו, אולם המעניין בפעולתו הוא העובדה שהוא הכניס קוד חסר שימוש לתוך המפענח שלו, שמטרתו האחת והיחידה הייתה לשנות את מראהו וחתימתו [14 p. 7.5].

- שינוי צורה (Metamorphism) – הרעיון בבסיס שיטת הסתרה זו הוא לשנות את קוד גוף הווירוס עצמו, לעומת הצפנתו בשיטות שתוארו לעיל, מה שחוסך את הצורך במפענח [13]. ניתן לשנות את קוד הווירוס במספר אופנים, לדוגמה, הוספת קטעי קוד לא רלוונטיים באזורים שונים בתוך קוד הווירוס, או שינוי מיקום קטעי קוד ביצועיים וסדר קריאתם בהתאם, כל זאת על מנת לקבל קוד ביצועי שונה באופן משמעותי מקוד הווירוס המקורי.

- הצפנה חזקה (Strong Encryption) – אחת המגרעות העיקריות של שיטת ההצפנה שתוארה לעיל, היא הדרישה שהווירוס יכלול, בין היתר, גם את המפתח לפענוח שלו. לצורך פתרון הבעיה קיימות שתי שיטות נוספות להשגת המפתח. ראשית, ניתן להשיג את המפתח ממקור חיצוני. המקור יכול להיות אתר אינטרנט, אולם במקרה זה הווירוס יאלץ להחזיק בכתובת האתר, עובדה שתקל על זיהויו. ניתן להתגבר על הבעיה האחרונה על ידי שימוש באתר חיפוש לצורך איתור המפתח.
מקורות נוספים יכולים להיות הודעות דואר אלקטרוני, הודעות מסר מיידי וכדומה. דרך נוספת היא על ידי שימוש בווירוס בינארי, שבו הווירוס מורכב משני חלקים, ורק המצאות שניהם במערכת תגרום לפעולת הווירוס. ניתן לדמיין וירוס מסוג זה שבו חלק אחד מוצפן והחלק השני מכיל את המפתח, אולם תסריט מסוג זה אינו סביר ביותר אם שני החלקים ישוחררו וינועו ביחד. במקרה זה הסיכוי לגילויים גדול. לעומת זאת, אם כל חלק ישוחרר בנפרד, עם פרק זמן סביר ביניהם, הסיכוי למציאתם קטן יותר.
הדרך השנייה להשגת מפתח היא מתוך המערכת הנגועה. המקור למפתח יכול היות שם התחום (Domain) של המחשב, שם משתמש או כל נתון אחר המאפיין את המערכת. שיטה זו מקלה על מיקוד ווירוס לקבוצה מסוימת או משתמשים מסוימים, היות ואלו אינם מודעים לעובדה שהמפתח נמצא בידם. בכל מערכת אחרת שאינה כוללת את המפתח, פיענוח ההצפנה לא יצלח ולכן הקוד לא יעבוד, לדוגמה, שימוש בשם המחשב (Hostname) כמפתח להצפנה יאפשר את מיקוד ההתקפה על מחשב מסויים [13].

- התגנבות (Stealth) – בשיטה זו משתמש הווירוס בטכניקות על מנת להסתיר את הימצאותו במערכת. לדוגמה, הווירוס יכול לשנות, בין היתר, קבצי מערכת שונים כגון פקודות ls ואחרות על מנת להסתיר את הימצאותו ולהציג תמונה של המערכת, כפי שהיא נראית לא נגועה. בדרך כלל שיטה זו נמצאת בשימוש על ידי ווירוסים השוהים בזיכרון [14 p. 5.2.1].

- שריון (Armoring) – הרעיון בבסיס שיטה זו הוא להגן על הווירוס בפני בחינה על ידי מומחי אבטחה או תוכנות אנטי ווירוס לצורך אנליזה של התנהגות ומבנה הווירוס. מחבר הווירוס משתמש בטכניקות מיוחדות למנוע את פירוקו של הווירוס לגורמים ומעקב אחר סט הפקודות שלו ועל ידי כך, למנוע את ניתוח התנהגותו. אנאליזה של ווירוס עשויה להיות דינאמית במהותה, למשל על ידי הרצת הקוד במנפה שגיאות (Debugger) או סטטית, על ידי פירוק וניתוח הפקודות. דוגמה לשיריון במקרה של אנאליזה דינאמית היא כאשר הקוד מזהה שהוא מורץ בתכנת ניפוי, למשל על ידי זיהוי נקודות עצירה, ונמנע מהתנהגות חשודה. במקרה של אנאליזה סטאטית למשל, ניתן לכתוב קוד העושה שימוש בבעיות קשות לפתרון, על מנת למנוע אוטומציה של פירוק וניתוח הקוד. דרך נוספת היא למנוע את זמינות כל הקוד בעת הפירוק.
דרך נוספת של הווירוס להתמודדות מול יישומי האנטי-ווירוס היא לגרום לתוכנת ההגנה להאמין שמיקומו של הווירוס שונה ממיקומו המעשי. דוגמה לווירוס מסוג זה הוא ווירוס ה Whale מ 1990 על מערכת ההפעלה DOS. ווירוס זה נחשב לווירוס המורכב ביותר על מערכת הפעלה זו [15]. בין היתר ידוע כי הווירוס גרם להגדלת גודל הקובץ בדיוק ב 9,216 ביטים אולם הסתיר זאת על ידי שינוי שהכניס לפקודות הספרייה של DOS.

- התחפרות (Tunneling) – אחת השיטות בה, משתמשים כותבי תוכנות האנטי ווירוס לצורך זיהוי ווירוס בזיכרון המחשב, היא כתיבת תוכנית שתפקידה ליירט קריאות למערכת הפסיקות (Interrupt) של מערכת ההפעלה לצורך אבחון פעילות חשודה שעשויה להצביע על המצאות ווירוס בזיכרון [13]. הדרך בה כותבי הווירוסים, המיועדים לריצה בזיכרון המחשב, מתמודדים מול צורת הגנה זו היא על ידי כתיבת ווירוס שממקם את עצמו ראשון בשרשרת הקריאות למערכת הפסיקות מתחת לתוכנית האנטי ווירוס, או כל תוכנית אחרת, שהקריאות למערכת זו מבוצעות ישירות על ידו, תוך מעקף של כל קריאות המערכת (System Calls) המנוטרות [14 p. 6.1]. פעולה זו נקראת התחפרות. כמובן שגם ווירוסים שאינם שוכנים בזיכרון יכולים לאתר את הטיפול בפסיקה ישירות אולם תכונה זו אופיינית בעיקר לווירוסים הנמצאים בזיכרון. המעניין בשיטה זו הוא הסימטריה בין שיטת האיום וההגנה בפניו [13]. תוכנות אנטי ווירוס מתקדמות מתימרות לאתר פעילות כזו ולהתקין עצמם ברמה נמוכה יותר מתחת לווירוס, עובדה העשויה לגרום למלחמת התחפרות ולעיתים, לחוסר תפקוד של מערכת ההפעלה.

איתור ווירוס על ידי תוכנות הוא קרב מתמשך, שסופו טרם נראה. בין השיטות המסורתיות ניתן לראות זיהוי דפוסי קוד, דפוסי התנהגות, בדיקת מספרי ביקורת (Checksums) וכדומה [16]. כותבי תוכנות האנטי ווירוס מנסים לשלב הגנות בפני כל טכניקות ההתקפה המוכרות וגם שילוב בין טכניקות שונות. באופן כללי ההתמודדות מול הטכניקות הוותיקות כהצפנה, רב צורתיות והתגנבות נוחלת הצלחה רבה, אולם הטכניקות המתקדמות, הידועות ואלו שלא, כולל השילוב ביניהם בווירוס אחד מהווים אתגר גדול יותר בפני כותבי האנטי ווירוס וקרב זה טרם הוכרע, אם בכלל.

2.3.3 תולעים (Worms)

לתולעת ולווירוס מספר מאפיינים דומים. ייחודה של התולעת לעומת הווירוס הוא, בעוד שהווירוס מתרכז בהדבקת המחשב בו הוא נמצא, התולעת מסוגלת לשכפל עצמה בצורה מושלמת ממחשב למחשב דרך הרשת. כמו כן, התולעת היא לרוב תוכנית עצמאית לחלוטין שאינה תלויה בתוכניות עזר או תמיכה ממערכת המחשב ומסוגלת לפעול, בדרך כלל, ללא צורך בכל התערבות חיצונית [13]. העובדה שתולעת מסוגלת להפיץ עצמה בצורה מהירה בצורה משמעותית לעומת ווירוס רגיל, היא שהפכה את התולעים לשיטת התקפה פופולארית ביותר בקרב התוקפים. תולעים מנצלים, בדרך כלל, חולשות ידועות במערכות הפעלה על מנת לחדור למערכת [11].

ניתן לחלק את תקופת ההפצה של תולעת ממוצעת לשלושה חלקים. בשלב הראשון, היא תפיץ עצמה בקצב מתון למחשבים פגיעים בלבד, אולם בשלב הבא נהיה עדים להפצה בקצב מהיר ביותר עד שכמות המחשבים הנגועים תגיע לרוויה. בשלב האחרון לא נצפה בגידול משמעותי בכמות המחשבים הנגועים [13].

בעבר, היה נהוג לדון בקצב הפצה של זדונה במונחים של שבועות ולעיתים אף חודשים. זהו אינו המקרה כיום. בבואנו לדון בקצב הפצה של תולעים, כבר נהוג לדבר במונחים של ימים ואף שעות. לדוגמה, תולעת Code Red משנת 2001 הופצה והדביקה 300,000 מחשבים תוך 14 שעות בלבד [10].

קיימות דוגמאות קיצוניות של תולעים שקבעו שיאים חדשים בקצב ההדבקה. תולעים מסוג זה נקראות Fast Burners היות והן מפיצות את עצמן בקצב המהיר ביותר שהן יכולות. לדוגמה התולעת Warhol Worm המסוגלת להדביק אוכלוסיה פגיעה של מחשבים תוך 15 דקות (ומכאן נלקח שמה, על פי אמרתו המפורסמת של אנדי וורהול על 15 דקות התהילה להם יזכה כל אחד). אף מהירה ממנה היא תולעת מסוג Flash Worm המסוגלת להדביק אוכלוסיית מחשבים פגיעה במספר שניות [13].

אחת מצורות האפיון של תולעים היא לפי הנשא המשמש להפצתן. תולעת המשתמשת במסרים מיידיים תיקרא IM Worm (Instant Messaging), בעוד תולעת המשתמשת בדואר אלקטרוני תיקרא email Worm. תולעים רבות מגיעות בצורה של קובץ מצורף לדואר אלקטרוני ובמקרים אלו נעשה שימוש בטכניקות הנדסת אנוש שונות על מנת להפיל את המקבל בפח ולגרום לו לפתוח את הקובץ. במקרה מיוחד זה, שני סוגי תולעים, Mailer ו – Mass-Mailer, עושים שימוש בתשתית הדואר האלקטרוני לצורך הפצת עצמם. תולעת מסוג Mailer תצמיד עצמה להודעת דואר אלקטרוני שתישלח מהמחשב הנגוע בעוד שתולעת מזן Mass-Mailer תשלח עצמה לרשימת תפוצה מרשימת הכתובות במחשב הנגוע ומכאן נובע שהפצתה תהיה מהירה בצורה משמעותית. בשני טיפוסים אלו נדרשת פעולה של המשתמש במחשב הנגוע על מנת להפעילן בניגוד לנאמר לעיל כמאפיין ווירוס תולעת [14 p. 2.3.2.1].

לחילופין, דרך נוספת להפעלה ללא צורך בהתערבות המקבל היא על ידי שימוש בטכניקת גלישת חוצץ (Buffer Overrun) ועל כך בהמשך [13].

תולעים יכולים להשתמש בכל צורות ההסוואה המתוארות בהקשר לווירוסים, הצפנה, רב צורתיות, שינוי צורה וכדומה. התולעת מורכבת משלושה חלקים עיקריים: מנגנון ההתקפה, המשמש להשתלטות ראשונית על המחשב הפגיע. המנגנון הביצועי (Payload) שאחראי להשיג את התוצאה הנדרשת מהתולעת. מנגנון ההפצה שתפקידו לאתר את המטרות הבאות בעזרת משאבים הנמצאים על המחשב הנגוע [10].

בדומה למקורם של וירוסים רבים, גם התולעים הראשונות שהופיעו לא נכתבו במטרה לפגוע בתשתיות אלא כתוכנית לגיטימית שנועדה להשיג מטרה מסוימת. לדוגמה תולעת שנכתבה במעבדות PARC של חברת זירוקס (Xerox) בשנת 1982, נועדה במקור לנצל כוח חישוב לא מנוצל במעבדים של מחשבים שונים ברשת במטרה להשיג כוח מחשוב מבוזר. התוצאה הייתה, למרבה המבוכה, 100 מחשבים שונים מתים ברחבי הבניין, כדברי עובדים של זירוקס [13].

תולעת נוספת בעלת משמעות היסטורית היה תולעת האינטרנט (Internet Worm) משנת 1988. ההתקפה במקרה זה הורכבה ממספר שלבים, כשבכל שלב נעשה שימוש בטכניקות שונות של התקפה. בשלב הראשון נעשה שימוש בנקודות תורפה של תכנית ה Sendmail הותיקה ששמשה כסוכן למשלוח הודעות דואר אלקטרוני, בשילוב עם טכניקת גלישת חוצץ על תכנית ה Finger [14 p. 10.4.1], המשמשת לבירור מידע על משתמשים במערכת. בחלק האחרון של שלב זה השתמשה התולעת בתכנית rexec או rsh, המאפשרות גישה מרחוק לתכנית המעטפת (Shell) של המחשב, תוך ניסיון ניחוש סיסמת המשתמש. בשלב השני, לאחר רכישת השליטה, התולעת הפעילה פקודות ליצירת, הידור והרצה של תכנית C קטנה במחשב הנגוע. התוכנית שימשה לצורך משיכת תכניות ביצועיות למחשב. בשלב השלישי של ההתקפה, התולעת התמקמה סופית במחשב הנגוע ויכלה להפעיל את תהליך הפצתה. המידע על מחשבים חדשים להדבקה נלקח מנתוני המחשב הנגוע, ממשקי וטבלאות ניתוב הרשת, או קבצים שונים שהכילו שמות של מחשבים אחרים ברשת. תולעת זו, למרבה המזל, לא כללה חלק הרסני והנזק היחידי שגרמה, נבע אך ורק מהעומס שהטילה על מערכת המחשב כתוצאה מהשימוש הגובר והולך במשאביו [13].

לסיכום, תולעים נועדו במקרים רבים לגרום לבזבוז משאבי מחשוב ומשאבי רשת. במקרים אחרים המטרה היא הפעלת מתקפת DDoS (מתקפת מניעת שירות מבוזרת, על כך בהמשך) או התקנת דלת אחורית במערכת. הנזק, לו מסוגלת תולעת לגרום, גדול לעין ערוך מהנזק הנגרם על ידי ווירוס. לדוגמה, הנזק שנגרם מהתולעים Code Red ו – Nimba מוערך בכ – 3.25 מיליארד דולר [10]. שני סוגי התולעים העיקריים הם תולעים המבוססים על תשתיות ושירותי הרשת ותולעים המבוססים על תשתיות שרתי הדואר האלקטרוני [11].

2.3.4 רוגלות (Spyware)

רוגלה היא תוכנה שמטרתה לאסוף מידע אישי או סודי מהמחשב הנגוע ולהעבירו לצד שלישי, כך זאת ללא ידיעת המשתמש במחשב וכמובן ללא אישורו.

המידע שרוגלה יכולה לאסוף ממחשב הלקוח יכול להיות מכל סוג שהוא וכולל בין היתר [13]:

- שמות משתמש וסיסמאות היכולים להיאסף מקובץ על המכונה הנגועה או מהקלטת הקשות המשתמש (Keylogger).
- כתובות דואר אלקטרוני, היכולות למשל, להיות בעלות ערך רב למפיצי דואר זבל.
- נתוני חשבונות בנק וכרטיסי אשראי, היכולים לשמש לצורך גניבת כסף ומרמה.
- נתוני מפתחות ורישיונות של תוכנות המותקנות על המחשב הנגוע, במטרה ליצור ולהפיץ תוכנה פיראטית.

ווירוסים ותולעים יכולים למעשה לבצע את אותה הפעילות כרוגלות אולם אינם נחשבים ככאלו מעצם הגדרתם וזאת כיון שרוגלות אינן יכולות לשכפל ולהפיץ עצמם, או להדביק קבצים אחרים [13]. הרוגלה היא, מעצם הגדרתה, תוכנה בעלת מטרה מוצהרת וברורה. כל פעילות נוספת שתבצע תגביר את הסיכוי לגילויה.

לרוגלה מספר דרכים להגיע למחשב נגוע. לדוגמה, דרך תוכנה שהותקנה במחשב וכוללת את קוד הרוגלה, או בעזרת ניצול חולשות של דפדפן האינטרנט, המאפשרות לעיתים את הפעלת הרוגלה אך ורק כתוצאה מגלישת המשתמש לאתר כלשהו.

דרך נוספת להתקנת הרוגלה על המחשב הנגוע היא על ידי תכנון חומרה שתבצע את פעילות הריגול. אכן בשנתיים האחרונות נפוצו שמועות על רכיבים תוצרת סין שכללו בתוכם רכיבי רוגלה אשר נוצלו לריגול תעשייתי או מדיני, עד שמקור במשרד ההגנה האמריקני ציין שלאור הסיכון הקיים, לא תהיה להם ברירה אלא להימנע מלרכוש חומרה מתוצרת סינית. לאחרונה אף נמצא ציוד רשתות שיוצר כזיוף של ציוד מתוצרת חברת סיסקו, עובדה שלא תרמה להרגעת הרוחות [1 p. 27].

לאחרונה אנו עדים לשימוש ברוגלות על ידי, או בתמיכת ממשלות, אף שקשה מאוד להוכיח מעורבות ממשלתית. לדוגמה, במאי 2008, טען השבועון דר-שפיגל כי שירות הביון הגרמני השתמש ברוגלות לצורך מעקב אחר משרד המסחר והתעשייה האפגאני. בספטמבר 2008, האשימו גורמים בסיאול כי גורמים מצפון קוריאה גנבו מסמכים צבאיים מסווגים מאנשי צבא בעזרת רוגלות [4].

בשנים האחרונות, חלה עליה תלולה בשימוש ברוגלות, לעומת השימוש בווירוסים או תולעים. משתמשי מחשב רבים, במיוחד כאלו שמחוברים לאינטרנט, חשים באיטיות גוברת והולכת של המחשב, עד לשלב שבו המחשב אינו שמיש מבחינה מעשית, למרות שאינם מבחינים בשינוי כלשהו במראה המחשב. התופעה נובעת מהעובדה שרוגלות גוזלות כוח מחשוב ומשאבים שונים מהמחשב במטרה לאסוף ולשלוח נתונים [10].

2.3.5 סוסים טרויאניים (Trojan Horses)

בדומה לסיפור המקורי של סוס העץ, ששימש כבסיס להשתלטות היוונים על טרויה מבפנים, כך גם בתחום מערכות המידע והמחשוב, נמצא את תוכנית הסוס הטרויאני שתפקידה לתקוף את מערכת המחשב מבפנים, מבלי לעורר חשד בליבו של המשתמש במחשב. הסוס הטרויאני מתיימר להיות תוכנית לגיטימית, המבצעת פעילות חוקית ורצויה במחשב, אולם מאחורי הקלעים, התוכנית מבצעת פעילות עוינת כגניבת נתוני זהות, נתונים מסחריים, זריעת הרס או כל פעילות זדונית.

כאשר תוכנית סוס טרויאני חודרת למחשב, היא מחליפה בשלב הראשון תכניות מפתח במחשב הנגוע, כקבציי מערכת ושירותים. בעת הפעלתם, תכניות אלו מבצעות את זממם בנוסף לפעילות הלגיטימית המצופה מהם.

דוגמה קלאסית לסוס טרויאני היא תכנית המציגה מסך הזדהות מדומה, מקבלת את הקלט מהמשתמש, מעבירה אותו למפעיל הסוס ומציגה הודעת סיסמה שגויה לפני העברת השליטה למסך ההזדהות המקורי. דוגמה נוספת היא החלפת ספריה דינאמית (DLL) של מערכת ההפעלה Windows בגרסת סוס טרויאני אשר תבצע את כל הנדרש מהספרייה המקורית ובנוסף תבצע פעילות כגון גניבת נתוני כרטיס אשראי.

באמצע שנות ה - 2000 הופיעו בתחום כלי תוכנה וחומרה שמטרתם הייתה ללכוד רצף של הקשות מקלדת במחשב הנגוע (Keylogger) והעברתם דרך האינטרנט בערוצים חבויים למטרתם. במקרה זה, ניתן לסווג את התוכנה גם כרוגלה (Spyware). השימוש בכלים אלו, ששימשו בעיקר למטרות ריגול, תעשייתי או מדיני, הפך להיות כה נפוץ, עד שהמחלקה לביטחון פנים בארה"ב (Homeland Security) הפיצה אזהרה בדצמבר 2005, בה כינתה את הסוסים הטרואיאניים שהופצו דרך רשתות בעזרת מערכת הדואר האלקטרוני כאיום הגדול ביותר על משתמשי המחשב ברשת [1 p. 24].

מקרה מפורסם הקשור בסוס טרויאני, אירע בישראל. המחבר אמנון ג'קונט גילה חלקים מספרו החדש באינטרנט, עוד לפני שזה התפרסם. בשלב הבא נעשה ניסיון לגנוב כסף מחשבונו בבנק, עובדה שעוררה את חשדו בחתנו לשעבר, מיכאל האפרתי. חקירת משטרה הובילה לגילוי סוס טרויאני מסוג Keylogger שהותקן במחשבו. חשיפה זו חשפה בעקבותיה מסכת מסועפת של ריגול תעשייתי שעירבה בכירים רבים בתעשייה. מסתבר שהסוס הטרויאני הגיע למחשב כקובץ מצורף לדואר אלקטרוני אשר, ברגע שהתקין עצמו, שלח מידע מהמחשב לשרת שמוקם בלונדון [1 p. 25].

ולסיכום, סוס טרויאני מהווה דרך יעילה לבצע פעולות על מחשב נגוע. המטרה שלשמה הוא מותקן יכולה להיות מעקב (לגיטימי?) של מעביד על עובדיו, ריגול, הפעלת פצצת זמן, התקפות מניעת שירות (על כל אילו, בהמשך) ומטרות רבות אחרות.

2.3.6 תוכנות פרסום (Adware)

חברות מסחריות רבות מעוניינות במידע על הרגלי הגלישה של משתמשים. מידע המעניין את החברות במיוחד הוא סוג המוצרים בהם המשתמש עשוי להיות מעוניין במטרה לכוון לו פרסומות הקולעות לצרכיו.

דרך להתמודד עם אתגר ניתוח צרכי המשתמש היא על ידי התקנת תוכנות עזר קטנות (Adware), האוספות ומנתחות מידע על דפוס הגלישה של המשתמש וגוזרות ממנו ידע על דברים שעשויים לעניין אותו, החל ממוצרי צריכה ועד אתרי מידע. התוכנות משתמשות במידע זה על מנת להציג למשתמש פרסומות הנוגעות לסוג המידע שיכול לעניין אותו. תוכנות מסוימות אף ירחיקו לכת ויגרמו לדפדפן לגלוש לדף מסוים, בניגוד לרצונו של המשתמש [13].

במקרים רבים תוכנות אלו נכתבות בצורה לגיטימית לחלוטין וללא כוונה זדונית, אולם עובדה זו אינה גורעת מהנזק אותו הן מסוגלות לגרום. העובדה היא שמשתמשים רבים אינם מודעים כלל לעובדה שבמחשב שלהם הותקנה תוכנה מסוג זה, כמו כן, השימוש בתוכנה מעלה שאלות אתיות רבות הנוגעות לשמירה על פרטיות המשתמש וחשיפת מידע אישי לגביו לצרכים מסחריים. בעיה נוספת נובעת מאיכות הקוד של התוכנות. היות ותוכנות פרסום באות למעשה לענות על צרכי החברה המסחרית ולאו דווקא על צרכי המשתמש, איכות הקוד של תוכנות רבות מסוג זה נמוכה ביותר והתוכנות צורכות משאבי מחשב רבים, ללא התחשבות בצרכי המשתמש, עובדה הגורמת להאטה משמעותית בביצועי מערכות מחשב נגועות רבות, עד לקריסתן המלאה.

מבחינות מסוימות ניתן להשוות את אופי הפעילות של רוגלות ושל תוכנות פרסום. שתיהן מותקנות לרוב, ללא ידיעת המשתמש, אוספות מידע עליו ועל המנהגים שלו ושולחות מידע זה למקור חיצוני [13]. העובדה שתוכנות אלו פועלות בצורה לגיטימית לכאורה, מקשה על ההתמודדות עימן. תוכנות רבות מתקינות עצמן כחלק בלתי נפרד מתוכנות אחרות, המספקות שירותים רצויים למשתמש (כגון תוכנות שיתוף קבצים למיניהן) ובעת התקנתן, המשתמש אף מאשר (לרוב בלי לקרוא את ההסכם) את הרישיון המתיר את ההתקנה. הסרתן תגרום לרוב להפסקת תפקודה של התוכנה הרצויה. בשל העובדה שבמקרים רבים ההתקנה חוקית ובאישור המשתמש (גם אם ללא ידיעתו), חברות המתקינות תוכנות אלו גוררות חברות אנטי ווירוס לבתי המשפט בטענה שפעילותן למניעה אינה חוקית במקרים אלו. מסיבה זו חברות אנטי ווירוס רבות נמנעות מלהפעיל את ההגנה בפני תוכנות פרסום כברירת מחדל.

2.3.7 תוכנות ורשתות רובוטיות (Bots & Botnets)

כפי שראינו בסעיפים הקודמים, במחשבים נגועים, עשויה זדונה לרוץ באין מפריע ולבצע פעילות בלא ידיעת המשתמש. השימוש יכול להיות למטרות רבות כגניבת נתונים, ניתוח פעילות המשתמש וכל פעילות לא חוקית אחרת. תוכנות מסוימות עשויות להיות מותקנות במחשב, בלא שתבצענה אף פעילות, למעט האזנה לערוצי תקשורת חיצוניים, כגון פרוטוקולים שונים המשמשים להידברות דרך האינטרנט (לדוגמה, IRC – Internet Relay Chat). מתקין התוכנה יכול, על ידי התקשרות מרחוק, להעיר את התוכנה ולגרום לה לבצע פעילות כלשהי. לאור צורת פעולתם, תוכנות מסוג זה מכונות בשם Bot כקיצור של רובוט (Robot). כינוי מקובל נוסף ל Bot הוא זומבי (Zombies) [13]. ניתן למצוא שימושים רבים בתוכנת Bot, אולם, קיימות לה שתי מטרות עיקריות, הפצת דואר זבל והתקפות מניעת שירות.

כיום, במספר מדינות רב, משלוח דואר זבל אינו חוקי בהתאם לחוקי המדינה או תקנות של ספקי שרותי אינטרנט. אתרים הידועים במשלוח דואר זבל מוכנסים לרשימה שחורה כך שמנגנוני הגנה ימנעו קבלת דואר מהם. מסיבה זו, מפיצי דואר זבל נמנעים מלשלוח דואר ישירות לנמענים ומעדיפים לנקוט בדרכים עקיפות. כאן באות תוכנות ה Bot לעזרתם. התוכנות משמשות להעברת דואר הזבל מהמקור המוקצה, דרך מקור לגיטימי לכאורה, הוא המחשב הנגוע, למטרתו כך שלא ניתן לזהות את אופי הדואר האלקטרוני על פי שולחו.

השימוש השני העיקרי הוא ביצירת התקפות מניעת שירות (DDoS – Distributed Denial of Service). מניעת שירות היא אחת הדרכים האפקטיביות לפגוע בספק שירות. הרעיון הוא לגרום עומס רב על ספק השירות, אם על ידי יצירת עומס על הרשת שלו מעבר לקיבולתה, או על ידי משלוח מספר רב של בקשות שירות לגיטימיות מעבר לכמות אותה יכולה מערכת הנחשב לשאת. כמובן שניסיון לבצע התקפה מסוג זה ממקור יחיד נועד מראש לכישלון, אם בגלל הכמות המוגבלת של המשאבים שמחשב אחד יכול לצרוך ואם בגלל שברגע שמזוהה מחשב המנסה לבצע התקפת DoS, ניתן לחסום את גישתו.

במקרה של שימוש ב Bot, ההאקר השולט מפעיל בעת ובעונה אחת את כך המחשבים הנמצאים תחת שליטתו, וגורם לכולם לפנות לאתר או המערכת, אותה ברצונו לתקוף, ולגרום לקריסתה בתוך זמן קצר [11]. הכינוי לרשת Bot מסוג זה הוא Botnet.

דוגמה להתקפת DDoS אירעה בשנת 2000, כאשר אתר יאהו (Yahoo.com), אמזון (Amazon.com), איביי (eBay.com), סי אן אן (CNN.com) ואחרים סבלו מהתקפה שמנעה את שרותיהם למספר שעות וגרמה להפסד כספי רב. חקירה של המשטרה הפדראלית בארה"ב (FBI) חשפה נער בן 15 ממונטריאול שיזם את ההתקפה תוך שימוש במודם לשליטה והפעלת ה Bot במחשבים הנגועים לצורך הפעלת ההתקפה בו זמנית מכולם [1 p. 39].

2.3.8 ניצול גלישת חוצץ (Buffer Overflow Exploits)

ניצול גלישת חוצץ הוא אחד מאותם איומים המנצלים חולשות תוכנה ומהוות את אחת הבעיות הקשות ביותר בתחום התוכנה כיום. לתוכנה הרצה בזיכרון המחשב מוקצה מקום בגודל מוגדר מראש לאכסון מידע בצורות שונות. המקום יכול להיות המחסנית (Stack), הערימה (Heap), או אפילו מקום המוקצה למחרוזת בזיכרון. התוקף עשוי לכתוב, בצורה ישירה או לא, לתוך מקום כזה כמות מידע שתגרום לגלישת חלק ממנו מעבר לשטח המוקצה ודריסת קוד אחר בזיכרון. במחשב מפעיל את הקוד הדורס, כאילו היה תוכנית רגילה, כך שקוד זה יכול לבצע פעילות שהקוד המקורי לא היה מתוכנן לה כלל. הקוד הדורס בדרך כלל ינסה לפתוח מעטפת משתמש חדשה (User Shell, ומכאן הכינוי לעיתים לקוד זה כ - shellcode) שתקבל את ההרשאות של הקוד המקורי שנדרס, ותבצע פעילויות לא חוקיות תחת הרשאות אלו [13].

בעיקרון, תוכנה מסוגלת לוודא את גודל המידע שנרשם לזיכרון ולהחזיר שגיאה במקרה וגודל זה חורג מהמקום המוקצה. השאלה המתבקשת היא מדוע גבולות המקום בזיכרון אינן נבדקות. אכן, בשפות ומערכות הפעלה מודרניות קיימת התייחסות רצינית יותר לחולשה זו אולם בשפות כגון C, אין עדיין בדיקה אוטומטית לחריגה מהמגבלות וגם בשפות אחרות, בהם קיימת הבדיקה, ניתן למצוא תקלות תוכנה (Bugs) המנוצלות לסוג התקפה זה.

התקפות מסוג ניצול גלישת חוצץ מסוכנות במיוחד מהסיבות הבאות [10]:

- אחוז ניכר מהתוכנות סובל מהחולשה של אי בדיקת מגבלות החוצץ בעת הרישום בזיכרון. קיים ידע על תוכנות רבות החשופות להתקפה ותוכנות רבות נוספות מתגלות מידי יום. מעל ל 50 אחוז מההכרזות של צוות החירום של האינטרנט, העוסק בבעיות ברשת כפריצת מערכי אבטחת מידע (CERT – Computer Emergency Response Team), נוגעות לבעיות הנובעות מניצול גלישת חוצץ.
- תכנון התקפה המנצלת סוג חולשה זו קל ביותר ואינו דורש ידע מיוחד. כל מתכנת בעל רקע בסיסי יחסית יכול להוריד מהרשת קוד המאפשר את ניצול גלישת חוצץ בתוספת מתכון קל לשחזור לצורך כתיבת קוד יעודי.

- התקפת ניצול גלישת חוצץ חזקה ביותר ולעיתים רבות, הקוד העוין מקבל הרשאות של מנהל מערכת, כך שגישתו למקומות רגישים במיוחד פתוחה.

דוגמה בולטת לקוד שניצל חולשה זו הוא תולעת ה Code Red משנת 2001, שניצלה חולשת גלישת חוצץ ממנה סבל שרת המידע של חברת מיקרוסופט (IIS – Internet Information Server) אשר אפשרה לבצע קוד בתוך השרת לצורך הפצת התולעת [10].

Rootkits 2.3.9

ערכה המכילה תכנית או מקבץ של תכניות רבות שמטרתן להחליף ולשנות את תפקודם הרגיל של תכניות מערכת הפעלה שונות מכונה RootKit (או בעברית ערכת Root). ההשראה לשם מגיעה מכינויו של המשתמש Root, בעל רמת ההרשאות הגבוהה ביותר במערכות ההפעלה UNIX או Linux. במערכות הפעלה אלו לא נדיר לראות ערכה שמחליפה עשרות תכניות עזר של המערכת, כגון תכניות cp, mv, diff ואחרות. במערכת ההפעלה חלונות לדוגמה, תוכניות המגיעות עם הערכה יכולות להחליף תוכניות של מערכת ההפעלה הרצות בזיכרון המחשב, ולעקוף קריאות למערכת ההפעלה [11].

מטרתה של ערכת Root היא בעיקר הסתרת נוכחותה ושל תוכניות אחרות, עליהן מיועדת הערכה להגן. בדרך כלל, תוכנית מהערכה תבצע את הפעולה לה מצפה המשתמש מהתכנית המקורית אולם, בהיחבא, תבצע פעילות נוספת. לדוגמה, הפקודה ls, שמטרתה להציג רשימת קבצים וספריות במערכות UNIX ו Linux יכולה לעשות זאת, פרט להצגת קבצים מסוימים המהווים חלק מתכנית Backdoor אותה השתיל ההאקר במחשב הנגוע.

שיטת הפעולה של ערכת ה Root, המגלמת בתוכה הגנה עצמית, מקשה ביותר את גילוייה וקיים חשש כי מערכות מחשב רבות מכילות ערכה ללא ידיעת המשתמש במשך זמן רב. בדרך כלל אין לערכה זכות קיום בפני עצמה והיא מצורפת לתוכניות עוינות אחרות כדלת אחורית או Keylogger ומשמשת כמנגנון הגנה והסתרה [11].

ערכות Root רבות קיימות להורדה ברשת. בין הערכות הנמצאות המוכרות נמצא את LRK5, Knark ו Adore. ערכה נוספת, נפוצה במיוחד, היא ה Hacker Defender. דוגמה לפעולת הסתרה שמבצעת האחרונה הוא במאבק מול תוכנת גילוי ה Rootkit, RootRevealer מחברת מיקרוסופט. תוכנת הגילוי של מיקרוסופט מבצעת השוואה בין חתימת הקבצים של מערכת ההפעלה, כפי שיצאו משערי מיקרוסופט, לבין אלו הנמצאים בפועל במחשב. ערכת ה Hacker Defender מסירה הבדלים אלו וגורמת למצג שווא, כאילו לא בוצע שינוי בקבצי מערכת ההפעלה.

2.3.10 ניצול חולשות HTTP (HTTP Exploits)

אחת מההגנות המסורתיות על מערכות מחשב ארגוניות (וכיום גם ביתיות) היא חומת אש (Firewall). פעולת חומת האש הבסיסית היא לחסום או לסנן את התנועה מהארגון לרשת ובחזרה, למעט התנועה שמתבססת על פרוטוקול HTTP (Hyper Text Transfer Protocol), בדרך כלל דרך פורט 80, המשמש בעיקר לתעבורה מדפדפן האינטרנט החוצה לרשת האינטרנט. העובדה שהתעבורה דרך פרוטוקול HTTP נעה כמעט באין מפריע פותחת למעשה קו ישיר בין המתקיף לבין שרת האינטרנט (Web Server), כך שאם המתקיף יוכל להשפיע על השרת לפעול כרצונו, יוכל לבצע פעילויות ולהשתמש במשאבים, אשר בכל מקרה אחר היו בלתי זמינים [10].

פרוטוקול HTTP מורכב ביותר וקיימות דרכים רבות לנצל את חולשותיו, או החולשות המלוות את דפדפן האינטרנט ו/או השרת. בין השיטות הנפוצות המוכרות ניתן למנות:

- הזרקת SQL (SQL Injection) – שיטה בה מוסיפים קוד SQL מוסתר לכתובת ה URL ומנצלים את העובדה שהשורה אינה נבדקת לתקינות על ידי השרת כך שהקוד יכול לעבור המרה לפקודת SQL שתגיע לבסיס הנתונים בשרת ותבצע המרת נתונים כרצון התוקף.
- המרת פרמטרים (Parameter Tampering) - שיטה בה מנצלים המרה של פרמטרים שעוברים בין השרת והלקוח דרך הפרוטוקול במטרה לשנות נתונים בשרת כגון נתוני התחברות, הרשאות גישה מחירים וכמויות של מוצרים וכדומה.
- הרעלת עוגיות (Cookie Poisoning) – שיטה המשמשת להתחזות ופגיעה בסודיות של לקוח על ידי מניפולציה על העוגייה המחזיקה את נתוני ההתחברות העכשוויים של המתחבר.
- Cross Site Scripting Attack – בשיטת התקפה זו, מוכנס קטע של קוד לשורת ה URL הגורם לדפדפן לבצע אותו ולגרום לשינויים בדף המקורי שאמור להיות מוצג, לאסוף נתונים ממנו ולשלוח אותם לאתר שלישי. לדוגמה:

http://www.myWeb.com/find.asp?SearchStr=<script>alert("You Have Been Hacked")</script>

יגרום להודעה *You Have Been Hacked* להופיע כהודעה למשתמש.

ושיטות רבות אחרות.

דוגמה להזרקת SQL יכולה להיות למשל לצורך התחברות לשרת. שורת ה URL לצורך התחברות אמורה להיראות לדוגמה כך:

http://www.anyWebServer.com/find.asp?Login=Admin&password=123456

עבור משתמש Admin בעל הסיסמה 123456. אולם אם השורה שתישלח תהיה:

http://www. anyWebServer.com/find.asp? Login=Admin&password= 123456 ' OR 1=1

הדבר יוסיף את קטע הקוד *OR 1=1* לשורת ה SQL, והיות ש 1=1 תמיד נכון, יוכל המשתמש להתחבר באין מפריע לשרת.

דוגמה נוספת לשיטה שנפוצה לאחרונה היא ביצוע מעבר לספריות מוגנות בשרת. דרך בעזרתה ניתן להגיע לספריות הנמצאות מחוץ לתחום היא על ידי העברת מחרוזות כ - \..\..\ המאפשרות הגדרות מעבר יחסיות לנקודה בה נמצאים בשרת כחלק מכתובת ה URL. השרתים כיום חוסמים את האפשרות לקבל כתובת URL הכוללת קטעי מחרוזת כמתואר. התוקפים עוקפים הגנה זו על ידי שימוש בקידוד הקסאדצימלי בשילוב עם קידוד Unicode על מנת לייצג את מחרוזות ה \..\. [10].

כיום ניתן למצוא באינטרנט ערכות כלים רבות לניצול חולשות HTTP המאפשרות למשתמש הלא מיומן לבצע פעולות כהפעלת תוכניות בשרת, שינוי נתוני מערכת, גישה למפתחות רג'יסטרי ופעולות זדוניות אחרות.

2.3.11 ניצול הרשאות משתמש (Privilege Escalation exploits)

בשיטה זו מנצל התוקף חולשות במערכת המחשב על מנת להגדיל את ההרשאות שלו למערכת. הגדלת רמת ההרשאה של התוקף מאפשרת לו לגשת למשאבי מערכת ואזורים החסומים בפניו בדרך כלל. לדוגמה, בכל מערכות ההפעלה חלונות NT וחלונות 2000 קיים משתמש כברירת מחדל בשם Guest. חשבון זה, כברירת מחדל, אינו כלל הגנת סיסמה כלל, כך שכל אחד יכול להתחבר למערכת בעזרת שם משתמש זה, ולהשתמש בחולשות ידועות של הגנה על הרשאות המשתמש על מנת להגדיל את הרשאותיו [10].

דוגמה לניצול חולשה כזו היא על ידי תוכנה שנקראת GetAdmin המאפשרת למתחבר למערכת במחשב בעל מערכות ההפעלה המצוינות מעלה לקבל הרשאות מנהל מערכת (Administrator) למרות שהתחבר כמשתמש בעל רמת הרשאה נמוכה יותר. חוץ מחולשה זו, קיימות פרצות נוספות רבות הקשורות להרשאות משתמש. תוכנות כ HackDLL ואחרות מאוד שימושיות לפורצים ומאפשרות להם להשיג רמת גישה גבוהה ככל שיחפצו בהתחברם למערכת המחשב.

2.3.12 פצצות לוגיות (Logic Bombs)

בדיוק כפצצה רגילה, פצצה לוגית נמצאת אי שם בתוך הקוד וממתינה לאירוע מסוים שייקרה על מנת לפעול את פעולתה.

הפצצה הלוגית מורכבת שני חלקים עיקריים [13]:

- המטען - הפעולה אותה תבצע הפצצה הלוגית. זו יכולה להיות כל פעולה על המחשב, אולם בדרך כלל היא תישא אופי זדוני.
- המרעום (trigger) – אירוע במחשב או תנאי לוגי מחושב הקובע אם המטען יופעל. התנאים שייבחרו מוגבלים אך ורק על ידי דמיון המחבר, החל מתאריך, המשתמש המחובר למחשב, גרסת מערכת הפעלה או אפילו על סמך היעדר אירוע כלשהו.

פצצת זמן היא תת מחלקה של קבוצת הפצצות הלוגיות המאופיינת בכך שהאירוע הגורם להפעלתה מבוסס על זמן [1 p. 20].

פצצה לוגית עשויה להיות חלק מתוכנה לגיטימית כלשהיא או כתוכנה בפני עצמה [13]. יצירתה היא פעולה פשוטה יחסית, אולם דורשת בדרך כלל קשר ישיר לקוד ולכן מחברי הפצצה הם בדרך כלל כותבי התוכנה או המתחזקים אותה. היות וקוד תוכנה יכול להכיל עשרות אלפי עד מיליוני שורות קוד, קשה ביותר לאתר קוד פצצה, במיוחד אם הקוד שנכתב אינו עובר סקירה על ידי גורם מפקח כלשהו ומכאן שפצצה לוגית מסוגלת לגרום לנזק רב רבות לפני שמבחינים בקיומה.

לעיתים, קיימות פצצות לוגיות שנעשו בלי משים, לדוגמה באג 2000 הידוע לשמצה, או במכוון אולם לא מתוך רצון לגרום לנזק כלשהו, לדוגמה, ביצי הפסחא (Easter Eggs) הנכתבים בדרך כלל לצורך מתן הכרה לכותבי התוכנה או סתם לצורך מתן דרור לדמיונו ויצירתיותו של הכותב.

לפצצה לוגית שימושים רבים. מחיקת קבצים לאחר פיטורי עובד, גניבת חלק הסכום, אותו המחשב מעגל, בעת ביצוע פעולה בנקאית או סתם זריעת הרס. דוגמה לשימוש האחרון נמצא בפצצת זמן שהוכנסה לתוכנה שפותחה לצורך בקרה על קווי אספקת גז וזאת, באישור גורמי ממשלה בארה״ב. המוטיבציה שעמדה מאחורי הפצצה היא הידיעה כי הסובייטים יגנבו את התוכנה ויעשו בה שימוש לא חוקי. התוצאה הייתה, לפי גורמים אמריקאיים, ההתפוצצות הלא גרעינית המרשימה ביותר שנראתה עד אותה תקופה [1 p. 20]. במקרה אחר, עובד חברה השתיל פצצה זמן שגרמה למחיקת כל הקבצים ב 1000 מחשבים בחברה בה עבד. המטרה במקרה זה הייתה להרוויח מצניחת ערך המנייה של החברה [13].

עוד מקרים מפורסמים של פצצות לוגיות הם וירוס ירושלים מ 1988 שפעל בכל יום שישי ובכל ה 13 לחודש ושכפל עצמו עד שלב מסוים שבו החל גם להרוס את כל הקבצים שעל המחשב. וירוס אחר, Cascade, גרם לאותיות על המסך ליפול לשורה האחרונה בשלושת החודשים האחרונים בכל שנה. וירוס ידוע נוסף, מיכאלאנג׳לו מ 1992, גרם להריסת הספריות על הדיסק הקשיח של המחשב בכל ה 6 לחודש מרץ. ולבסוף, המקרה של המתכנת מג׳נרל דיינמיקס, אשר השאיר פצצה לוגית שמטרתה למחוק מידע חיוני על מנת שיוכל לחזור לאחר מכן ולשחזר מידע זה תמורת תשלום [1 p. 20].

Phishing 2.3.13

אחת מסוגי התרמיות בעלות קצב הגידול הגדול ביותר באינטרנט, בצורה החושפת נתונים אישיים של מיליוני גולשים באינטרנט. התרמית מערבת בדרך כלל דואר אלקטרוני בעל תוכן שיקרי, לדוגמא המציג עצמו כמופנה למקבל מהבנק שלו, המפנה בדרך כלל לאתר, לדוגמה אתר הבנק המציג מצג שווא כמסך הכנסת נתונים המאפשר לקבל נתונים אישיים ופיננסיים, כמספר חשבון בנק, סיסמה ואף פרטי כרטיס אשראי מהגולש, הבטוח שהוא נמצא באתר לגיטימי לחלוטין. מטרתם העיקרית של מפעילי תרמית מסוג זה היא השגת רווח כספי במרמה.

2.4 המאבק - לבני נגד שחורי הכובע

בעולם פשעי המחשב, הכינוי להאקרים בעלי כוונות זדוניות הוא שחורי הכובע (Black Hats) בעוד שהכינוי לאותם שומרי חוק שמטרתם להגן על מערכת המחשב בפני פגיעה הוא לבני הכובע (White Hats). קיימת קבוצה שלישית, הנמצאת בתחום האפור של החוק. קבוצה זו פועלת בדרך כלל בדרכים לא מסורתיות במטרה לחשוף פגיעות וחולשות בהגנות של ארגונים ממשלתיים ואחרים. לרוב, החשיפה היא קודם כל לציבור עוד לפני אחראי האבטחה בארגונים עצמם, עובדה המאפשרת ניסיונות התקפה תוך ניצול החולשות שנחשפו, עד לרגע בו מותקנת הגנה מתאימה. אלו נקראים באופן טבעי אפורי הכובע (Gray Hats) [1 p. 41].

הדרכים בהם לבני הכובע מתמודדים עם האתגרים הניצבים בפניהם הן רבות ומגוונות. החל מהדרכים הוותיקות והמסורתיות להתמודדות בפני ווירוסים, המשך בהתמודדות עם תוכניות הרצות בזיכרון, התמודדות עם תולעים וחיות אחרות וכלא בפיתוח מתודולוגיות לפיתוח וכתיבת מערכות מחשב בטוחות יותר. הכלי שבעזרתו נערכת ההתמודדות כנגד האיומים למערכת הוא חיישני הזדונה (Malware Detectors). חיישן זדונה הוא למעשה קוד, המממש טכניקת הגנה זו או אחרת, שמטרתו לאתר קוד אחר בעל כוונות לא טהורות, ולנטרל את פעולתו הזדונית. שחורי הכובע משתמשים בשתי טכניקות עיקריות במטרה להערים על חיישני הזדונה. טשטוש או ערפול (Obfuscation) שמטרתו לטשטש את מטרתה העיקרית של הזדונה ללא הרחבת פעולתה. שיטה שנייה מערבת שינוי או הרחבה של פעולות הזדונה הקיימת, במטרה לייצר זדונות חדשות בהתבסס על עקרונות ואף שימוש חוזר בזדונה קיימת, עובדה המשמשת כתפקיד מפתח בצורת המימוש של חלק ממנגנוני ההגנה כנגד זדונות, במיוחד אלו המתבססים על חתימות [9].

על מנת שארגון יהיה מסוגל להתמודד בפני פגעי הזדונה, בראש ובראשונה עליו לוודא שמדיניותו המוצהרת מכילה התייחסות ספציפית וממוקדת להתמודדות. אם נהלי הארגון אינם מכילים שיטות מוגדרות מראש להתמודדות, לא סביר שכל מחלקותיו יוכלו להתמודד בפני האיומים הרבים העומדים בפניהם. ההתייחסות לשיטות ההגנה והמניעה חייבת להיות כללית ככל האפשר על מנת לאפשר גמישות והתאמה לחלקים השונים של הארגון ולהימנע מהצורך בעדכונים תכופים בנהלים, אולם ספציפית מספיק על מנת להגדיר בצורה בהירה את הנדרש להגנה [11].

ניתן לחלק בהרחבה את השיטות לגילוי זדונה לשתי קטגוריות עיקריות [9]:

- גילוי המתבסס על אי נורמאליות של קוד או פעילותו בהתבסס על הידוע והמצופה מהתנהגותו או מראהו של קוד נורמאלי. שיטה זו מחולקת בדרך כלל לשני שלבים, שלב הלימוד בו החיישן לומד ומאפיין כיצד צריכה תוכנית נורמאלית להתנהג ושלב הניטור בו התוכנית בודקת קוד ומשווה את התנהגותו להתנהגות המאופיינת כנורמאלית.
תת קבוצה משמעותית של קטגוריה זו משתמשת בהגדרות או מערכת חוקים, הניתנים לעדכון או שינוי, לצורך הגדרת התנהגות נורמאלית של קוד מתוך מטרה לאתר סטייה של הקוד תחת המבחן מחוקים אלו. תוכנית הסוטה מסט חוקים נחשבת לא נורמאלית ובדרך כלל בעלת מטרה זדונית.
שיטת פעולה של תת קבוצה זו נקראת גילוי מבוסס מפרט
(Specification-based detection).

- גילוי המתבסס על חתימות ידועות מראש של זדונה קיימת. בשיטה זו מטרת החיישן היא לאתר קטע קוד, הידוע מראש כזדוני, הנמצא כחלק מהתוכנה הנבחנת מתוך מטרה לאפיין את התוכנה כזדונית.

כל אחת מהקטגוריות עושה שימוש באחת משלוש גישות שונות, המגדירות בעקיפין את השלב בו המידע לגבי התוכנה, תחת המבחן, נאסף:

- אנליזה סטאטית, העושה שימוש בתחביר או תכונות מבנה של הקוד מתוך מטרה לזהות חוסר נורמאליות בתוכו. לדוגמה, אנאליזה סטאטית בגילוי מבוסס חתימות ינסה לזהות רצף ידוע מראש של ביטים בתוך קוד התוכנית. מטבע הדברים, אנאליזה סטטית תיעשה לפני שהתוכנה תרוץ בפועל על המחשב.

- אנאליזה דינאמית, בה התוכנית נבחנת לאחר שהיא מתחילה לרוץ על המחשב או לעיתים לאחר סיום הריצה. לדוגמה, בגילוי אי-נורמאליות, החיישן ינסה לאתר התנהגות לא צפויה, מהתכנית שתחת המבחן, כגון קריאות לא צפויות למערכת ההפעלה, פנייה לקווי תקשורת או דפוסי התנהגות אחרים שאינם צפויים.

- שיטה היברידית המשלבת את שתי השיטות שתוארו לעיל.

כנגד כל סוג של איום קיימת גם הדרך המתאימה להתמודדות. לדוגמה, כנגד ווירוס מחשב קיימות טכניקות ספציפיות של סריקת הקוד, החל מהשיטות הוותיקות והמסורתיות המנסות לזהות דפוסים ספציפיים וידועים מראש של ווירוס מסוים וכלה בשיטות היוריסטיות, המשתמשות במודלים סטטיסטיים על מנת לאתר מוטציות של ווירוסים ומאפשרות גילוי של קבוצות שלמות של ווירוסים. מנגנוני ההגנה המתקדמים כנגד ווירוסים אף מסוגלים, במקרים רבים, להחזיר את הקובץ הנגוע למצבו הנקי המקורי. חלוקה נוספת היא לפי זמן פעולת ההגנה. קיימים מנגנוני הגנה לפי דרישה אשר סורקים, בהמשך לדרישת המשתמש, את הקבצים, כולל אלו שטרם הופעלו, במטרה למצוא ווירוסים. לעומתם קיימים מנגנוני הגנה בעת גישה אשר נמצאים בכל עת בזיכרון המחשב בעת הריצה ומפקחים על גישת התוכניות הרצות ועל פנייתם למשאבים שונים של המחשב. האחרונים מסוגלים לזהות פעילות עוינת של תוכנה במחשב בזמן אמיתי בעת ריצתה [14 p. 414].

נראה כי התבססות ההגנה כנגד תולעים על חתימות הייתה מאפשרת ההתפשטות מהירה ביותר של התולעים עד שהייתה נמצאת חתימת התולעת וממומשת ההגנה המתאימה, אילולא זיהו מפתחי האנטי ווירוס את פוטנציאל האיום ופיתחו הגנות מתוחכמות יותר כמנגנוני בלימה לתסריטי פקודות מחשב המזוהים בזמן ריצה בניסיון לבצע פעילות החשודה כזדונית ומנגנונים אחרים לבלימת תולעים המנצלים את פרוטוקול ה SMTP (Simple Mail Transport Protocol) המשמש בסיס להעברת הדואר האלקטרוני דרך תשתית האינטרנט. מנגנונים אלו הוכיחו עצמם כיעילים ביותר, עובדה שהביאה לירידה בשנים האחרונות בכמות מערכות המחשב הנגועות בתולעים בהשוואה לתקופה הראשונה בה נחשפו איומים אלו[14 p. 517] .

להבדיל מהשיטות ההגנה המסורתיות, המתבססות על מנגנונים המותקנים על המארח עצמו, טכניקות הגנה נוספות ומתקדמות כנגד התקפות מתבססות על שכבות הרשת והתקשורת למחשב. מנגנונים אלו מספקים הגנה כנגד תולעים, דלתות אחוריות ואף התקפות מניעת שירות. בין השיטות ההגנה מבוססות רשת ניתן למצוא:

- רשימות גישה הנמצאות בנתבי הרשת ומאפשרות או חוסמות תנועה של בלוקי מידע (Packets) על סמך תכונותיהם.
- חומות אש המממשות מדיניות גישה לתוך הארגון.
- מערכות זיהוי חדירה דרך הרשת (NIDS – Network Intrusion Detection Systems) הבוחנות את כוון התנועה ותוכן חבילות התוכנה ברשת ומנסות לזהות דפוסים של חדירה על ידי זיהוי חתימות או בשיטה היוריסטית.
- מלכודות דבש בעלות מערכות הגנה מינימאליות לחדירה לצורך איתור איומים ולמידת דפוסי התנהגות של תוקפים.
- מערכות התראה מוקדמת שתפקידן לגלות כאשר מערכת נמצאת תחת התקפה ולהגן על משאבי המערכת. אלו אוספות מידע ממספר מוקדים של הגנה כמערכות אנטי ווירוס, חומות אש, מלכודות דבש ואחרים ומתריעות על התקפה צפויה על ידי הוספת התראות במאגר נתונים מרכזי.
- סינון כתובות אינטרנט (URL Filtering) המשמשות לסינון אתרים אשר הגלישה אליהם נמצאת בניגוד למדיניות הארגון או הידועים כבעלי מטרות זדוניות. שיטה דומה מתבססת על שירותי מוניטין (Reputation Services) המקנים לכל אתר ציון המבוסס על כתובת ה IP של האתר, הארגון אליו הוא שייך, הזמן ששם התחום (Domain Name) של האתר רשום, האם כתובת האתר מופיעה ברשימות דואר זבל ופרמטרים רבים נוספים.
- הגנה תוך שימוש בארגז חול (Sandboxing) המשתמש בטכנולוגיה המדמה סביבת ריצה, אולם בניגוד לסביבת ריצה רגילה, ב"ארגז חול" כל הקריאות למערכת מוחלפות על ידי קריאות מותאמות המאפשרות את ניטור הקריאות לצורך זיהוי פעילות עוינת.

המלחמה באיומים העומדים בפני מערכות המחשב צורכת משאבים נרחבים וזמן מחקר רב. לבני הכובע עוסקים במחקרים מתקדמים הכוללים, בין היתר, טכניקות של Reverse Engineering לצורך ניתוח התנהגות ומראה של איומים ובמיוחד, מציאת דרכים לניקוי מערכות נגועות. במסגרת מחקרים אלו מנסים החוקרים לאפיין קבוצות של ווירוסים הפועלים בצורה דומה ועל פי קריטריונים דומים מתוך מטרה לאפיין משפחות ווירוסים ולפתח הגנה משותפת בפני משפחות שלמות. לצורך מחקרים אלו, קיימים כלים, בדרך כלל יעודים של חברות אבטחה ואשר, בדרך כלל, אינם ניתנים לרכישה מסחרית ומהווים חלק מהקניין הרוחני של חסרות אלו.

2.5 נטיות וכוונים עתידיים.

מוסדות וארגונים, ממשלתיים ומסחריים הופכים להיות יותר ויותר תלויים ברשת לצורך ניהול עסקיהם, אם לצורך תקשורת פנים וחוץ ארגונית, קשר על לקוחות, סחר מקוון, גישה למאגרי מידע וצרכים רבים נוספים. עובדה זו נותנת לפושעי הרשת הזדמנויות רבות ומגוונות לאיתור חולשות במערכות המיגון של הארגון לצורך מתקפה. כל זאת מביא ארגונים להכרה כי עליהם לאמץ אסטרטגיות הגנה מתאימות על מנת להגן על הארגון בפני חדירת תוכן עוין, קוד או נתונים, ומתוך מטרה להמשיך את פעילות הארגון הנורמאלית ללא מפריע.

פושעי המחשב מונעים כיום לרוב ממניעי השגת רווח ולצורך כך הפכה הרשת לכר הפעילות העיקרי שלהם. היום כבר ברור כי ההגנות המסורתיות המבוססות על חתימות או רשימות מתוחזקות במאגרי נתונים אינן מספקות הגנה מספקת כנגד ההתקפות המתוחכמות, המשתמשות באמצעי הסוואה מתוחכמים לצורך הסוואת פעילותם. כיום ניתן לחזות בהתקפות רבות שמכוונות לפגוע בדיוק באותן נישות הנשארות חשופות על ידי מנגנוני אנטי ווירוס, סינון כתובות ואחרים.

אתרים זדוניים רבים מוקמים למספר שעות, מבצעים תרמית ויורדים מהרשת עוד לפני שחברות האבטחה מזהות את האיום ובונות הגנה מתאימה. באתרים לגיטימיים אחרים, תוכן האתר מוחלף על ידי קוד זדוני מוסווה, הממשיך את הפעילות הלגיטימית של האתר אולם במקביל מבצע פעילות נוספת שמטרתה חשיפה של נתוני המשתמש או כל פעילות עוינת אחרת. משלב השתלת הקוד העוין ועד השלב בו הוא נחשף יכול לחלוף זמן רב, כתלות באיכות ההסוואה של החדירה.

עובדות אלו ואחרות הביאו להכרה כי יש לפתח שיטות מתקדמות יותר להתמודדות בפשעי המחשב, כאלו שלא יתבססו על ידע מהעבר (חתימות וכדומה) אלא שיוכלו לזהות פעילות עוינת, עוד בטרם אופיינה ככזו בעבר, במטרה לספק הגנה בזמן אמת.

אחד מכווני המחקר נוגע ביכולת לזהות את אופיו של קובץ, ללא מידע מוקדם כחתימת ווירוס,על תוכנו. קיימים כיוונים רבים למחקר זה ועל חלקם נתעכב בפרק הבא.

פרק 3 רקע ועבודות קודמות - כוונים בזיהוי אופי קוד בעזרת אנאליזה סטאטית

מטרתו הראשונה של התוקף היא להחדיר את הקוד למערכת המחשב תחת "גלי הראדאר" של מערכות הגילוי למיניהם. אחד מהדרכים העיקריות המשמשות אותו לצורך הערמה על מערכות הגנה, היא הסוואת הקוד הזדוני, לעיתים על ידי הצגתו כטיפוס שונה של קוד. לדוגמה, הצגת קובץ ביצועי כקובץ תמונה.

בעיית ההתמודדות של מערכות ההגנה מחריפה כשמדובר בקוד שלא נצפה לפני כן. מערכות ההגנה המסורתיות מתקשות להתמודד עם קוד זדוני חדש ולא מוכר, כך שקיימת מוטיבציה רבה בקרב חברות האבטחה למציאת שיטות מתקדמות יותר, המסוגלות לסמן קוד לא מוכר כזדוני, וזאת על סמך מאפייני מבנה אלו ואחרים.

מנגנוני הגנה הוותיקים, מבוססי טביעת אצבע, מהירים ביותר אולם אינם סלחניים לקוד רב צורתי או קוד משנה צורה. במחקר שנערך בפברואר 2007, נבחנו 17 תוכנות אנטי וירוס, מהמובילות בתחום ההגנה, כנגד 12 תוכניות דוגמה ידועות שכללו קוד זדוני רב צורתי. אחוז ההחמצה הממוצע היה 38%, כאשר מספר וירוסים לא התגלו כלל [17]. מצד שני, מנגנוני הגנה מבוססי קוד היוריסטי מספקים הגנה טובה יותר אולם, המחיר במקרה זה הוא במהירות הביצוע ותלות בסביבת הריצה. לאור זאת, רצוי למצוא שיטת בדיקה המשלבת בין יתרונות השיטות השונות, מהירות האנטי וירוס ואיכות סריקת קוד היוריסטית. זוהי למעשה הפילוסופיה מאחורי השיטות שנבחן בפרק זה.

בפרק זה, בסעיפים 3.1 – 3.3, נסקור בצורה מעמיקה מספר מאמרים בתחום ההתמודדות בזדונות על ידי סריקת קוד ואפיונו כתמים או עוין. למאמרים אלו מכנה משותף עיקרי, הניסיון לאפיין מעין טביעת אצבע להתנהגות או מבנה הקוד, להבדיל מהשיטה המסורתית המאפיינת חתימה לאיום מסוים או סוג של איום. כל מאמר מציע דרך שונה להתמודדות עם האתגר של יצירת טביעת אצבע אולם נוכל לראות כי, למרות גישתם השונה, כל המאמרים דוגלים במציאת קווי אופי דומים לכל סוג קוד, אם לפי פיזור הביטים, מספר קריאות הפעולה (OPCODE), ייצוג קובץ ריצה על ידי גרף ומציאת מאפיינים על גרף זה או ניתוח סטאטי של התנהגות הקוד. המאמרים נסקרים בסדר אבולוציוני. בסעיף 3.4 נסקור בקצרה מאמרים נוספים העוסקים בתחום הקרוב לליבת עבודה זו ובסעיף 3.5 מובאת סקירה של מוצר מסחרי הנוקט בגישה פרואקטיבית לצורך איתור קוד עוין.

3.1 אלגוריתמים לזיהוי סוג הקובץ על פי תכולתו

מבוסס על המאמר Content Based File Type Detection Algorithms [18]

3.1.1 כללי

בעיית זיהוי טבעו האמיתי של קובץ מחשב עשויה להיות בעיה קשה. השיטות המסורתיות לזיהוי מתבססות בעיקר על אינפורמציה המכילה מידע על אופי הנתונים (Metadata) וכוללות:

- שימוש בסיומת שם הקובץ (File extension). שיטה זו אינה אמינה בעליל בהתחשב ביכולת של כל אחד לשנות סיומת זו ולהערים על מערכת המתבססת אך ורק על שיטת זיהוי זו. שינוי זה בלבד, מספיק על מנת שמערכות הפעלה מסוימות (לדוגמה Windows) תנסה לפתוח את הקובץ באופן שאינו מתאים לסוגו. כמו כן, מערכות סריקת וירוסים מסוימות יכולות להתעלם מקובץ זה ולא לסרוק אותו היות והן מותאמות לסרוק אך ורק קבצים בעלי סיומות מסוימות המוגדרות מראש, עובדה החושפת את המחשב למתקפה למרות שהוא מוגן על ידי מערכת אנטי וירוס.
- שימוש בסט של חוקים ידועים מראש עבור סוג קובץ. שיטה זו צורכת זמן, משאבים רבים ומומחיות של החוקר לצורך הגדרת חוקים אלו. ראשית, התהליך כולל בחינה של הגדרות סוג הקובץ וזיהוי תכונות מבנה שיאפשרו את זיהוי סוגו. במקרה של חוסר בתכונות כאלו, השלב הבא מצריך את הבחינה הפיזית של מבנה הקובץ וניסיון למצוא מאפיינים דומים במספר קבצים לגיטימיים מאותו סוג. שיטה זו נפוצה בעיקר במערכות הפעלה מבוססות UNIX. שיטה שכיחה, במערכת הפעלה זו, מבוססת על רעיון דומה ועושה שימוש במספרי קסם - קבועים הכלולים ב 16 הבתים הראשונים של הקובץ. קובץ במערכת (בדרך כלל נקרא magic) מכיל רשימה המשייכת לכל מספר קסם את סוג הקובץ, ובעזרתו מערכת ההפעלה מזהה את סוג הקובץ.
לשיטה האחרונה מגבלות רבות. מספרי הקסם חייבים להיות מוגדרים מראש ולהשתלב בגוף הקובץ. בתחילה מטרת המספר הייתה לזהות קבצים בינאריים וקבצי ריצה, בעוד שסוגים אחרים בדרך כלל לא זוהו. הגדלת תחום הקבצים המזוהים חייבה הוספת הרחבות למספרי הקסם היות ו 16 בתים לא הספיקו יותר, מה שהקשה את התמיכה בקבצים וותיקים שהכילו מספר קסם ישן.

מוגבלות שיטות אלו ואחרות מעלה את הצורך במנגנון אוטומטי ויעיל לאפיון סוג קובץ, הנושא בו מתמקד מאמר זה. המאמר מתאר ניסיון להרחיב את רעיון ניתוח תדירות הבתים בגוף הקובץ, למקרה הכללי של מציאת טביעת אצבע לסוגי קבצי מחשב שונים. בהימצא טביעת אצבע אמינה לסוג קובץ, ניתן בהמשך ליצר טביעת אצבע עבור קובץ לא ידוע, שאת סוגו ברצוננו לוודא, ולהשוות לטביעת האצבע המאפיינת סוג קובץ זה לצורך אישור או שלילה. תהליך זה יהיה אוטומטי לחלוטין ולא יושפע משינוי מאפיינים חיצוניים של הקובץ כסיומת שלו.

המאמר מציב מספר אתגרים לצורך השגת מטרתו:

- על הפתרון המוצע להיות מדויק על מנת למנוע ריבוי של טעויות מהסוג הראשון או השני (Type I error - False Positive, Type II error – False Negative). טעות מסוג I מתרחשת כאשר מערכת הגנה מזהה ומדווחת, בטעות, על קוד זדוני שאינו קיים. טעות מסוג זה לא תגרום לפגיעה של קוד עוין במערכת המחשב, אולם היא עלולה לפגוע בחויית המשתמש על ידי חסימת הגישה לקוד לגיטימי לחלוטין, ותגרום לבזבוז זמן מיותר במרדף שוא. טעות מסוג II חמורה יותר ומתרחשת כאשר מערכת ההגנה אינה מאתרת קוד זדוני ומאפשרת לו לחדור למערכת המחשב. במקרה זה, בניגוד לטעות מסוג I, קיימת סכנה ממשית למערכת. כל מערכת הגנה סובלת, במידה זו או אחרת, מהמצאות טעויות מסוג I ו II אולם ריבוי טעויות אלו עלול לגרור לאובדן האימון במערכת.
- תהליך יצור החתימות צריך להיות אוטומטי.

- קובץ טביעת האצבע חייב להיות קטן ככל שניתן ואינו תלוי בגודל הקובץ הנבדק.
- תהליך ההשוואה בין חתימות חייב להיות מהיר.
- חייבת להיות דרך גמישה לבחירה בין רמת דיוק זיהוי לעומת מהירות דגימה גבוהים יותר.

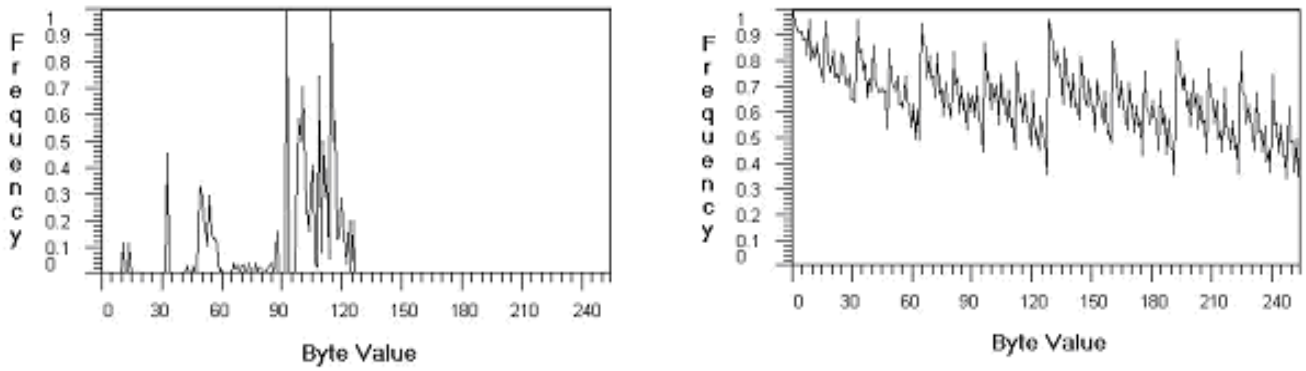
כפתרון, המאמר מציע שלושה אלגוריתמים לזיהוי אוטומטי של סוג הקובץ נבדק:

3.1.2 ניתוח תדירות בתים

Byte frequency analysis (BFA) algorithm

קובץ מחשב מורכב מבתים המייצגים כל אחד ערך בתחום [0..255]. על ידי ספירת מספר הבתים מכל סוג, ניתן לחשב את התפלגותם. סוגים רבים של קבצים מכילים דפוסים קבועים של התפלגות בתים, עובדה המקלה את זיהוים.

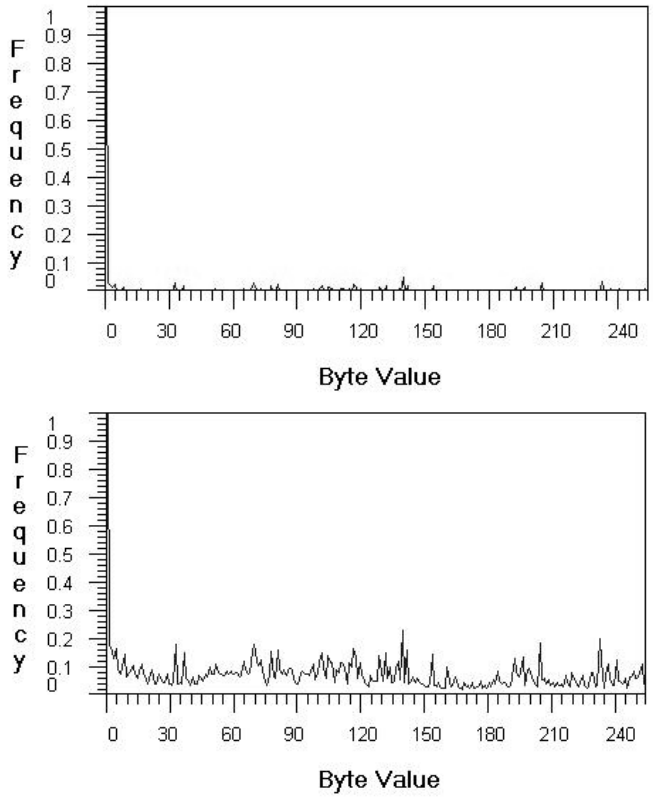
המאמר מציג תרשים התפלגות עבור שני קבצי RTF (Rich Text File) ושני קבצי GIF (Graphic Interchange Format). ניתן בבירור להבחין בהבדלים בפיזור בין שני סוגי הקבצים. סוגים רבים נוספים של קבצים כוללים אף הם מאפיינים ייחודיים לסוגם.



תרשים 1: תרשים התפלגות הבתים של קובץ RTF (משמאל) לעומת קובץ GIF

בחלק זה של המאמר מתוארת השיטה בה נאסף המידע על ההתפלגות של הבתים בקובץ בודד והדרך לייצור טביעת אצבע בעזרת מידע זה. לצורך כך נעשה שימוש במערך באורך 256 שאיבריו בעלי מפתחות בתחום [0..255]. בשלב הראשון, כל אברי המערך מקבלים את הערך 0. עבור כל בית בקובץ הנבדק, התא המתאים במערך מקודם באחד, כך שבסיום תהליך דגימת הקובץ, מתקבל מערך שכל אחד מאיבריו מכיל את מספר הבתים בקובץ שערכם כערך מפתח האיבר. לצורך נרמול התוצאות, כל איבר במערך מחולק בערך הגדול ביותר הנמצא במערך, קרי, הבית המופיע במספר הפעמים הגדול ביותר בקובץ, והתוצאה היא שכל איבר במערך יכלול בסופו של תהליך ערך בתחום [0, 1]. צעד הנרמול נועד למנוע הטיה של תוצאות יצור טביעת האצבע לכוון קבצים גדולים יותר ומאפשר מתן משקל דומה לכל קובץ נבדק, ללא שום קשר לגודלו.

בסוגים מסוימים של קבצים נבחין בכך שבתים מסוימים נמצאים בכמות גבוהה משמעותית מאחרים. המאמר מציג כדוגמה את התפלגות הבתים בקובץ ביצועי (exe), שם ניתן להבחין בבירור בחוד גבוה בבית בערך 0 כאשר שאר הבתים אינם מספקים מידע על מנת שנוכל להבחין בתבנית מסוימת. ניתן לפתור בעיה זו על ידי העברת ההתפלגות דרך פונקציה שתפקידה לשכך את ההשפעה המזיקה של ערכים קיצוניים. המאמר מציין את האלגוריתמים A-law ו μ-law, המשמשים בעיקר בתחום התקשורת, כמתאימים למטרה זו. העברת הנתונים דרך הפונקציה תשמר את הנרמול בתחום $[0, 1]$. המאמר מציג את הגרף המתקבל לאחר שהעבירו את הקובץ דרך פונקציית שיכוך ואכן ניתן להבחין שהפונקציה מיתנה את השפעת הערך 0 וחיזקה את השפעת שאר הערכים כך שמתקבלת תבנית ברורה יותר, המאפשרת השוואה נוחה יותר לקבצים אחרים.



תרשים 2: דוגמה להתפלגות הבתים בקובץ exe (למעלה) והתוצאה לאחר העברת הנתונים בפונקציית השיכוך

על ידי מיצוע הנתונים של קבצים שונים מהטיפוס הרצוי, ניתן לקבל את טביעת האצבע המייצגת את הטיפוס עצמו. לצורך הוספת נתוני קובץ נוסף לטביעת האצבע, ניתן להשתמש במשוואה:

$$\mathrm{NFPS} = \frac{(\mathrm{OFPS} \times \mathrm{PNF}) + \mathrm{NFS}}{\mathrm{PNF} + 1}$$

כאשר נתוני המשוואה הם:

- NFPS – ערך הבית בטביעת האצבע המתקבלת החדשה.
- OFPS – ערך הבית בטביעת האצבע הנמצאת בידינו.

- PNF – מספר הקבצים ששימש לחישוב טביעת האצבע שבידינו.
- NFS – ערך הבית המנורמל בקובץ הקלט.

ניתן להבחין כי בשימוש במשוואה לצורך הוספת נתוני הקובץ הראשון (PNF = 0), נקבל טביעת אצבע זהה להתפלגות הבתים של קובץ זה.

בנוסף לחישוב התפלגות הבתים, עלינו לשים לב לתופעה נוספת שתאפשר לנו לעדן את תוצאות הבדיקה. עבור קבצים השייכים לטיפוסים מסוימים, נמצא ערכים בעלי שכיחות עקבית בין הקבצים השייכים לאותו הטיפוס, בעוד שכיחות ערכים אחרים תהיה אקראית. לדוגמה ניתן לראות את הגרף המייצג את התפלגות הבתים בקבצי RTF, בו כמעט כל הבתים שוכנים בתחום הערכים [32..126] המייצגים תווים הניתנים להדפסה, תכונה אופיינית לפורמט של קובץ Rich Text. לעומת זאת, קיים הבדל גדול בפיזור התווים בתחום זה בין קובץ לקובץ, כתלות בתוכן המסמך. תכונה זו מאפשרת הוספת מדד נוסף, חוזק המיתאם (Correlation Strength) של ערך בית בין קבצים שונים מאותו הטיפוס. ניתן להוסיף מדידה זו כחלק מטביעת האצבע של הקובץ הנבדק, במילים אחרות, אם ערך בית מסוים מופיע תמיד באותה תדירות עבור קובץ מטיפוס מסוים, משמעות הדבר שזוהי תכונה חשובה של טיפוס זה ולכן משמשת כפיסת מידע חשובה בזיהויו.

חוזק המיתאם עבור קובץ חדש, יהיה אף הוא בתחום [0, 1] ויחושב על ידי השוואת ערך הבית של הקובץ שנוסף לדגימה, לערך הקיים בטביעת האצבע שחושבה עד כה. ככל שההבדל בין שני הערכים קטן יותר, יגדל ערך המקדם עבור בית זה לכוון 1. ככל שההבדל יהיה גדול יותר, ישאף הערך ל – 0. פונקציה שתחזק הפרשים קטנים ותחליש הפרשים גדולים היא זו המתארת עקומת פעמון עם שיא בעל ערך 1 הממוקם הנקודת ה – 0 של הציר האופקי. המשוואה הכללית המתארת עקומה זו היא:

$$F(x) = e^{\left(\frac{-x^2}{2\sigma^2}\right)}$$

כאשר $F(x)$ הוא חוזק המיתאם, x הוא ההפרש בין ערכו של הבית החדש והערך הממוצע המחושב של אותו הבית בטביעת האצבע ו σ- מקדם. ניסויים גילו ש $\sigma = 0.375$ מספק את הזיהוי המדוייק ביותר.

לאחר חישוב חוזק המיתאם עבור כל בית בקובץ הקלט, ניתן לשלב את הערכים המחושבים בטביעת האצבע. השילוב נעשה בדרך דומה לחישוב טביעת האצבע עבור התפלגות הבתים בעזרת הנוסחה:

$$NCS = \frac{(OCS \times PNF) + NCF}{PNF + 1}$$

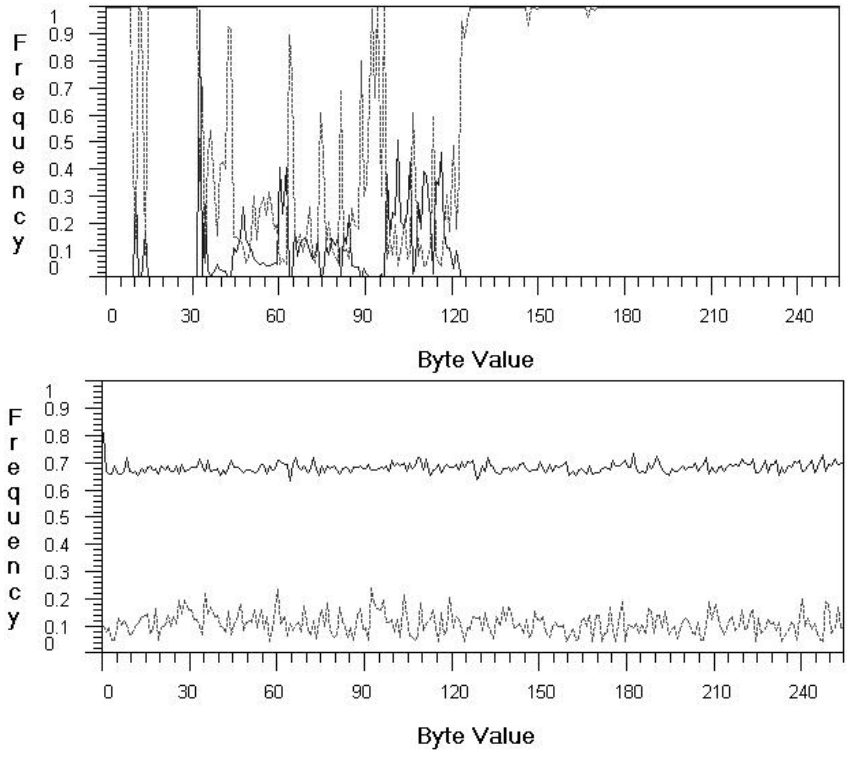
כאשר נתוני המשוואה הם:

- NCS – חוזק המיתאם החדש המתקבל.
- OCS – חוזק המיתאם שבידינו.
- PNF – מספר הקבצים ששימשו לחישוב חוזק המיתאם שבידינו.

- NCF – חוזק המיתאם של הבתים בקובץ הקלט, כפי שחושב בעזרת המשוואה.

האלגוריתם למציאת טיפוסו של קובץ קלט לא ידוע מורכב מהשלבים הבאים:

- חישוב ציון הדמיון לטביעת האצבע - עבור כל טביעת אצבע של טיפוס ידוע, חשב את המרחק בין התפלגות הבתים של הקובץ הלא ידוע, להתפלגות הבתים בטביעת האצבע. ככל שהמרחק בין הבתים קטן, ציון ההשוואה גדל לכוון 1 וככל שהמרחק גדל, הציון קטן לכוון 0.
- חישוב חוזק המיתאם - עבור כל חוזק מיתאם, חשב ציון המתאר את רמת האמון שנוכל לתת בהתאמה לטביעת האצבע הנבדקת וזאת בעזרת התכונה שלטיפוס קובץ, בעל התפלגות אופיינית מובהקת, נקבל חוזק מיתאם גבוה יותר עבור ערכי בתים רבים.
- מציאת הטיפוס בעל ההתאמה הגבוהה ביותר - עבור כל טביעת אצבע של טיפוס כלשהו, השווה את תוצאות הדמיון וחוזק המיתאם בין הקובץ הנבדק לטביעת האצבע ובחר בהתאמה הטובה ביותר.



תרשים 3: גרף התפלגות הבתים (קו רציף) וחוזק הקשר ההדדי (מקווקו) עבור קובץ HTML (למעלה) וקובץ ZIP

לסיכום סעיף זה, המאמר מציג שני תרשימים, אחד עבור קבצים מטיפוס HTML ואחד לקבצים מטיפוס ZIP. בכל תרשים, גרף המתאר את התפלגות הבתים וגרף המתאר את חוזק המיתאם. ניתן לראות כי בקובץ מטיפוס HTML, חוזק המיתאם גבוה בדרך כלל, עובדה המאפשרת מתן אמון רב יותר בהשוואת טביעת האצבע. לעומת זאת, קבצים מטיפוס ZIP בעלי חוזק מיתאם נמוך, עובדה המצדיקה בחינת דרך אחרת לבדיקת סוג הקובץ.

3.1.3 ניתוח קשר הדדי בין תדירות בתים

Byte frequency cross-correlation (BFC) Algorithm

בעוד שאלגוריתם BFA משווה את התפלגות הבתים באופן כולל, הוא אינו מתייחס לתכונות נוספות חשובות של ההתפלגות. בתרשים התפלגות הבתים עבור קובץ HTML למשל, ניתן להבחין בבירור בשני נקודות קיצוניות בערכים 60 ו – 62. ערכים אלו מייצגים את התווים'<' ו – '>' המשמשים כתג פותח וסוגר לזיהוי תווית HTML חוקית. היות ושני תווים אלו משמשים בדרך כלל כצמד לעטיפת התווית, נצפה לראות אותם במספר דומה או זהה של מופעים. יחס זה בין זוגות בתים ניתן למדידה ולמתן ציון לחיווי הקשר בין מספר מופעיהם. חלק זה במאמר מתאר את הדרך לאיסוף ולבניית טביעת אצבע המתבססת על הצלבת מופעי זוגות תווים שונים והשימוש בטביעת אצבע זו לאישור או שלילת שייכותו של קובץ לא ידוע לקבוצת טיפוס.

לצורך ביצוע אנליזה לחישוב המיתאם נחשב את נתוני המידע הבאים:

- חישוב חוזק המיתאם– כפי שחושב באלגוריתם BFA עבור כל זוג בתים.
- ההבדל הממוצע בהתפלגות בין כל זוג בתים. צמדי בתים המופיעים בצורה עקבית בתדירות דומה, כדוגמה שניתנה עבור הערכים 60 ו – 62 בקובץ HTML, יקבלו ערך גבוה יותר של חוזק מיתאם, בעוד שזוגות ערכים ללא קשר בין מספר מופעיהם יקבלו ערך נמוך.

האלגוריתם עושה שימוש במערך דו מימדי (מטריצה) בגודל 256 × 256 לצורך אכסון הנתונים. מפתח השורות והעמודות נמצא בתחום [0..255]. עבור בתים i ו – j, $i > j$, האיבר (i, j) במערך יכיל את המיתאם בעוד האיבר (j, i) יכיל את הפרש התדירות בין הבתים i ו - j. השוואת בית לעצמו תניב תמיד את הערך 0 עבור הפרש התדירות והערך 1 עבור המיתאם ולכן אלכסון המטריצה יכול לשמש לאכסון נתוני עזר אחרים. האלגוריתם המתואר עושה שימוש בתא הראשון, (0, 0), לאכסון מספר הקבצים ששימשו לחישוב טביעת האצבע.

חישוב הפרש התדירות בין בתים i ו – j נעשה על ידי החסרת ערך מקדם ההתפלגות המנורמל של i מזה של j. היות והערכים המחושבים נמצאים בתחום [0, 1], התוצאה שתתקבל תהיה בתחום [-1, 1] , ככל שערך התוצאה מתקרב ל -1, המשמעות היא שהבית שערכו i נמצא בתדירות גבוהה בהרבה מזה שערכו j, ולהיפך, כאשר ערך התוצאה מתקרב ל 1. כאשר ערך התוצאה מתקרב ל 0, המשמעות היא שתדירות הופעת שני הבתים דומה או זהה.

לאחר חישוב הפרש תדירות הבתים, בדומה לאלגוריתם BFA, ניתן לחשב את טביעת האצבע בעזרת הנוסחה:

$$\mathrm{NFPD} = \frac{(\mathrm{OFPD} \times \mathrm{PNF}) + \mathrm{NFD}}{\mathrm{PNF} + 1}$$

כאשר נתוני המשוואה הם:

- NFPD – הערך בטביעת האצבע המתקבלת החדשה.

- OFPD – הערך בטביעת האצבע שבידינו.
- PNF – מספר הקבצים ששימשו לחישוב טביעת האצבע שבידינו.
- NFD – הערך המחושב עבור קובץ הקלט.

ערך חוזק המיתאם, הנמצא בתחום [0, 1], ניתן לחישוב עבור כל זוג בתים על ידי השוואת הפרש תדירותם עבור קובץ הקלט לעומת זה הנמצא בטביעת האצבע. ככל שההפרש בין שני הערכים יהיה קטן יותר, כך מקדם האמון בטביעת האצבע עבור זוג זה יגדל לכוון הערך 1 ולהיפך. לצורך החישוב ניתן לעשות שימוש באותה הנוסחה המשמשת באלגוריתם BFA. ככל שנוסיף קבצי דגימה לטביעת האצבע, כך נקבל מקדם אמון המשקף את טיפוס הקובץ בצורה מדויקת יותר.

השוואת תוצאות הריצה של אלגוריתם BFA לאלו של BFC מציגה תוצאות מעניינות. המאמר משווה לדוגמה בין תוצאות הריצה עבור קבצים מטיפוס HTML. בקובץ HTML אופייני קיימים ערכי בתים, בדרך כלל בערכים הגבוהים מ 127, שמספר המופעים שלהם הוא באופן עקבי 0. עבור זוגות בתים הנמצאים באזורים אלו נקבל הפרש תדירות בערך 0. כמו כן, עבור הצמד 60 (>) ו 62 (<) נקבל ערך 1 עבור חוזק המיתאם.

השיטה המשמשת לאפיון הטיפוס של קובץ לא ידוע לפי אלגוריתם BFC:

- חישוב ציון הדמיון לטביעת האצבע - עבור כל טביעת אצבע של טיפוס ידוע, חשב את המרחק בין הפרש ההתפלגות של כל צמד בתים בטביעת האצבע מהצמד המקביל בקובץ הקלט. ככל שהמרחק בין הצמדים קטן, הציון ההשוואה גדל לכוון 1 וככל שהמרחק גדל, הציון קטן לכוון 0.
- חישוב חוזק המיתאם - בדומה לאלגוריתם BFA. משמעות ציון גבוה יותר היא שניתן לתת משקל רב יותר לציון ההשוואה לטביעת האצבע.
- מציאת הטיפוס בעל ההתאמה הגבוהה ביותר – עבור כל טביעת אצבע, השווה את התוצאות שקיבלת לערכים שבטביעת האצבע ובחר בהתאמה הטובה ביותר.

3.1.4 ניתוח כותרת/סיומת תוכן הקובץ

File header/trailer (FHT) algorithm

בעוד שבמקרים מסוימים, האלגוריתמים BFA ו – BFC עשויים להיות יעילים בזיהוי טיפוס קובץ על פי התפלגות הבתים בתוכו, קיימים טיפוסי קבצים רבים בעלי מבנה בתים שאינו מכיל תבניות קלות לזיהוי.

דרך נוספת להתמודד עם זיהוי טיפוס הקובץ היא על ידי ניתוח ראשית הקובץ או הסיומת שלו (Header/Trailer). ראשית או סיומת הקובץ הם קטעי בתים בעלי אורך ומקום קבוע, הנמצאים בתחילת או בסוף רצף הבתים של הקובץ. חלק זה של המאמר מתאר את הדרך לאיסוף המידע מקבצי דגימה לצורך יצירת טביעת אצבע לפי ראשית או סיומת הקובץ וכן את הדרך לשימוש בטביעת אצבע לצורך זיהוי קובץ לא ידוע.

בשלב הראשון לבניית טביעת אצבע, יש להחליט על גודל הראשית והסיומת אותם האלגוריתם ידגום. נסמן ב H את אורך הראשית וב T את אורך הסיומת הנבחרים, H ו – T עשויים להיות באורכים שונים. בשלב הבא בונים שני מערכים דו מימדיים, האחד בגודל H × 256 (-0..H] [0..255]×[1) והשני בגודל T × 256 ([0..255]×[0..T-1].

בעת דגימת קובץ בודד, שני המערכים מאותחלים בערך 0 עבור כל איבריהם. לדגימת ראשית/סיומת הקובץ משתמשים במערך המתאים. עבור כל בית, מציבים 1 בתא הנמצא בשורה הנמצאת במיקום הבית הנדגם ובעמודה בערך בית זה. לדוגמה, אם בבית השלישי של הקובץ נמצא הערך 32 נציב את הערך 1 בתא (2, 31). בסיום התהליך נקבל שני מערכים בהם שורות המכילות 255 פעמים את הערך 0 ופעם אחת את הערך 1. מקרה יוצא דופן הוא כאשר אורך הקובץ קצר מ H או מ T. במקרה זה יסומנו כל ערכי השורות המיותרות בערך 1-. יש לשים לב למקרה מיוחד שבו אורך הקובץ ארוך מהראשית והסיומת אולם קצר מסכום אורכיהם. במקרה זה יתקבל שטח חפיפה בין מערכי הראשית והסיומת.

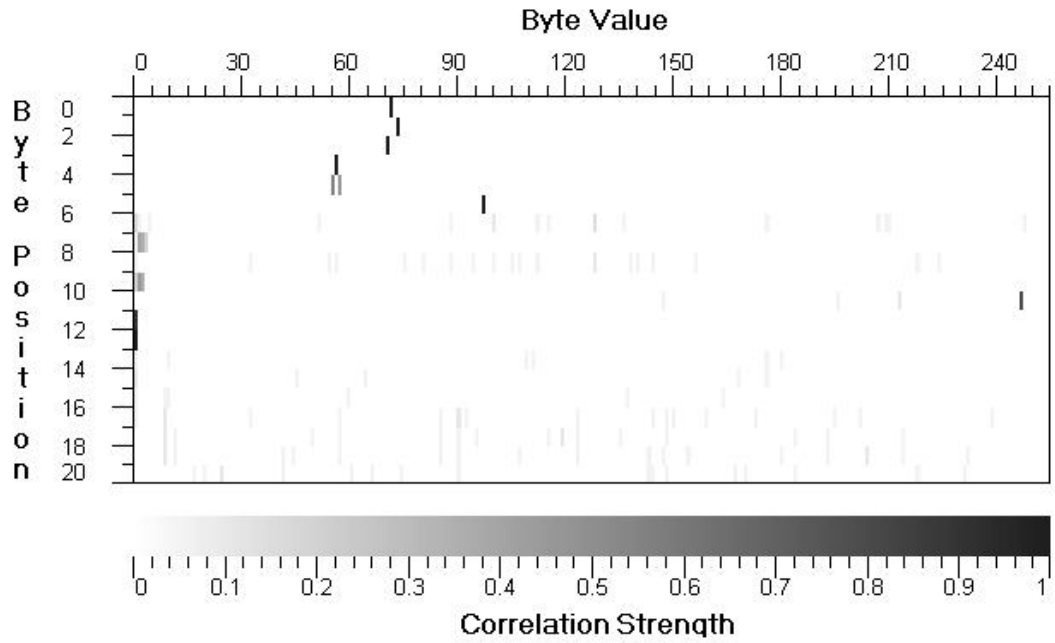
הדרך לבניית טביעת אצבע דומה לזו בה נוקטים באלגוריתמים BFA ו – BFC. לצורך כך משתמשים בנוסחה:

$$\mathrm{NFPA} = \frac{(\mathrm{OFPA} \times \mathrm{PNF}) + \mathrm{NA}}{\mathrm{PNF} + 1}$$

כאשר נתוני המשוואה הם:

- NFPA – ערך התא במערך טביעת האצבע המתקבלת החדשה.
- OFPA – ערך התא במערך טביעת האצבע הנמצאת בידינו.
- PNF – מספר הקבצים ששימש לחישוב טביעת האצבע שבידינו.
- NA – ערך התא במערך קובץ הקלט (0 או 1).

המאמר מציג בתרשים 4 את חתימת הראשית שנוצרה עבור קבצים מטיפוס GIF. הפס בחלקו התחתון של התרשים (Correlation Strength) משמש כמקרא המתאר את גוון הסימון עבור כל ערך בתרשים. ציר X מתאר את המיקום הפיסי של הבית בקובץ, החל מהבית הראשון בשורה העליונה וכלה בבית הנמצא במיקום ה 21. ציר Y של התרשים מתאר את ערך הבית והסימון בתוך התרשים מתאר את חוזק המיתאם בין המיקום בקובץ וערך הבית במיקום זה. לדוגמה ניתן לראות קו שחור בשורה 0 ובעמודה ה 71. המשמעות היא שבמקרה זה קיים מיתאם מלא בין הבית הראשון בקובץ וערכו, ותמיד יופיע הערך 71 במיקום זה. לעומת זאת, בשורה החמישית ניתן לראות שהבית עשוי להיות אחד משני הערכים, 55 ו 57, בהסתברות דומה. מפרט טיפוס GIF מציין כי ראשית הקובץ תכלול את המחרוזת "GIF87a" או "GIF89a". ואכן, בחינת תרשים טביעת האצבע מגלה חוזק מיתאם השווה ל - 1 עבור התאים הנמצאים בארבעת המקומות הראשונים וזה הנמצא במקום השישי. עבור המקום החמישי, ניתן להבחין כי ערכי הבתים 55 (ערך ASCII 7) ו 57 (ערך ASCII 9) חולקים בקירוב את אותו ערך מיתאם, עובדה המצביעה על דגימה מאוזנת של מספר קבצים מכל הסוג. בכל מיקום מעבר לשישי, ניתן להבחין בפיזור גבוה ומיתאם נמוך.



תרשים 4: חוזק המיתאם עבור ראשית של קובץ מטיפוס GIF

בדיקת תרשים הסיומת עבור קבצים מטיפוס MPEG תראה תופעה דומה של פיזור גבוה למעט ארבעת הבתים האחרונים בקובץ. בטיפוסי קבצים ללא מאפיינים מיוחדים בראשיתם או בסיומת שלהם, נקבל ערכי חוזק מיתאם נמוכים עבור כל התאים, עובדה המצביעה על חוסר התאמת האלגוריתם לזיהוי טיפוסים אלו.

השיטה המשמשת לאפיון הטיפוס של קובץ לא ידוע לפי אלגוריתם FHT:

- בניית מערכי הראשית והסיומת עבור הקובץ הלא ידוע.
- מתן ציון לראשית וסיומת הקלט, לצורך זה משתמשים בנוסחה:

$$\overline{S} = \frac{C_1 G_1 + C_2 G_{2\ldots} + C_n G_n}{G_{1_{max}} + G_{2_{max}} \ldots + G_{n_{max}}}$$

- עבור כל ערך במערך הצמוד לבית מסויים במיקום i, C_i מייצג את הערך המתאים מקובץ הקלט, G_i מייצג את חוזק המיתאם המקביל הנמצא בחתימה, ולבסוף, G_{max} מייצג את חוזק המיתאם הגדול ביותר במערך הנמצא בטביעת האצבע עבור הבית האמור. המשוואה מספקת ממוצע של חוזק המיתאם עבור כל ערך בית מקובץ הקלט, משוקלל לעומת חוזק המיתאם המקסימאלי המתאים למיקום. ניתן להבחין כי בתים בעלי ערך מיתאם גבוה יותר מקבלים משקל משמעותי יותר מאלו בעלי הערך הנמוך.
- השוואת נתוני הראשית והסיומת מקובץ הקלט לחתימות השונות ובחירת טביעת האצבע המתאימה ביותר.

אם נבחן שוב את קובץ ה GIF בשיטה המתוארת לעיל, נראה כי חוזק המיתאם עבור ארבעת המקומות הראשונים והמקום השישי שווים ל – 1, היות וקבצי המדגם כללו ערכים זהים במקומות אלו. קובץ שחוזק המיתאם שלו יהיה שונה מ - 1 במקום המתאים במערך עבור בתים אלו, יקבל ציון נמוך ולא יעבור את המבחן. לעומת זאת, אם נבחר בית במקום אקראי אחר, השפעתו לא תהיה משמעותית בקבלת ההחלטה בגלל חוזק מיתאם נמוך.

הגדרת סף הקבלה לערך הגבוה ביותר תאפשר למספר בתים קטן ביותר, עם חוזק מיתאם גבוה ביותר להשפיע על ההחלטה, כמו לדוגמה במקרה של קובץ GIF.

לגודל הראשית והסיומת השפעה גם על מהירות ריצת האלגוריתם. ככל שאלו יהיו ארוכים יותר, כך זמן הריצה של האלגוריתם יהיה ארוך יותר. מניסיונות שערכו מחברי המאמר מתקבל כי גודל הראשית והסיומת האופטימאליים הוא 5.

3.1.5 תוצאות ניסיוניות

לבחינת האלגוריתמים השתמשו כותבי המאמר בשלושים חתימות לטיפוסים שונים ולצורך המבחן בחרו בארבע קבצים שונים מכל טיפוס, סך הכול 120 ספריה בת 120 קבצים. את הקבצים בפורמט של מיקרוסופט אופיס, ACD, DOC, PPT ו – XLS איחדו לטביעת אצבע אחת בפורמט OLE DOC (Object Linking & Embedding). טכנולוגיה זו של Microsoft מאפשרת יצירת קשר בין מסמכים מטיפוסים שונים של חבילת Office (לדוגמה Word ו – Excel) תחת מסמך אחד. איחוד זה גרר איתור מדויק יותר עבור האלגוריתמים BFC ו – FHT אולם מדויק פחות עבור אלגוריתם BFA.

לאחר הרצת שלושת האלגוריתמים על 120 הקבצים התקבלו התוצאות הבאות:

- אלגוריתם BFA השיג את התוצאות הפחות טובות, כאשר הצליח לזהות בהצלחה 27.5% מטיפוסי הקבצים. ממוצע זה טוב מניחוש אקראי אולם הוא בהחלט אינו מספק. במקרה של הפרדת קבצי ה - OLE DOC לטיפוסיהם המקוריים, האלגוריתם מזהה בהצלחה 29.17%, שיפור קטן ביחס לתוצאה הקודמת.

- אלגוריתם BFC הצליח לזהות בהצלחה 45.83% מטיפוסי הקבצים. אמנם מדובר בשיפור משמעותי לעומת אלגוריתם BFA אולם תוצאות אלו עדיין אינם מספקות לשימוש מעשי.
- אלגוריתם FHT השיג את התוצאות המדויקות ביותר כאשר הצליח לזהות בהצלחה 95.83% מטיפוסי הקבצים. ראוי לציין כי הפרדת קבצי ה - OLE DOC לטיפוסיהם המקוריים, תגרור ירידה בדיוק הזיהוי ל - 85%, כאשר מירב הטעויות מתקבלות בין הטיפוסים השונים שבתוך קבצי אופיס.

3.1.6 מסקנות

אלגוריתם BFA התגלה כמהיר ביותר בין האלגוריתמים, אולם רמת דיוקו היא הנמוכה ביותר. אלגוריתם BFC השיג אמנם תוצאות מעט מדויקות יותר אולם זמן הריצה שלו היה, באופן קיצוני, האיטי ביותר (כמעט פי 120 ביחס ל BFA), כך שאף הוא אינו מתאים עדיין לשימוש במערכות מסחריות.

אלגוריתם FHT השיג את התוצאות המדויקות ביותר וזאת, בזמן ריצה קרוב לזה שהשיג אלגוריתם BFA, מה שמעמיד אותו כמועמד הטוב ביותר לשימוש מעשי מבין השלושה. אולם, יש לקחת בחשבון כי קיימים סוגי קבצים, להם אין ראשית או סיומת טיפוסיים ולכן, קרוב לודאי ששימוש באלגוריתם זה כשיטה העיקרית לזיהוי טיפוס קובץ אינה מספקת.

3.2 קודי פעולה, כמנבאי זדונה

מבוסס על המאמר Opcodes as predictor for malware [17]

3.2.1 כללי

בדומה למאמר הקודם, מאמר זה עוסק בניתוח סטטיסטי של תכונות מבנה קובץ, אולם להבדיל, הוא עוסק בניתוח קבצים בינאריים ובוחר ביעד אחר לצורך הניתוח, קריאות קוד הפעולה המופיעות בחלק התפעולי של הקובץ הבינארי. הבדל נוסף בין המאמרים הוא במטרתו המוצהרת של זה, מציאת סטיות מהמצופה במבנה הבינארי של הקוד בקובץ במטרה לנבא אם הקוד זדוני. אולם, למרות המטרות השונות, תוכן המאמר תורם רבות להרחבת הידע בתחום ניתוח סטאטי של תוכן קובץ ולכן הוא מובא כאן.

המאמר מחולק למספר חלקים. בחלקו הראשון מוסברת המוטיבציה העומדת מאחורי המחקר ונערכת סקירה של התחום והעבודה הקשורה לנושא המאמר. משיקולי מקום וזמן, תזה זו לא תסקור חלק זה. בחלק הבא מוסברת הטכניקה לשליפת הקודים מהקובץ הבינארי, ומתוארים התהליכים לניתוח הסטטיסטי של התוצאות, ולבסוף, חלקו האחרון של המאמר דן במשמעות התוצאות, דרכים נוספות לשיפור התהליך, האיומים בעתיד ותרומת מאמר זה להתמודדות בהם.

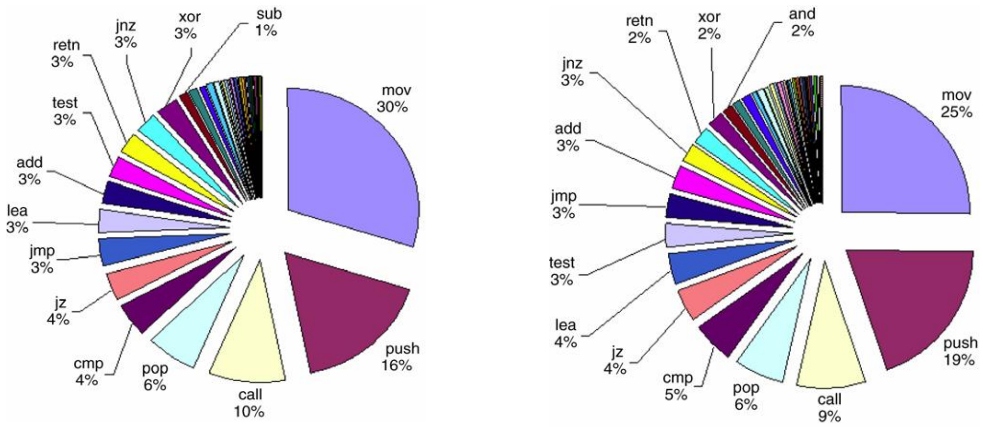
3.2.2 חילוץ קודי הפעולה (Opcodes) וניתוח סטטיסטי

בשלב הראשון, לצורך חומר לדגימה, נערך איסוף אקראי של דוגמאות קוד בינארי, ביצועי (exe) או ספריות (dll), חלקן נקיות מקוד זדוני (Goodware) וחלקן נגועות, סך הכול 20 דוגמאות שהתחלקו לארבע קבוצות גודל, חמישה קבצים בכל קבוצה. תחומי הגודל הוגדרו כ - [0 – 10 K), [10 – 100 K), [100K – 1M), [1 – 10M).

הזדונות ששימשו לצורך הדגימה כללו 77 קבצים נגועים שמוינו לשבע קבוצות, המאופיינות לפי סוג האיום ובינם תולעים, סוסים טרויאניים, וירוסים, תוכנות רובוטיות וערכות Rootkit משני סוגים.

תהליך ניתוח קוד זהה בוצע על כל קובץ דגימה, נגוע או נקי. הקובץ נבחן על ידי תוכנית הקוראת קוד בינארי ובין היתר, מפרקת אותו לפקודות אסמבלר (IDA Pro, DataRescue 2006). רכיב תוכנה בתכנית שלף נתונים סטטיסטיים לגבי קודי פעולה שאוחזרו ותכנית שירות נוספת שלפה את נתוני המהדר, אם זוהו. אם מדובר בקוד נקי, נוסף באופן ידני מידע על סוג הקוד, לדוגמה, תכנית שירות, תכנית עריכה, רכיב תקשורת וכדומה. נתונים אלו נותחו על ידי תכנית ניתוח נתונים (Java Matrix numerical analysis package מחברת Mathworks) ונורמלו להמשך ניתוח בעזרת קובץ אקסל. כל התהליך בוצע על מערכת הפעלה Windows XP תחת סביבת ריצה ווירטואלית שסופקה על ידי תוכנה מחברת VMWare.

ניתוח הקוד הנקי העלה בערך 1.5 מליון קריאות שכללו 192 קודים שונים. תדירות השימוש ב 72 מתוכם הייתה מעל 99.8% מהמקרים, 14 קודים תפסו כ 90% וחמשת הקודים השכיחים ביותר תפסו כ 64% מהעוגה. המאמר מציג תרשימי עוגה של 14 הקריאות הנפוצות ביותר, כפי שנאגרו בין כל קטגוריות התוכנה, עבור קוד נקי וקוד נגוע. הרשימה הכוללת את חמשת הקודים הנפוצים ביותר (mov, push, call, pop, cmp) בשני המקרים זהה ובקריאות בעלות השכיחות הנמוכה יותר ניתן להבחין בהבדלים קטנים.



תרשים 5: הקריאות הנפוצות ביותר בין קטגוריות התוכנה עבור קוד נקי (מימין) וקוד נגוע

המאמר מציג טבלה המשווה את 13 הקודים השכיחים בחלוקה לפי אופי הקובץ, קרי, קוד נקי מול כל אחת מקטגוריות הזדונה. בהקשר לנתונים המוצגים בטבלה זו הועלו 3 שאלות:

- האם ניתן להבחין בשינוי סטטיסטי משמעותי בשכיחות קודי פעולה בין קוד נקי לבין 7 סוגי הזדונה?
- האם יש שינוי סטטיסטי משמעותי, לו אחראי/ם קוד אחד או יותר?
- עד כמה חזק המיתאם בין מחלקת הזדונה ושכיחות הקודים בתוכה?

למציאת תשובות לשאלות אלו עשה הכותב שימוש בשיטות סטטיסטיות, המפורטות במאמר. המבחן הראשון, chi-square, בודק את ההתאמה (ישור) בין שני מערכים עם נתוני מדידה גולמיים (לא יחסיים או אחוזים). במקרה זה נבחנות שכיחויות הקודים בתוך מחלקות הזדונות השונות, תוך השוואתם לשכיחות הקודים בתוכנות הנקיות. הנתון המחושב, z-score, מכמת את המרחק של כל תוצאה מתקבלת מהממוצע בתוכנות הנקיות ונמדד על ידי סטיית התקן.

מבחן chi-square אומר אם קיים קשר בין משתנים, אולם אינו מציין עד כמה הוא חשוב ומשמעותי. למתן מענה לשאלה זו משתמשים במבחן השני, Cramer's V, המשמש כדרך לחישוב מיתאם (Correlation) בתוך טבלאות בגודל העולה על 2×2. השימוש בו הוא למציאת חוזק הסמיכות (Strengths of Association) לאחר שמבחן chi-square בחן אם קיים קשר כלשהו בין הקוד למחלקת הזדונה. במקרה זה נבחן חוזק הסמיכות בין מופעי הקוד ומחלקת הזדונה בה הוא נמצא.

המבחן חולק לשתי חלקים. באחד הוא בוצע על קודים בעלי השכיחות הגבוהה ביותר ובשני על אלו בעלי השכיחות הנמוכה.

עבור הקודים בעלי השכיחות הגבוהה, במבחן ה z-score בהשוואה לקוד נקי, נמצא כי כ 1/3 מהקודים המופיעים בזדונות מציגים שכיחות דומה, כ – 1/3 שכיחות גבוהה יותר ו 1/3 נמוכה יותר. אם בוחנים את הנתונים לעומקם, מגלים תוצאות אופייניות שונות לפי סוגי מחלקת הזדונה. לדוגמה, עבור ערכות Root המיועדות לעבודה עם ליבת המחשב (Kernel), מתקבלות הסטיות הגבוהות ביותר לעומת קוד נקי. התופעה, אולי, יכולה להיות מוסברת על ידי קוד שיוצר באופן ידני, ללא עזרת כלים אוטומטיים. בווירוסים ותולעים, לעומת זאת, נמצאו הסטיות הנמוכות ביותר מקוד נקי (למעט פקודת jmp בתולעים), עובדה שמרמזת אולי על דמיון רב בין ווירוסים לקוד נקי ושימוש נרחב יותר בקפיצות בתולעים, מה שיכול להצביע על קוד פשוט ורזה יותר, המערב מרכיבים גדולים יותר של בקרת זרימה.

כאשר בוחנים את חוזק הסמיכות, התוצאות מצביעות על קשר חלש בין קוד למחלקת המופע שלו, מכאן שהשימוש בנתון במטרה לחזות את המחלקה, לה שייך הקובץ הנבדק, לא יניב תוצאות מספקות. לפי הנתון, התוצאות מסוגלות להסביר כ 15% – 5 מהשונות בין המחלקות.

היות ו - 14 הקודים בעלי השכיחות הנמוכה ביותר כמעט לא הופיעו כחלק מהמדגם, סוגי הקוד למבחן נבחרו בצורה אקראית מתוך כלל סוגי הקוד שהופיעו לכל היותר 0.2% בתוצאות המדגם.

ניתוח תוצאות המבחן שלהם העלתה תוצאות שונות מאלו שהתקבלו עבור קודים שכיחים. כ 70% מהקודים הציגו שכיחות דומה לזו של הקוד הנקי, כ 30% שכיחות גבוהה יותר וכ 10% שכיחות נמוכה יותר. סטיות נמוכות אלו מצביעות על חוסר מעשיות בהסתמכות על נתונים אלו לצורך ניבוי מחלקת הקוד הנבחן. למרות זאת, ניתן להבחין בשני נתונים מעניינים. ב Rootkits ניתן להבחין בשימוש רב בקוד int, מה שעשוי אולי להיות מוסבר על ידי שימוש נרחב בפסיקות ב - Rootkits. בווירוס ניתן לראות שכיחות גבוהה יותר של קוד nop, שיכול אולי לרמז על שימוש רב בריפוד (Padding) קוד.

בחינת חוזק הסמיכות בין הקודים בעלי השכיחות הנמוכה, מעלה תוצאות מעודדות בהרבה. במקרה זה, הסמיכות מצליחה לחזות ברמת דיוק של 63% – 10%. ניבוי Rootkits הוא בסבירות הגבוהה ביותר, כפי הנראה בשל ריבוי קוד הפסיקות.

לסיכום התוצאות, ניתן להתייחס למבחן חוזק הסמיכות כמסביר קשר בין מספר המופעים של הקוד למחלקת הזדונה בה הוא נמצא. ראינו כי בקוד בעל שכיחות גבוהה, תוצאות המבחן אינו מאפשרות להשתמש בו לצורך ניבוי המחלקה לה הוא שייך. בקוד בעל שכיחות נמוכה, לעומת זאת, התוצאות מצביעות על קשר חזק יותר, המאפשר במקרים מסוימים שימוש מעשי לצורך ניבוי מחלקת הקוד.

3.3 אנליזה סטטית של קבצים ביצועיים לאיתור דפוסים עוינים

מבוסס על המאמר Static Analysis of Executables to Detect Malicious Patterns [19]

3.3.1 כללי

מאמר זה מציע גישת אבחון סטטית שונה מקודמיו לאיתור קוד זדוני ביישומים. אם במאמרים הקודמים הגישה התבססה על ניתוח סטטיסטי של שכיחות בתים או קודי פעולה, מאמר זה עוסק בייצוג קבצי יישומים במבנה של גרף תלויות, ניתוח תכונות הגרף ויצירת טביעת אצבע, המבוססת על מבנה הגרף, עבור מחלקות תוכנה מיוחדות.

המטרה במקרה זה היא השוואת טביעת אצבע של קוד לא ידוע, vמבוססת על מבנה גרף, עם טביעות אצבע של קוד זדוני מוכר וידוע.

אחת הבעיות העיקריות העומדות בפני יצרני תוכנות האנטי ווירוס למיניהם, היא הקושי בהתמודדות עם קוד זדוני שעבר תהליכי הסוואה (Obfuscation). התהליך הקלאסי לאיתור ווירוס מנסה לאתר רצף הוראות, המזוהות עם ווירוס מוכר כלשהו, בתוך הקוד. רצף זה נקרא חתימת הווירוס. שיטה זו יעילה במקרה ומבנה הווירוס אינו משתנה לאורך זמן או כאשר מדובר במשפחת ווירוסים שמקורם באותו קוד, כך שניתן לשייך חתימה אחת לכולם.

למרבה הצער, כותבי הווירוסים עוסקים בפיתוח בלתי פוסק של אמצעי הסוואה והסתרה של קוד זדוני. בניסיון שערכו עורכי המאמר על מספר ווירוסים אשר שימשו לצרכי המחקר, הסתבר כי תוכנות האנטי ווירוס המסחריות שנוסו, Norton, McAfee ו Command, איתרו את הווירוסים בצורתם המקורית ללא בעיה אולם נכשלו בניסיון לאתר אותם לאחר שעברו תהליך הסוואה.

המאמר מציע טכניקה שמטרתה להתמודד עם שיטות ידועות של הסתרת קוד עוין ומציג את SAFE (Static Analyzer For Executables), כלי אב טיפוס לצורך הדגמת טכניקת הבדיקה המשמשת כבסיס למחקר.

בסעיפים הבאים, נסקור מספר שיטות הסוואה מקובלות של ווירוסים, נציג את ארכיטקטורת SAFE, אב הטיפוס להדגמת הטכניקה המוצעת ונסקור את תהליך המחקר ותוצאותיו.

3.3.2 שיטות הסוואת ווירוסים

בווירוס רב צורתי (Polymorphic), בדרך כלל קוד גוף הווירוס מוצפן ורק קטע קוד קטן, שמטרתו לפענח את גוף הווירוס המוצפן, נשאר חשוף. כותבי הווירוס מסווים את קטע קוד הפענוח במטרה למנוע את זיהוי הווירוס בעזרת התאמת טביעת אצבע. בווירוס משנה צורה (Metamorphic), כל הווירוס מוסווה בצורה שאינה משאירה אף קטע קוד חשוף לזיהוי.

לצורך הסוואות קטעי הקוד, עושים הכותבים שימוש בשיטות שונות שאת חלקן סוקר המאמר:

- השתלת קוד מת (Dead-Code Insertion) – בשיטה זו מוסיפים הוראות בגוף הקוד בצורה שאינה משנה את התנהגותו המקורית של הקוד. לדוגמה ניתן להוסיף מספר בלתי מוגבל של הוראת nop (No Operation) שאינה מבצעת בפועל דבר. ניתן להוסיף גם הוראות אחרות כגון הוספת או החסרת הערך 0 מאוגר כלשהו או כל הוראה דומה כיד הדמיון השורה על המחבר. הוספת הוראות אלו אינה משנה במאום את התנהגות הקוד אולם עשויה לשנות לחלוטין את טביעת האצבע שלו.
תוכנות האנטי ווירוס המודרניות מסוגלות לרוב להתמודד עם קוד מת בסיסי כ nop על ידי שימוש ב Regular Expressions, אולם עשויות להיתקל בקשיים במקרה של השתלת קוד מתוחכם יותר. ניתן להוכיח טענה זו על ידי ביצוע רדוקציה של בעיית הקוד המת לבעיה "האם רצף ההוראות הנבדק שקול לתכנית הריקה", שהיא בעיה בלתי פתירה. למרות זאת, עורכי המחקר מאמינים שמקרים רבים של רצפי קוד מתים ניתנים לאפיון וזיהוי במחיר ביצועי סביר.
- הזזת קוד (Code Transposition) – בשיטה זו מעבירים קטעי קוד ממקום למקום כך שהמבנה וסדר ההוראות שונה מסדר הביצוע שלהם בפועל. שינויים אלו מקשים ואף מונעים את זיהוי טביעת האצבע באמצעים הקיימים בתוכנות האנטי ווירוס כיום. על מנת לשחזר את סדר הביצוע המקורי של הקוד משתמשים בהוראות jmp המעבירות את השליטה מהוראה אחרונה בכל קטע קוד, להוראה הבאה אחריה, כפי שהייתה בסדר המקורי. שיטה נוספת מחליפה מיקום הוראות בתנאי שאינם תלויות זו בזו. יישום השיטה זו מורכב יותר היות ושלפני ההעברה, יש לוודא את חוסר התלות בין ההוראות.
שיטת הסוואה זו מקשה את איתור הקוד לעין האנושית אולם רוב כלי הניתוח המיועדים לאיתור קוד מוסווה, כולל זה הנמצא בלב מחקר זה, משתמשים בייצוג ביניים שאינו רגיש לתוספות מיותרות בניתוח זרימת הקוד. יש לציין שמייעל (Optimizer) של מהדר מבצע למעשה פעולת ניקוי דומה, על ידי איתור קפיצות בלתי מותנות מיותרות בקוד והסרתן.

Original code

```
call 0h
pop ebx
lea ecx, [ebx + 42h]
push ecx
push eax
push eax
sidt [esp - 02h]
pop ebx
add ebx, 1Ch
cli
mov ebp, [ebx]
```

Code obfuscated through dead-code insertion

```
call 0h
pop ebx
lea ecx, [ebx + 45h]
nop                    (*)
nop                    (*)
push ecx
push eax
inc eax                (**)
push eax
dec [esp - 0h]         (**)
dec eax                (**)
sidt [esp - 02h]
pop ebx
add ebx, 1Ch
cli
mov ebp, [ebx]
```

Code obfuscated through code transposition

```
        call 0h
        pop ebx
        jmp Step2
Step3:  push eax
        push eax
        sidt [esp - 02h]
        jmp Step4
        add ebx, 1Ch
        jmp Step6
Step2:  lea ecx, [ebx + 45h]
        push ecx
        jmp Step3
Step4:  pop ebx
        cli
        jmp Step5
Step5:  mov ebp, [ebx]
```

תרשים 6: דוגמת קוד מקורי (שמאל), והתוצאה לאחר השתלת קוד מת (אמצע) או הזזת קוד.

- הצבה מחדש באוגרים (Register Reassignment) – בשיטה זו מחליפים שימוש בין אוגרים הנמצאים באותו תחום חי. ההחלפה נעשית בפועל על שמות האוגרים ולכן אין לה השפעה על תוצאות הריצה של הקוד. החלפה זו דורשת לעיתים קוד הכנה, שתפקידו להגדיר את מצב האוגרים ההתחלתי וקוד סיום, שתפקידו לשחזר את המצב לקדמותו. שיטת הסוואה זו בסיסית ביותר וכל מטרתה היא למנוע השוואה לטביעת אצבע של וירוס.
- החלפת פקודות (Instruction Substitution) – בשיטה זו משתמשים במילון לצורך החלפת הוראות בודדות או קבוצת הוראות באחרות, בעלות משמעות זהה אולם מבנה שונה. המאמר מציג לדוגמה קטע קוד, בו פעולות ישירות על המחסנית (Stack) כ push ו pop מוחלפות בפעולות על הזיכרון שתוצאתן על המחסנית זהה. היות ומספר האפשרויות להחלפת קוד מוגבלת על ידי דמיון מחבר וירוס בלבד, שיטה זו מציבה את האתגר הגדול ביותר לאיתור אוטומטי של קוד עוין מוסווה. על מנת לטפל בקוד מוסווה בשיטה של החלפת פקודות, על כלי הניתוח להשתמש במילון זהה למילון בו נעשה שימוש לצורך הסוואת הקוד. פתרון זה אינו מקיף אולם ניתן למצוא מספר מקרים שכיחים יותר, הניתנים לתיעוד ושימוש על ידי כלי הניתוח. המאמר מציין גם כלים שמאפשרים ניתוח של שני קטעי קוד לצורך הוכחת או הפרכת הטענה שהם שקולים.

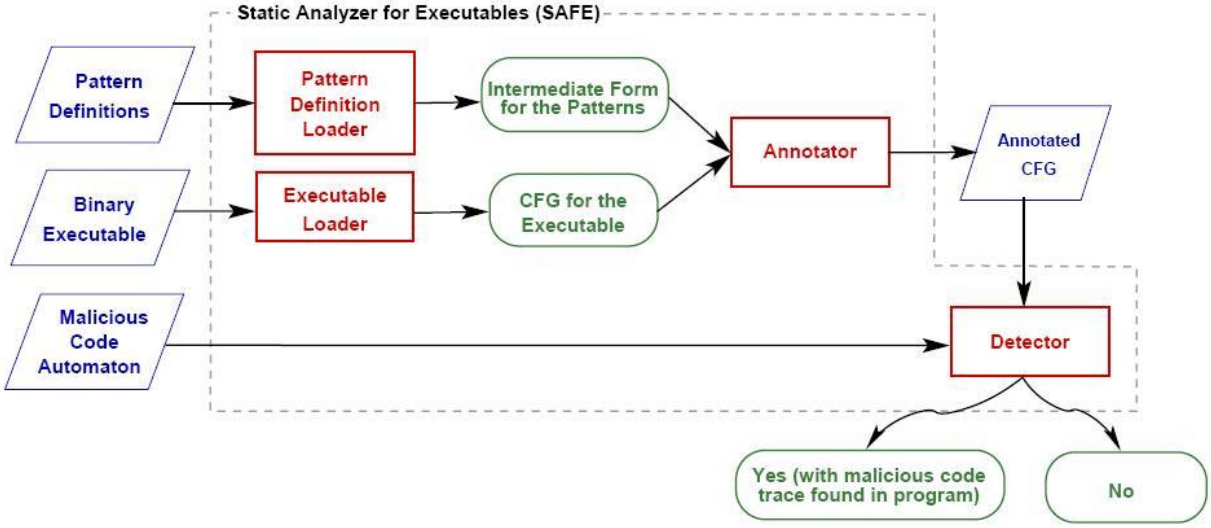
Original code	*Obfuscated code*
`call 0h`	`call 0h`
`pop ebx`	`pop ebx`
`lea ecx, [ebx + 42h]`	`lea ecx, [ebx + 42h]`
`push ecx`	`sub esp, 03h`
`push eax`	
`push eax`	
`sidt [esp - 02h]`	`sidt [esp - 02h]`
`pop ebx`	`add [esp], 1Ch`
	`mov ebx, [esp]`
`add ebx, 1Ch`	`inc esp`
`cli`	`cli`
`mov ebp, [ebx]`	`mov ebp, [ebx]`

תרשים 7: קוד מקורי (משמאל) והתוצאה לאחר החלפת פקודות

3.3.3 ארכיטקטורת SAFE (Static Analyzer For Executables)

לצורך גילוי דפוסים של קוד עוין בקבצים ביצועיים בנו עורכי המחקר את SAFE. סעיף זה מתאר את הארכיטקטורה של אב הטיפוס וחלקיו.

אב הטיפוס בונה ייצוג מופשט של הקוד הזדוני, אותו אנו רוצים לאתר, וייצוג דומה של הקוד אותו אנו רוצים לבחון. כאשר יש בידינו את הייצוג של שני קטעי הקוד, כל שנותר לעשות הוא לנסות ולאתר את הקוד הזדוני בתוך הקוד הנבחן.



תרשים 8: ארכיטקטורת SAFE

בהמשך ניתוח המאמר, נתייחס לקוד של ווירוס צ׳רנוביל המקורי, המוצג בתרשים 9 מימין, אחר שעבר תהליך הסוואה בעזרת קפיצות קוד, פעולות nop ופעולות לא רלוונטיות אחרות, כמוצג בתרשים 9 משמאל.

Obfuscated code

```
WVCTF:
        mov     eax, dr1
        jmp     Loc1
Loc2:
        mov     edi, [eax]
LOWVCTF:
        pop     ecx
        jecxz   SFMM
        nop
        mov     esi, ecx
        nop
        nop
        mov     eax, 0d601h
        jmp     Loc3
Loc1:
        mov     ebx, [eax+10h]
        jmp     Loc2
Loc3:
        pop     edx
        pop     ecx
        nop
        call    edi
        jmp     LOWVCTF
SFMM:
        pop     ebx
        pop     eax
        push    eax
        pop     eax
        stc
        pushf
```

Original code

```
WVCTF:
        mov     eax, dr1
        mov     ebx, [eax+10h]
        mov     edi, [eax]
LOWVCTF:
        pop     ecx
        jecxz   SFMM
        mov     esi, ecx
        mov     eax, 0d601h
        pop     edx
        pop     ecx
        call    edi
        jmp     LOWVCTF
SFMM:
        pop     ebx
        pop     eax
        stc
        pushf
```

תרשים 9: ווירוס צ'רנוביל, גרסה 1.4 (ימין) והגרסה המוסווית בעזרת פקודות קפיצה ו nop

3.3.3.1 הגדרות בסיסיות

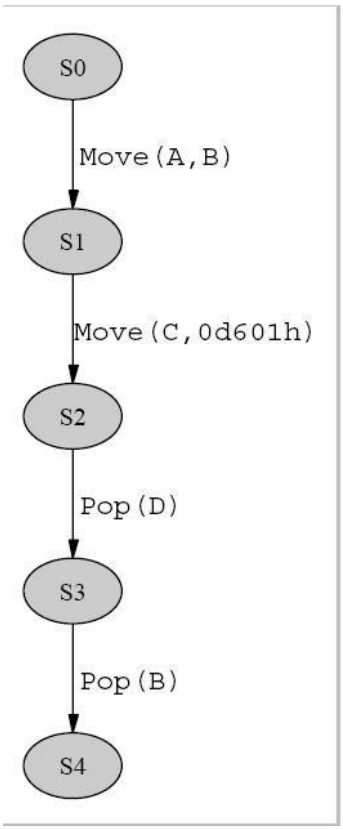
לצורך ההסבר על פעולת מערכת SAFE, נניח את היסוד על ידי מספר הגדרות בסיסיות.

- הוראה: בקשה לביצוע פונקציה בסיסית כלשהי. לצורך הפשטה לא נתייחס במקרה זה לפונקציות המורכבות ממספר פונקציות אחרות או פונקציות המסוגלות לקבל ולהחזיר פונקציות אחרות כארגומנט.
- תכנית: רצף הוראות המבוצעות אחת אחרי השנייה, לפי סדר הופעתן, למעט הוראות בקרת זרימה המסוגלות לשנות את סדר ביצוע ההוראות.
- בלוק הוראות בסיסי: רצף הוראות רגילות המכילות הוראת בקרת זרימה אחת, לכל היותר, בסופו של הבלוק.
- גרף זרימת בקרה: גרף שצמתיו מייצגים בלוקי הוראות בסיסיות וקשתותיו מייצגות את הוראות בקרת הזרימה. לכל קשת מוצמדת אות מייצגת לתנאי שצריך להתקיים על מנת לנוע בקשת זו. T מייצגת אמת ו - F מייצגת שקר. לקשתות המייצגות בקרת זרימה לא מותנית תמיד תוצמד האות T.

- פרדיקטים: מאפשרים לשאול שאלות בסיסיות על גרף זרימת הבקרה כגון, קבוצת הבלוקים הבסיסיים הקודמים לבלוק בסיסי מסוים, אלו הבאים מייד אחריו, מי ההוראה הראשונה או האחרונה של בלוק בסיסי, האם נקודת תוכנית מסוימת משתמשת או "מחסלת" משתנה מסוים ועוד שאלות כאלו ואחרות.

3.3.3.2 ייצוג הקוד זדוני

את הקוד הזדוני מייצגת ארכיטקטורת ה - SAFE בעזרת אוטומט (MCA – Malicious-Code Automaton) המבצע הכללה לקוד הוירוס המקורי (Vanilla Virus). הכללה זו מאפשרת את השוואת האוטומט לגרסאות מוסוות של אותו וירוס. המשתנים המשמשים באוטומט אינם מפורשים, והמאפיין היחידי המחייב הוא הטיפוס שלהם. עובדה זו מאפשרת את ניתוח התלויות ביניהם, תוך התעלמות משיקולים מגבילים כשמות משתנים או מקום ייצוגם בזיכרון. לדוגמה, בתרשים 10 מוצג חלק מהאוטומט המייצג את קוד וירוס צ'רנוביל, עם המשתנים המופשטים A, B, C, D. בתרשים ניתן לראות שתי גרסאות, β1 ו - β2 , של קישור משתנים מסוימים בתכנית למשתנים המופשטים.




```
mov     esi, ecx
mov     eax, 0d601h
pop     edx
pop     ecx
```

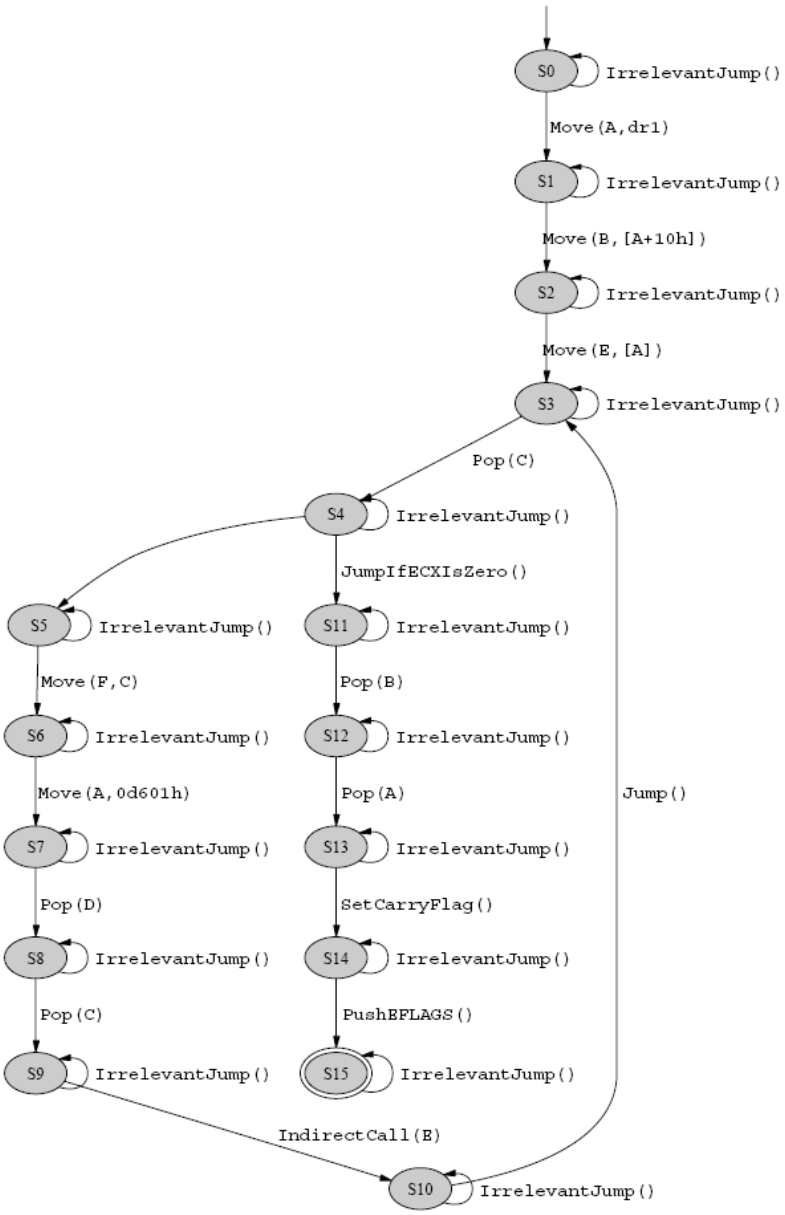
$\mathcal{B}_1 = \{ [A, esi], [B, ecx], [C, eax], [D, edx] \}$

```
mov     esi, eax
mov     ebx, 0d601h
pop     ecx
pop     eax
```

$\mathcal{B}_2 = \{ [A, esi], [B, eax], [C, ebx], [D, ecx] \}$

תרשים 10: חלק מהאוטומט המייצג את וירוס צ'רנוביל והצבות שונות של האוגרים

האוטומט המלא המייצג את הוירוס מוצג בתרשים 11.



תרשים 11: אוטומט MCA עבור ווירוס צ'רנוביל

3.3.3.3 יצירת גרף זרימת בקרה (CFG – Control Flow Graph) מפורש עבור הקוד הביצועי

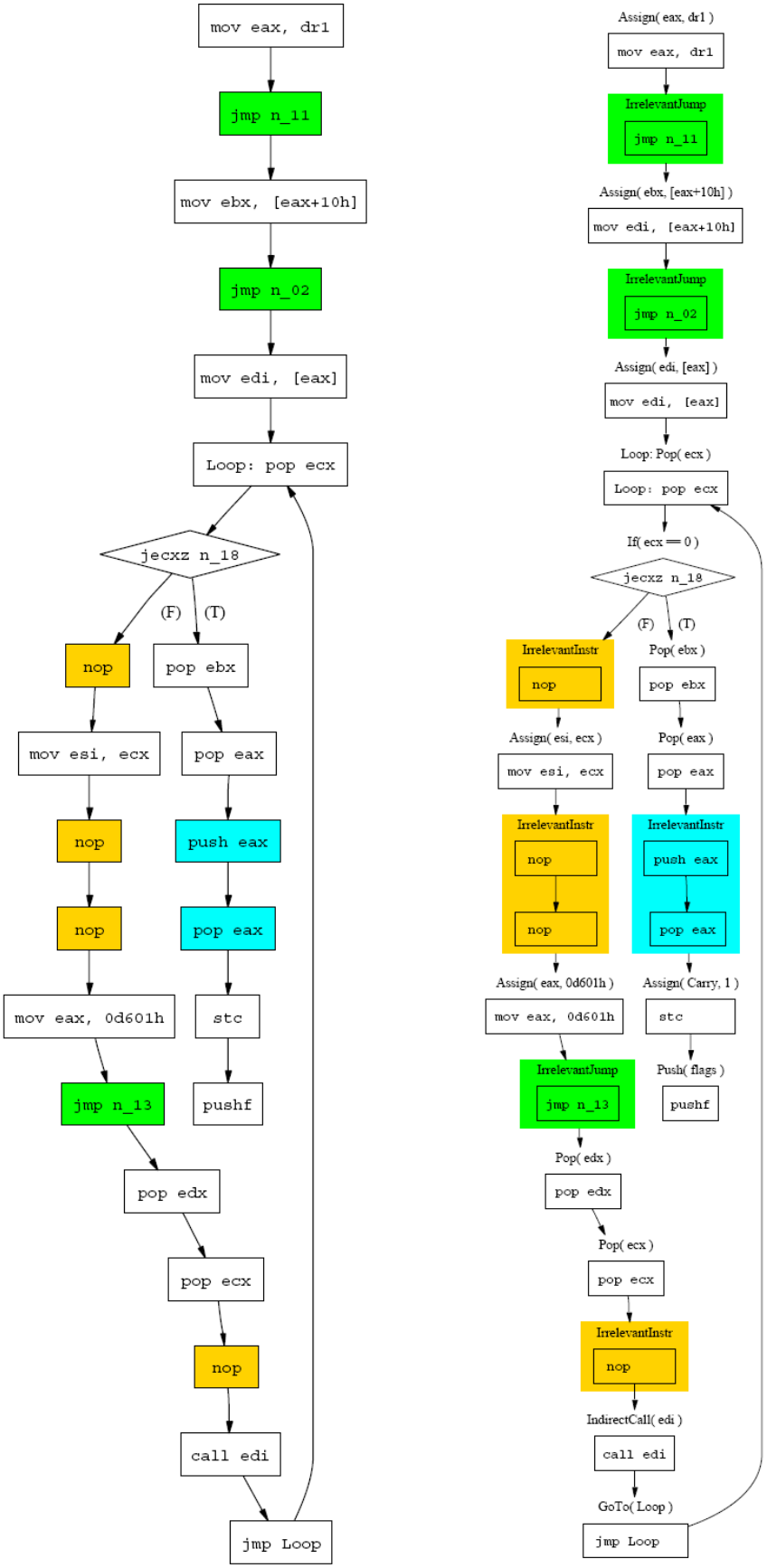
תהליך יצירת גרף זרימת הבקרה (CFG) של הקוד הנבדק, מתחיל בשני מסלולים מקבילים. האחד משמש לטעינת הגדרות דפוסי התנהגות קוד (Pattern Definitions) והשני לייצוג הקוד הבינארי הנבדק במבנה של גרף זרימת בקרה (CFG for the Executable). המסלולים מתאחדים בתהליך מסיים (Annotator) שתפקידו לקבל כקלט את ייצוג הדפוסים והגרף המייצג של הקוד וליצור גרף זרימת בקרה מפורש (Annotated CFG).

תהליך "טוען ספריית הדפוסים" (Pattern Definition Loader) טוען את ספריית דפוסי ההתנהגות ומייצג אותה בעזרת מבנה נתונים פנימי בתבנית ביניים. יש לשים לב שספרייה זו משמשת גם כקלט ליצירת האוטומט המייצג של הקוד הזדוני, וזאת במטרה להשתמש בבסיס משותף לצורך האפיון המופשט של הקוד הנבדק והקוד הזדוני.

"טוען הקוד הביצועי" (Executable Loader) טוען את הקוד הבינארי בנפרד עבור כל תהליך תוכנה (Procedure), והופך אותו לייצוג פנימי של גרף זרימת בקרה. התהליך עושה שימוש בתוכניות סטנדרטיות. האחת, IDA Pro, משמשת לפירוק קוד בינארי לפקודות המרכיבות אותו (Disassembler), והשנייה, CodeSurfer, משמשת לביצוע ניתוח סטאטי של הפקודות שהופקו כפלט מתוכנת הפירוק במטרה להבין את פעולת הקוד. תוכנה זו מאפשרת להבחין בפעולות חסרות משמעות בקוד כקפיצות שאינן שייכות או הוראות חסרות משמעות כ nop. לדוגמא, התוכנה יכולה לבצע בדיקה של הקוד בין שני נקודות בתכנית ולבדוק עבור כל משתנה חי בתחום בין הנקודות, אם ערכו משתנה. אם התשובה שלילית עבור כל המשתנים, ניתן להסיק שקטע הקוד בין שני הנקודות חסר משמעות ולכן, אולי ניתן להתעלם ממנו. עורכי המחקר כתבו ממשק בין התוכנות שתפקידו לקבל כקלט את פקודות הפלט מתוכנת הפירוק ולהביאן למבנה המתאים כקלט של תוכנת הניתוח הסטאטי.

המפרש (Annotator) מקבל כקלט את דפוסי ההתנהגות ואת גרף זרימת הבקרה של הקוד הבינארי ומוציא כפלט את גרף זרימת הבקרה המפורש (Annotated CFG), בו נמצא המידע המציין אם נמצא דפוס התנהגות מסוים בקוד הבינארי.

עבור כל צומת בגרף הזרימה של הקוד הנבדק, הגרף המפורש משייך דפוסי התנהגות שהוא מזהה בהתאם להגדרות הנמצאות בספריית דפוסי ההתנהגות. לדוגמה, תרשים 12 מציג את הקוד של ווירוס צ'רנוביל לאחר שעבר המרה למבנה CFG (משמאל) ולאחר שעבר תהליך פירוש והוצמדו לו הדפוסים שזוהו (מימין). במקרה זה הדפוסים שנמצאו הם קפיצות לא רלוונטיות ופקודות לא רלוונטיות. יש לשים לב שניתן להצמיד למספר צמתים בגרף המקורי דפוס אחד בגרף המפורש. משמעות הדבר שמספר קטעי קוד בסיסיים (צמתים) בתוכנית המקורית יכולות ביחד להוות דפוס התנהגות מוכר ויסומנו כך בגרף המפורש. דוגמה פשוטה לכך ניתן לראות בתרשים כאשר שתי פעולות, האחת push eax ומיד אחריה pop eax בגרף הקוד הנבדק מסומנות ביחד בגרף המפורש כ IrrelevantInst שמשמעותה פקודה חסרת משמעות.



תרשים 12: גרף בקרת הזרימה (CFG) של וירוס צ'רנוביל (משמאל) והתוצאה לאחר פירושו (מימין)

עורכי המחקר ביססו אותו על ארכיטקטורת x86 של אינטל ואף הגדירו שפה פורמאלית לצורך האנליזה הסטאטית לארכיטקטורה זו. למרות זאת, ניתן לאמץ את השיטה לארכיטקטורות שונות, תוך שמירה על העקרונות המנחים במאמר זה.

3.3.3.4 גלאי (Detector) הקוד הזדוני

הגלאי מקבל כקלט את גרף זרימת הבקרה (CFG) המפורש של הקוד הנבדק ואוטומט הקוד הזדוני (MCA). אם הדפוס המתואר באוטומט נמצא בגרף, הגלאי מחזיר את סט ההוראות המתאר את הדפוס, אחרת, הגלאי מחזיר תשובה שלילית. בצורה פורמאלית, אם נסמן את התכנית הנבדקת כ – P, את אוטומט הקוד הזדוני כ – A, את B_{All} כקבוצת כל הקישורים של משתנים ספציפיים ב – P למשתנים מופשטים ב – A ואת L כשפה המתקבלת מאוטומט, הגלאי קובע אם השפה

$$L(P) \bigcap \left(\bigcup_{B \in B_{All}} L(B(A)) \right)$$

אינה ריקה, עובדה המצביעה על המצאות קוד זדוני בקוד הנבדק.

כפי שציינו לעיל, אותו האלפבית של הדפוסים המופשטים צריך לשמש כקלט, הן ליצירת אוטומט הקוד הזדוני והן ליצירת גרף זרימת הבקרה המפורש של הקוד הנבדק. אינטואיטיבית, הגלאי מנסה למצוא דפוס המופיע, הן באוטומט והן בגרף הקוד. בפועל, הגלאי בודק אם קיימת הצבת משתנים, שתמצא דפוס זהה באוטומט הקוד הזדוני ובקטע בגרף זרימת הבקרה של הקוד שנבחן.

האלגוריתם בו נעשה שימוש מתבסס על אלגוריתם קלאסי [20] הבודק אם החיתוך בין שתי שפות אינו ריק. לצורך השימוש המעשי באלגוריתם במחקר זה, הוא הותאם לתמיכה בקישורים שונים של משתנים.

3.3.4 מבנה המחקר

המטרות העיקריות של המחקר והניסויים היו:

- בדיקת זמני הביצוע של אב הטיפוס המבוסס על השיטה המוצעת.
- מציאת שיעור הטעויות מסוג I (False Positive) וסוג II (False Negative).

קוד הדגימה הזדוני התבסס על ארבעה וירוסים שונים, Chernobyl, z0mbie-6.b, f0sf0r0, Hare. עבור וירוסים אלו נבנו אוטומטים מייצגים של הקוד הזדוני.

עבור כל וירוס, נוצרו תשע גרסאות הסוואה שונות המבוססות על השיטות שהוצגו בתחילת מאמר זה, בתוספת הגרסה הלא מוסוות של קוד הווירוס.

סביבת המבחן התבססה על מערכת הפעלה Windows 2000. תהליך הריצה עבור כל וירוס כלל את הרצת עשר הגרסאות של הווירוס מול האוטומט המייצג את הקוד של הווירוס. תהליך זה סיפק את נתוני הטעויות מסוג II, דהיינו, התהליך היה חייב לגלות את הווירוס בכל אחת מהגרסאות.

בשלב השני, עבור כל סוג ווירוס, הורצו גרסאות הווירוס מול האוטומטים המייצגים של הווירוסים האחרים (אלו שאינם מייצגים את הווירוס). תהליך זה סיפק את נתוני הטעויות מסוג I, דהיינו, התהליך היה חייב לאשר כל אחת מגרסאות הווירוס כנקייה.

תוצאות הריצה הניבו 0 טעויות מסוג I ו – II עבור התהליך. תהליך זה שימש גם לצורך מדידת הביצועים של המערכת. במדידת הביצועים התעלמו עורכי המחקר מזמן הביצוע של התוכנות החיצוניות ששימשו לפירוק הקוד והניתוח הסטאטי שלו. כחלק מהמחקר נערכו בדיקות סריקה של תוכנות בגדלים שונים. הבדיקה הניבה תוצאות ביצועים גרועים, במיוחד עבור יישומים גדולים בעלי מספר הליכים גדול. עורכי המחקר מציינים שיתייחסו לנושא הביצועים במחקר המשכי.

3.4 הגנה סטאטית על שער הכניסה מהרשת – Secure Web Gateway

מבוסס על דוחות (White Papers) של חברה העוסקת בתחום.

3.4.1 כללי

בשנים האחרונות אנו עדים למגמה הולכת ומתרחבת של שימוש ברשת (Web) לצורך התקפות על ארגונים ומחשבים אישיים לצורך חשיפת מידע, כלכלי או אחר, במטרה מוצהרת להפקת רווחים. הסיבה להפיכת ה Web לשדה מערכה עיקרי נובעת ממודעות התוקפים כי הגנות מבוססות חתימה (כאנטי ווירוסים) או מבני נתונים (כקטלוג כתובות אינטרנט, URL Categorization) אינן מספקות הגנה מספקת כנגד התקפות קוד עוין חבוי היטב, שמקורו באתרים לגיטימיים.

במקביל לתהליך העברת המערכה לרשת, התפתח תחום ההגנה לאיזורים אילו. חברות אבטחה שונות משקיעות משאבים רבים יותר לפיתוח הגנה מתאימה לסוגי האיומים החדשים, תוך התמקדות מיוחדת בנושא הגנה בפני התקפות זמן 0 (0-dey attacks), קרי, התקלות ראשונה בסוג התקפה שאינו מוכר, או נוקט בשיטות שאינן מוכרות.

3.4.2 האתגר

ארגונים גדולים ומוסדות תלויים יותר ויותר בזמינות הרשת לצורך עסקיהם המקוונים. רבים מהעסקים רואים ברשת את המקור העיקרי להכנסתם. לאלו, גניבת נתונים, זהויות פיננסיות או פגיעה באיכות השרות עלולה להוות איום ממשי על קיומו של העסק. מצד אחד, מערכות הגנה הדוקות, עשויות להוות מחסום מתאים בפני התקפות אולם, לפגוע באיכות השרות כזמן תגובה ארוך או שלילת פניות לגיטימיות כקוד עוין (Type I, False Positive), ומצד שני, מערכות הגנה רופפות עשויות לשפר את איכות זמן התגובה אולם להחמיץ קוד עוין (Type II, False Negative).

בעיה נוספת נובעת מאופי הקוד, המתחלף בתדירות גבוהה (בהחלפת דפי Web) ומורכב בדרך כלל מקוד תסריט, בו גמישות הכותב רבה, בעיה הדורשת פתרון בסריקת כל תוכן המגיע למחשב והתייחסות לקוד חדש, שטרם נראה בעבר.

מכאן שהאתגר הוא לספק הגנה מספקת בפני סוגי האיום האופיניים לרשת, סריקה של הקוד בזמן אמת בעת קבלתו, כל זאת תוך כדי מתן מענה יעיל ומהיר מספיק כדי להמנע מפגיעה קיצונית באיכות השרות.

חלק זה של התזה מתמקד בתיאור כללי של שיטת ההגנה של אחת החברות בתחום. לצורך כך אתמקד בתהליך אבחון של קוד Java Script, למרות שהתהליך המודגם עשוי לשמש גם בשפות תסריט אחרות או כל שפה המאפשרת תהליך של פירוק (Reverse Engineering) לצורך בחינת התנהגות הקוד.

3.4.3 תהליך הבדיקה

היות ומדובר במערכת הפועלת בתנאי עבודה מעשיים, אחת הדרישות הבסיסיות היא זמן תגובה (Latency) קצר ככל האפשר על מנת לא לפגוע בחווית המשתמש. בדיקה של כל האפשרויות הקיימות בכל קוד עשויה לצרוך משאבי מחשב רבים ולגרור זמני תגובה ארוכים. מסיבה זו, תהליך בדיקת הקוד מחולק לשלושה שלבים. נירמול הקוד, הסריקה המקדמית והסריקה הקפדנית.

3.4.3.1 נירמול הקוד

החופש באפשרויות כתיבת הקוד בשפות ה Web כ Java Script, פותחות בפני מפתחי הקוד הזדוני אפשרויות פשוטות להסתרת פעולות הקוד. פקודות כ charAt() ,unescape() ,fromCharCode() ואחרות מאפשרות לכותב הקוד לכתוב מחרוזות תווים, לכאורה חסרות משמעות, ולתרגם אותם בשלב הטעינה לדפדפן לקוד פעיל.

תרשים 13 מציג לדוגמה קוד מסוג זה.

```
<script>

var myobject ='Script' + 'ing.FileSystemObject';

var obj=new ActiveXObject(myobject);

obj.DeleteFile(unescape('%43%3a%5c') + 'boot.ini';);

</script>
```

תרשים 13: קוד Java Script עם פעולות חשודות חבויות

בשורה הראשונה נעשה שימוש באפשרויות שרשור המחרוזות של השפה לבניית ביטוי קוד בעל משמעות ובשורה השלישית נעשה שימוש בפקודת unescape() שתפקידה לשחזר את התווים המיוצגים על קוד Ascii הקסדצימאלי, המועבר כמידע לפונקציה.

היות ומגוון האפשרויות רחב ביותר, תפקיד פעולת נירמול הקוד היא להביא את הקוד, במידת האפשר, למצבו הנורמלי. במקרה של הקוד בתרשים, נקבל לאחר פעולת הנירמול את הקוד הבא:

```
<script>

var obj=new ActiveXObject('Scripting.FileSystemObject');

obj.DeleteFile('c:\boot.ini');

</script>
```

תרשים 14: קוד Java Script לאחר פעולת נירמול

3.4.3.2 סריקה מקדמית

תהליך הסריקה המקדמית מבצע בדיקה סטאטית מהירה של הקוד הנסקר, במטרה לאתר דפוסי קוד חשודים בביצוע פעולות לא חוקיות (בדומה לאנטי ווירוס). הבדיקה שטחית, מבוצעת מקומית בלבד ואינה מאתרת מתקפות מורכבות המשלבות פעולות בחלקי קוד באיזורים שונים. כמו כן הבדיקה אינה מסוגלת לעקוב דינאמית אחר התנהגות הקוד. אופי הבדיקה מבטיח זמן תגובה מהיר ולמרות חולשותיה המוצהרות, היא מצליחה לאתר ולהסיר כמות ניכרת של קוד זדוני.

תוצאת הבדיקה עשויה להיות אחת מהאפשרויות הבאות:

- הקוד נקי – הקוד נבדק ולא נמצא בו שמץ חשד ולכן הוא יאושר למעבר ללא בדיקות נוספות.
- הקוד זדוני – הקוד נמצא מכיל פעולות או דפוסים עויינים ולכן דרכו למחשב תחסם. דפוסים אלו בדרך זוהו בשלב מוקדם ואובחנו כזדוניים, בדומה לחתימת אנטי ווירוס.
- הקוד חשוד – הקוד נמצא מכיל פעולות או דפוסים החשודים עויינים ולכן יועבר לתהליך הסריקה הקפדנית לצורך השלמת הבדיקה. דוגמה לפעולה חשודה ניתן לראות בכתיבה ל Registry של מערכת ההפעלה חלונות או בפנייה לקובץ בדיסק.

לדוגמה, הקוד בתרשים 14 יזוהה בשל ביצוע נסיון פעולת מחיקת קובץ מערכת מהדיסק ובמקרה זה יוגדר כזדוני ופעולתו תחסם.

תהליך הסריקה המקדמית מסוגל לאשר או לשלול במהירות ובבטחון המצאות קוד זדוני באחוז ניכר מהקוד הנסרק, ולכן כמות הקוד המופנה לבדיקה מעמיקה בתהליך הסריקה הקפדנית, קטן משמעותית וזמן התגובה הכללי של המערכת קטן בהתאם ובזאת תרומתה העיקרית של הסריקה המקדמית.

3.4.3.3 סריקה קפדנית

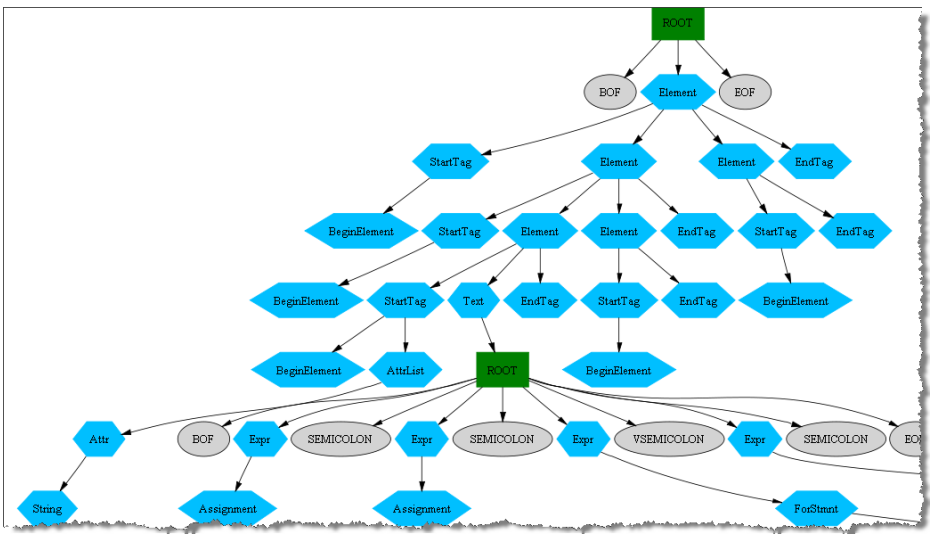
בחלק מהמקרים, תהליך הסריקה המקדמית אינו מסוגל לקבל החלטה אם הקוד עונה להגדרה של קוד זדוני. הדבר נובע, בדרך כלל, מסוג התקפה חדש ולא מוכר, המערב חלקים באזורים שונים בקוד. אם עולה חשד שהקוד מבצע פעולות המהוות חלק ממתקפה מורכבת על המחשב, הקוד עובר לתהליך סריקה קפדנית.

קיימים מקרים בהם כותב הקוד משתמש בקידוד מיוחד אותו פיתח. הכינוי לקוד מסוג זה הוא "קוד תוצרת בית" (Home-made Code). במקרים אלו יקשה על תהליך הנירמול להביא את הקוד למצבו הנורמאלי. לדוגמה, בקוד בתרשים 15 מוגדרת מחרוזת, לכאורה חסרת משמעות בשורה הראשונה. לאחר מכן מתבצעת לולאה בה נלקחת כל אות במחרוזת ומתורגמת לאות אחרת (במקרה זה על ידי תוספת ערך קבוע) ומשורשרת למחרוזת חדשה, הפעם בעלת משמעות. מחרוזת זו נכתבת לאחר מכן לאובייקט המסמך של הדפדפן כקוד פעיל. תהליך הנירמול יחמיץ קוד זה ולכן תהליך הסריקה המקדמית יתקשה לאתר את הדפוס זדוני, אולם, הוא יצליח להבחין בפעולות מעוררות חשד כשירשור וקידוד המחרוזת ולכן יעביר את הקוד לתהליך הסריקה הקפדנית.

```
<script>
a="*8Hxhwnuy*75qfslzflj*8I*77Of%7BfXhwnuy
*77*8J*5I*5F44*75pnqqkwfrj*75E*75%7C…";
b="";
for (i=0; i< a.length; i++)
{
      x=a.charCodeAt(i);
      b+=String.fromCharCode(x+16);
}
z=document.write;
z(b);
</script>
```

תרשים 15: קוד תוצרת בית - Home-made Code

תהליך הסריקה הקפדנית מתחיל בניתוח הקוד ובניית עץ זרימת בקרה, המתאר את כל התהליכים המתרחשים בקוד. העץ משמש כבסיס לניתוח התהליכים והתלויות בינהם. דוגמת עץ תהליכים מתואר בתרשים 16.



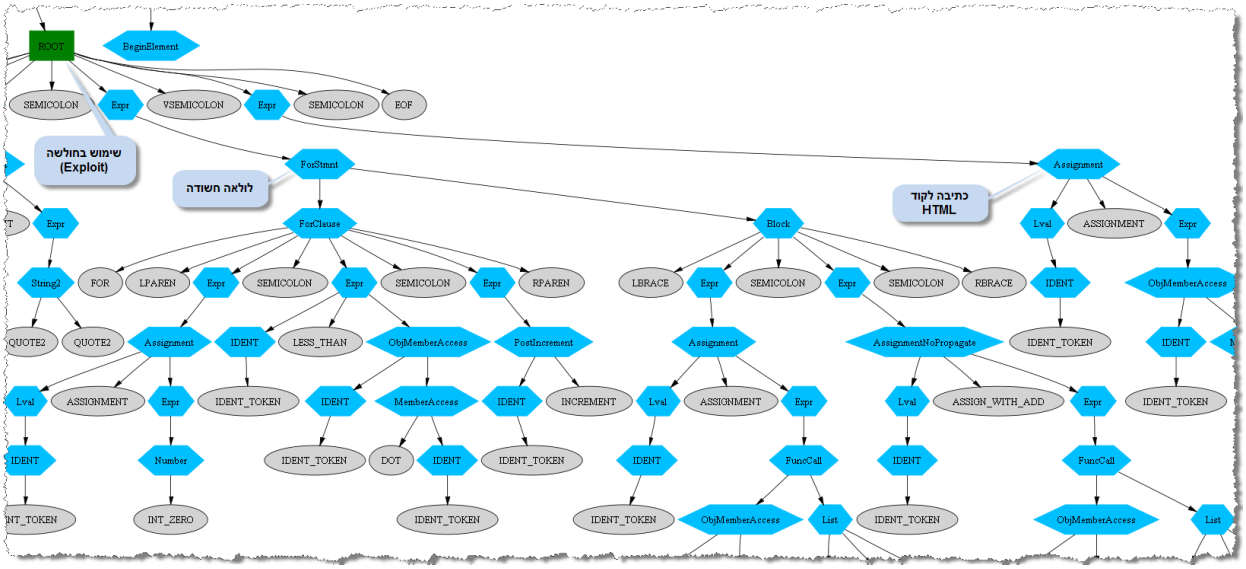
תרשים 16: תרשים חלקי של עץ תהליכים

ההנחיות לתהליך הסריקה הקפדנית מגיעות בצורת מערכת חוקים המגדירים התנהגות קוד חשודה או זדונית.

לדוגמה, נרצה לתאר את ההתנהגות המתוארת בתרשים 15. נבנה את החוק בצורה הבאה:

- אם נמצאה "לולאה (Loop) חשודה" הכוללת...
  - קוד הכותב לדף ה – HTML
- אזי מדובר בשימוש בחולשה (Exploit) של הדפדפן מסוג <זיהוי ניצול חולשה>

תהליך הסריקה יעבור על הצמתים בעץ וינסה לאתר דפוסים מוכרים. אם יתקל בקוד לולאה חשוד (על כך בהמשך) יסמן אותו בצומת המתאים. אם הקוד יתקל בנסיון כתיבה לדף ה – HTML, יבדוק אם אחד מאבות הצומת, בו הקוד התגלה, כולל לולאה חשודה. אם אכן זה המצב, יתאים התהליך את החוק המתואר מעלה בצומת האב של הלולאה החשודה. תוצאת תהליך זה מתוארת בתרשים הבא.

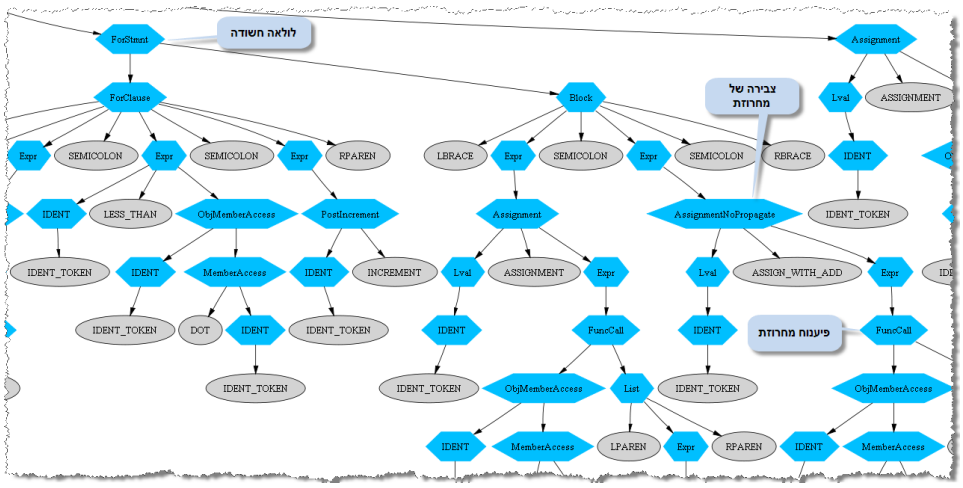


תרשים 17: מופע חוק בעץ התהליכים

בחוק המתואר למעלה עשינו שימוש במושג "לולאה חשודה". מערכת החוקים מאפשרת הגדרת חוקים המבוססים על חוקים אחרים. לדוגמה במקרה זה, נגדיר "לולאה חשודה" כחוק הבא:

- אם נמצאה "לולאה (Loop) " הכוללת...
  - פיענוח מחרוזות (חוק בפני עצמו הבודק שימוש לדוגמה ב charAt(), unescape(), fromCharCode()) וגם...
  - צבירה של מחרוזות (אף הוא חוק, לדוגמה הבודק שימוש לדוגמה ב דפוס כמו a+=b)
- אזי מדובר ב <לולאה חשודה>.

בתרשים הבא נמצא לדוגמה קוד העונה להגדרת לולאה חשודה:



תרשים 18: לולאה חשודה

המבנה המודולרי של מערכת החוקים מאפשר בניית חוקים חדשים על בסיס חוקים קיימים, בצורה המזכירה פיתוח תוכנה המאפשר שימוש בתהליכים בסיסיים לצורך יצירת תהליכים מורכבים יותר. שיטה זו מאפשרת תהליך פיתוח החוקים יעיל יותר, ללא חזרה על קוד קיים.

3.4.4 שימוש מעשי

המנגנון המתואר בסעיפים הקודמים מאפשר בניית מאגר חוקים עשיר, המתחדש בצורה עקבית עם חשיפת חולשות חדשות של דפדפנים. יתרון השיטה הוא ביכולת לנתח התנהגות קוד, ללא מידע מוקדם על דפוס זה או אחר, למעט מידע על חולשות הדפדפן והדרכים לנצל אותן. זאת בניגוד לאנטי וירוסים למינהם או כל אמצעי הגנה המתבסס על מידע מוקדם. חסרונה הוא בתלות במומחים המנתחים את התנהגות הקוד ומוסיפים חוקים חדשים על סמך ניתוח זה.

לעיתים התנהגות הקוד עשויה להיות לגיטימית לחלוטין. כאן בא לידי ביטוי תפקידו של מנהל המערכת כאחראי על בחירת החוקים להחלה על משתמש זה או אחר וזאת בהתאם למדיניות הארגון. ארגון עשוי לקבל התנהגות מסויימת כלגיטימית, וזאת למרות המצאות חוקים המאתרים אותה, ולאשר אותה למרות הגילוי.

המנגנון המתואר מוסיף רובד נוסף במנגנוני ההגנה, שאינו תלוי במידע מוקדם למעט ידע על חולשות של לקוחות קצה של הרשת, ובכך מאפשר גילוי קוד חדש, המנסה לנצל חולשות, למרות שלא נתקל בו בעבר.

3.5 מאמרים נוספים בתחום המחקר של עבודה זו

בתחום העוסק בזיהוי טיפוס קבצים על פי תכולתם לא נרשמה התקדמות משמעותית בשנים האחרונות. בשונה מהעיסוק המסיבי בתחום זיהוי קוד זדוני, נראה שנושא זה נמצא במקום נמוך יותר בסדר העדיפות של החברות העוסקות בנושא. לרוב, מבוצע זיהוי סוג הקובץ על סמך סיומת הקובץ או חתימה דיגיטלית קבועה ידועה מראש. כפי שראינו, שיטת זיהוי זו מאפשרת לגורם עויין להערים על מערכת ההגנה בקלות יחסית. שיטה נוספת מתבססת על סט חוקים ידוע ומוגדר מראש עבור כל סוג של קובץ. הבעייה העיקרית הנובעת מסוג זיהוי זה היא בעיקר כמות המשאבים הנדרשת. החל ממומחיות של החוקר בסוג הקובץ, אותו הוא מנתח, המטרה לקבוע את סט החוקים המתאים לטיפוס, וכלה בזמן האבחון עצמו, עת משתמשים בסט החוקים על מנת לאתר דפוסים מוכרים. כיום קיים מספר סוגים רב של קבצים. בכל יום, ככל שנוספות תוכנות חדשות לשוק, מתווספים סוגים נוספים, עובדה המקשה אף היא על קצב העידכון.

מחקרים מסויימים נערכו באקדמיה במטרה לפתח שיטות אוטומטיות ליצור מאפיינים לטיפוסים שונים במטרה לחסוך את שלב המחקר הספציפי עבור טיפוס זה או אחר. הכוון העיקרי הוא בנסיון ולאתר דפוסים מייצגים של הטיפוס בתכולת הקובץ באופן אוטומטי עבור כל טיפוס.

Kolter et al. [21] בחנו שימוש ב N-Grams בעלי אורך קבוע כמאפיינים לצורך איתור קוד זדוני בקבצים ביצועיים, ללא מידע מוקדם על קוד זה. שיטתם התבססה על טכניקות איחזור מידע משולבות בטכניקות אפיון טקסט. בדומה לגישת המחקר בעבודה זו, הם עשו שימוש במדגם מייצג קבצים, תמימים וזדוניים לצורך מציאת המאפיינים היחודיים של הקוד הזדוני, לעומת הקוד התמים. תהליך המחקר כלל את המרת הקבצים לייצוג טקסטואלי הקסדצימאלי של הקוד הבינארי. בשלב הבא חושבו כל ה N-Grams בגודל 4 בקבצים והתוצאה כללה כ – 255 מיליון יחודיים. N-Grams יחודיים. המשך התהליך התייחס לכל N-Gram כתכונה בינארית, קרי, נמצא בקוד ואז ערכו 1, אחרת 0. המחקר לא כלל התייחסות לכמות היחסית או המוחלטת של N-Gram בכל קובץ. שלב המחקר הבא כלל את בחירת 500 ה N-Grams המשמעותיים ביותר, בהתבסס על נוסחת IG (Information Gain). על מידע זה נוסו מספר שיטות למידה שונות שהתבססו על מזהים שונים כ Naive Bayes, SVM (Support Vector Machine), ועצי החלטה (Decision Trees). עצי ההחלטה הניבו את התוצאות הטובות ביותר וכצפוי, שימוש במדגם מייצג גדול יותר שיפר אף הוא את התוצאות.

Kumar Dash et al. [22] לוקחים את הגישה המוצגת ב [21] צעד אחד קדימה ומציעים שימוש ב N-Grams באורך משתנה לאיתור קוד זדוני. במאמרם הם משתמשים במינוח "מקרה" לתיאור N-Gram בעל אורך משתנה כלשהו, בעל משמעות, מתוך הרצף הנבדק. במסגרת המחקר נאספו מקרים משמעותיים מ 250 גרסאות שונות של תוכנות טוענות ווירוסים (Virus Loaders) ומ 250 קבצים ביצועיים נקיים. כל הקבצים עברו המרה לקוד הקסאדצימלי ביצוג טקסטואלי. בשלב הבא שימשו ההמרות ליצירת עץ Trie ממנו נאספו "המקרים" השכיחים ביותר. להוכחת נכונות מחברי המאמר עשו שימוש בבדיקת K-Fold Cross Validation בה מבצעים בכל פעם בדיקה של תת קבוצה מהמדגם לעומת שאר המדגם. במקרה זה בוצעו 10 חזרות. לטענת מחברי המאמר, שימוש ב N-Grams בעלי אורך משתנה משפר את ביצועי האלגוריתם לעומת אורך קבוע היות ובמקרה האחרון נאבד מידע חיוני של N-Grams משמעותיים הארוכים יותר מהגבלת אורך ה N-Gram, כפי שנקבעה על ידי האלגוריתם. התוצאות אכן מראות פחות טעויות מטיפוס I.

במאמרם, Irtfan Ahmed et al. [23] מציעים 2 גישות שונות לזיהוי טיפוס קובץ. גישה ראשונה עושה שימוש ב Cosine Similarity להשוואת דפוסים של רצפי בתים בעלי תדירות גבוהה. לצורך יצירת המאפיין של טיפוס מוצאים, עבור כל הקבצים השייכים לו במדגם, דפוסים שונים של רצף בתים ומנרמלים אותם. ממאגר דפוסים זה נלקחים אלו בעלי השכיחות הגבוהה ביותר. לטענת המחברים, דפוסים אלו מייצגים את הטיפוס נאמנה בעוד דפוסים בעלי שכיחות נמוכה יותר פוגעים בדיוק הזיהוי. יתרון נוסף של בחירה בדפוסים השכיחים בלבד הוא בכמות המידע הנדרשת לשמירה עבור כל איפיון של טיפוס. לטענת המחברים, התוצאות שהתקבלו מדויקות יותר במרבית הטיפוסים לעומת מחקרים קודמים שהתבססו על השוואה בעזרת Mahalanobis Distance המוגדר כמדד מרחק סטטיסטי המודד קשר הדדי בין משתנים שונים. בנוסף, הקטנת כמות הדפוסים במדגם כמעט ואינה משפיעה על דיוק התוצאה, עד לגבול מסוים כמובן. לאור הכמות הקטנה יחסית של דפוסים והדיוק היחסי המתקבל, מהירות האלגוריתם גבוהה משמעותית יחסית לקודמיו. מסקנה נוספת שעולה מהמחקר היא שיעילות האלגוריתם תגדל משמעותית עם הגדלת גודל ה – N-Grams של הדפוסים הנבדקים.

גישה שונה, אותה מציע המאמר, עושה שימוש בתהליך הפרד ומשול (Divide and Conquer). התהליך מורכב משילוב של איפיון ראשוני וחלוקה לאשכולות של קבצים בעלי מאפיינים דומים, ללא שום קשר לסוגם. התוצאה המתקבלת היא של אשכולות שונים, כאשר בכל אשכול קבצים מסוג דומה או זהה. בשלב השני, אם כל הקבצים באשכול הם מאותו הטיפוס, האשכול יסומן על ידי טיפוס זה, אחרת יתבצע על האשכול מיון נוסף המתבסס על אלגוריתם רשתות ניורוניות (NN - Neural Networks) המחלק את הקבצים באשכול לפי הטיפוס שלהם. אלגוריתם זה מאומן לכל סוג של אשכול בצורה שונה, המתאימה לאופי הטיפוסים באשכול הנדון. שיטה זו של הפרד ומשול מאפשרת לקבל אשכולות טיפוסים בעלי מאפייני פיזור בתים דומים ולבצע בדיקה מדוקדקת ויעילה יותר על כל אשכול וזאת לאור הכמות הקטנה יותר של טיפוסים עימה צריך להתמודד אלגוריתם המיון בשלב השני. המחקר העלה שקיימת כמות אופטימאלית של אשכולות (6 אשכולות במקרה של מחקר זה) שתניב את התוצאות היעילות ביותר. מסקנות המאמר הם שגישת הפרד ומשול הנקוטה משפרת, במקרים מסויימים את דיוק האיפיון של טיפוסים ובמקרים אחרים לא לעומת אותם אלגוריתמים

מסויימים שאינם נוקטים בגישה זו. לדוגמה, במקרה של אלגוריתם ה NN השימוש בגישת הפרד ומשול הניב תוצאות מדויקות ב 20% -7% לעומת אותו האלגוריתם ללא שימוש בגישה זו עבור 5 טיפוסים מסויימים. עבור 5 הטיפוסים הנותרים התקבלו תוצאות פחות מדויקות. מחברי המאמר מייחסים זאת לעובדה שעבור 5 הטיפוסים האחרונים נמצאו קבצים באשכולות שונים, כנראה לאור מספר גדול של פיזור בתים, דבר שגרם להחמצתם בשלב הלימוד של האלגוריתם.

Martin Karresand ו - Nahid Shahmehri מטפלים במאמרם [24] בבעיה של זיהוי טיפוס של קוד תוך התבססות על מקטעי קוד חלקי. מטרתם היא למצוא דרך יעילה לזיהוי הטיפוס במקרה של מקור מידע חלקי כדיסק קשיח שניזוק, קטע קוד בזיכרון או כל סוג אחר של מידע דומה. המחקר מתבסס על מאמר קודם של אותם כותבים, המציג שיטה המכונה Oscar לזיהוי סוג קובץ תוך התבססות על המבנה הפנימי של מקטעי קוד שלו. בשיטה המקורית מחשבים את התפלגות תדירות הבתים עבור כל המקטעים המשמשים כמדגם. עבור כל בית, מחשבים את הממוצע ואת סטיית התקן. שלושת מדדים אלו יוצרים יחידת מידע הנקראת Centroid. ההשוואה נעשית לאחר מכן מול המידע המחושב מקטע לא מוכר. אם מרחק תוצאה זו מה - Centroid קטן מסף מסוים, הקטע מזוהה כשייך לטיפוס המיוצג על ידי ה Centroid.

במאמר זה מבקשים להרחיב את השיטה על ידי הוספת מדד של קצב השינוי (RoC – Rate of Change) בערכי בתים עוקבים בתוך מקטעי הקוד. ההשוואה מול מרכז הCentroid נעשית בעזרת 2 מדדים שונים, הראשון הוא 1-norm (הידוע גם בכינויו Manhattan Distance) על קצב השינוי של קטעי המידע והשני מתבסס על תדירות ההתפלגות של של קצב השינוי במקטע, אותו יש לזהות.

לצורך המחקר יצרו עורכיו קובץ אחד בגודל של כ – 72 MB משירשור של 57 קבצים מטיפוסים שונים, וזאת על מנת לחקות מבנה של מידע מאוחזר מדיסק קשיח פגום. הנסיונות משקפות הבדל בין התוצאות המתקבלות מקבצים מכווצים לאילו שלא, במיוחד במדד סטיית התקן של קצב השינוי. התוצאות שהתקבלו במסגרת המחקר לא היו אחידות עבור הטיפוסים השונים. לדוגמה, מימוש ספציפי עבור קבצי jpeg הניב אחוז גילוי שעלה על 86.8% אולם במקביל התקבלו כ – 20% False Positives. הוספת מודול מיוחד עבור קבצי jpeg, אשר בדק מאפיינים יחודיים לטיפוס זה, שיפר את המדד האחרון עד קרוב למצב בו לא נמצאו False Positive כלל. במקרה של קבצים ביצועיים התוצאות שהתקבלו היו פחות מבטיחות, גילוי בין 45% ועד 85% ובמקביל נתון של עד 35% False Positives . אחד מחסרונותיה של השיטה המוצעת במאמר נובע מהצורך מהתייחסות מסויימת לסוג טיפוס זה או אחר, לדוגמה בקבצי jpeg במחקר, עובדה הפוגעת ביכולת השימוש בשיטה כחלק ממנגנון אוטומטי ליצור חתימות טיפוס.

Wei-Jen Li et al. מציעים במאמרם [25] שיטה לזיהוי טיפוס של קבצים תוך שימוש באנליזה סטטיסטית יעילה, המתבססת על 1-gram (NGram באורך 1) של תוכנם הבינארי. הם ממשיכים את הקו שהותווה במאמר של McDaniel and Heydari ([18]) אולם טוענים לשיפור התוצאות המתקבלות על ידי עידון החלוקה בין הטיפוסים השונים ששימשו במדגם ואימוץ אסטרטגיה של קיבוץ

(Clustering) . שינוי נוסף ביחס למאמר המקורי, אותו הם מציינים, הוא בשיטת הנירמול של התוצאות ביחס לגודל קובץ הנבדק, בשונה מהמאמר הקודם בו השתמשו במדגם הגדול ביותר שעלה, כבסיס לנירמול התוצאות. במאמר, משתמשים המחברים במדגם של קבצים מייצגים לכל טיפוס, עבורו רוצים למצוא "טביעת קובץ" (Fileprint). לצורך המאמר נעשה שימוש ב 800 קבצים שונים ששימשו לאיפיון 6 טיפוסים שונים שכללו בתוכם DLL ו – EXE כקטגוריה אחת וקבצי MS Office כקטגוריה אחת. האיפיון נעשה בשני אופנים. במקרה אחד נסרקה כל תכולת הקבצים ובמקרה השני, רק חלק מהקובץ נסרק. במקרה האחרון נערכו מספר נסיונות על 20 – 1000 הבתים הראשונים של כל קובץ, הכוללים בתוכם גם את מספרי הקסם והראשית של הקבצים.

במסגרת המחקר בוצעו שלושה סוגי ניסויים. בראשון יוצרה טביעת קובץ עבור כל קבוצת קבצים שייצגו טיפוס. בשני, כל קבוצה מייצגת לטיפוס חולקה ל – 5 תת קבוצות, מתוכם נבחרה הקבוצה המייצגת בצורה הנאמנה ביותר את הטיפוס וממנה נוצרה טביעת הקובץ. במקרה השלישי הקצינו את הניסוי השני ובמקום קבוצת קבצים מייצגים, נבחר קובץ אחד בלבד כמייצג את הטיפוס.

התוצאות שהתקבלו היו טובות באופן כללי. עבור המקרה הראשון, רמת הזיהוי שהתקבלה היתה בין 100% - 80.4%, כתלות באורך הקטע הנסרק וסוג הקובץ. כמובן שסריקת קטע קטן מקובץ מועדת לבעיות של ראשית קובץ פגומה או עריכה למטרת הסוואת סוג הקובץ. מסיבה זו נערך נסיון נוסף של סריקת כל תכולת הקבצים. וההצלחה במקרה זה נעה בין 88.3% - 62.7%. במקרה השני התקבלו תוצאות טובות יותר, אשר נעו בין 100% - 90% ו – 94.5% - 76.8% בהתאמה. במקרה השלישי התקבלו אמנם התוצאות 100% - 86.9% ו – 98.9% - 77.1% בהתאמה, אולם באופן כללי התקבלו תוצאות טובות יותר מאשר בשני המקרים הראשונים.

בעבודת התזה של Ryan M. Harris בנושא שימוש ברשתות ניורונים לצורך זיהוי טיפוס קבצים [26], המחבר ביקש לבדוק אם שימוש ברשתות ניורונים כמערכת לומדת תספק מערכת יעילה יותר, מהשיטות שנחקרו עד כתיבת התזה, לצורך זיהוי טיפוסי קבצים וכן, אילו טיפוסים יספקו תוצאות זיהוי טובות מאחרים. המחקר התמקד בטיפוסים בעלי תוכן בינארי מובהק כקבצי תמונה, קבצים מכווצים וקבצים ביצועיים והתעלם במכוון מטיפוסים בעלי תוכן טקסטואלי, הניתן לניתוח, לטענת המחבר, על ידי אלגוריתמים ידועים. בפועל נבחנו אך ורק טיפוסי תמונה כ - JPEG, PNG, BMP, GIF, TIFF כאשר 13,000 קבצים שימשו כקבוצה מייצגת של הטיפוסים לצורך בניית הרשת ו – 1,300 קבצים שימשו כקבוצת בדיקה. כל קובץ חולק למקטעים של 512 בתים ולצורך המדגם נלקחו 10 המקטעים הראשונים בלבד. על מקטעים אלו נערכו שתי בדיקות. בבדיקה הראשונה נלקח כל בית במקטע כקלט לניורון אחד ברשת (Raw Filtering) ובשניה נספרו מספר המופעים של כל בית בארבעה מקטעים, נתון ששימש כקלט לניורון ברשת (Character Code Frequency). ההנחה היתה שהבדיקה מהסוג הראשון מתאימה לקבצים בהם קיימת מחזוריות מסויימת לרצף בתים, בעוד השנייה מתאימה יותר לטיפוסים בהם רצף הבתים אינו סדיר. הניסויים שנערכו במסגרת התזה לא הראו הצלחה גדולה ועבור השיטה הראשונה התקבלה הצלחה בזיהוי של 50% - 1%, תלוי בסוג הנבדק, בעוד בשיטה השנייה התקבלה הצלחה בזיהוי של 60%-0%.

המאמר של Erbacher et al. [27] עוסק בניתוח תוכן קבצים אולם מתמקד בניסיון לאתר טיפוסים של מבני נתונים/סוגי קוד בתוך הקובץ. המאמר מתאר שימוש בהתפלגות הבתים והסתברות תוך שימוש בחלון בגודל קבוע הנע על פני הקוד לצורך איתור מהיר של קוד יוצא דופן בתוך קובץ. לצורך אפיון הקוד משתמשים במבחנים סטטיסטיים שונים. המאמר פוסל מספר מבחנים סטטיסטיים ומציין אחרים כיעילים יותר בזיהוי טיפוס הקוד. הנסיונות בוצעו על מספר קטן יחסית של קבצים, כחמישה קבצים שונים על כל סוג קובץ, סה"כ 7 סוגים שונים, כך שהתוצאות מרמזות אולי על פוטנציאל אפשרי, אולם אינן חד משמעיות.

המאמר של Moskovitch et al. [28] עוסק בשיטות למציאת קוד זדוני, על פי ניתוח המבנה ומציאת מאפיינים לקוד זה על ידי אנאליזה המבוצעת על מדגם קבצים מייצגים לקוד זדוני ומדגם לקבצים הידועים כנקיים. עורכי המחקר מציפים את בעיית הסווג השגוי המתקבל בעת אפיון מחלקות שאינו מאוזנות בגודלן בעת שימוש ברוב המסווגים המקובלים. בעייה זו אופיינית במיוחד למקרים בהם שכיחות העצמים השייכים למחלקה מסויימת גדולה משמעותית ביחס למחלקות האחרות. במקרים קיצוניים ניתן להגיע למצב בו כל כל העצמים יסווגו כשייכים למחלקה הגדולה, תוך התעלמות מנוכחות המחלקות הקטנות יותר. המחברים מציינים כי, בניגוד למקרים רבים אחרים, הבעיה בה דן המאמר אינה נעוצה באיזון המדגם אלא בתנאים האמיתיים בהם מתנהל המסווג, קרי, מטרתם אינה מציאת אלגוריתם פיצוי למדגם לא מאוזן, אלא מציאת מבנה אופטימאלי של מדגם שישיג את התוצאות הטובות ביותר במימוש על מידע אמיתי.

היות והמחקר עוסק בקוד בינארי, עורכיו מציעים המרה של הקוד לטקסט קריא, ממנו מאוחזרים N-Grams בגדלים שונים, כל אחד מקביל למלה במרחב טקסטואלי. במחקר מציע שימוש בשיטות הלקוחות ממערכות איחזור מידע לצורך קיטלוג טקסט, במטרה למצוא שיטה אוטומטית לזיהוי קוד הזדוני. לצורך המחקר נעשה שימוש במעל ל 7500 קבצים הידועים כמכילים קוד זדוני ומעל 22,500 קבצים תמימים.

המאמר מעלה מתמקד במתן תשובות למספר שאלות כ:

- גודל מדגם אופטימאלי.
- גודל ה N-Gram האופטימאלי.
- יצוג ה N-Grams האופטימאלי, קרי, בחירה במדד תדירות המילה במסמך (TF) או גרסתו המורחבת למדד המשלב גם את תדירותו באוסף כל המסמכים (TFIDF).
- כמות אופטימאלית לבחירה מקדמית וסופית של מאפיינים והדרך הטובה ביותר למציאת מאפיינים אלו.
- המסווג האופטימאלי לקבלת כמות מינימאלית של טעויות מטיפוס I ו II, ז"א, רמת דיוק גבוהה.
- אחוז החלוקה בין הקוד הזדוני האופטימאלי במדגם שינפק את התוצאות הטובות ביותר עבור נתונים אמיתיים.

המחקר כלל שני מסלולי ניסויים. במסלול הראשון הייתה מטרת מחברי המאמר למצוא תשובות לשאלות המובאות, למעט האחרונה. מסלול הניסויים השני ביקש לתת מענה לשאלה האחרונה, בעיית המחלקות הלא מאוזנות.

ממצאי המסלול הראשון העלו ששימוש ב TF על 5500 מאפיינים ראשוניים, הניב את רמת הדיוק הגבוהה ביותר. שיטת בחירת קבוצת המאפיינים הסופיים נבחנה כנגזרת של הכמות שלהם. באופן כללי התוצאה הטובה ביותר התקבלה בעזרת Fisher's Score המתבסס על ערכי ממוצעים וסטיות תקן מול המאפיין. רמת הדיוק הטובה התקבלה כבר עבור 50 מאפיינים בלבד וכן עבור 300 מאפיינים. בחירת המסווג בוצעה תוך השוואת המועמדים על בסיס כל המדדים שצויינו לעיל. המסווגים שסיפקו את כמות הטעויות המינימאלית מסוג I ו II הם BDT (Boosted Decission Trees), DT (Decission Trees) ו ANN (Artificial Neural Networks). מדדי ה SVM (Support Vector Machines) סיפקו כמות קטנה של טעויות מסוג I אולם אכזבו במספר הטעויות מסוג II.

התרומה העיקרית של המאמר מתקבלת ממסלול הניסויים השני, המטפל בבעיית חוסר האיזון בגודל המחלקות, קרי, מענה על השאלה מהו היחס האופטימאלי בין קוד תמים לקוד זדוני במדגם המשמש לבניית המאפיינים. מענה על שאלה זו הוא בעל משמעות מעשית חשובה היות והוא מספק כלי יעיל לבחירת יחס מחלקות מתאים למדגם במטרה לקבל תוצאה סופית המשקפת בצורה אמינה את היחסים בין קוד תמים וזדוני בפועל. כבסיס לנסיון זה השתמשו מחברי המאמר בפרמרטרים היעילים ביותר שהתקבלו בתוצאת הניסוי הראשון. מציאת המאפיינים בוצעה על 5 חלוקות שונות שונות ביחסים בין קוד תמים וזדוני (16.7%, 33.4%, 50%, 66.7%, 83.4%), בעוד שהבדיקות עצמן בוצעו על 17 רמות חלוקה שונות, סה"כ 85 ריצות ניסוי שונות. תוצאות הריצה השונות באו לידי ביטוי במדדים של טעויות מטיפוס I, טעויות מטיפוס II, שילוב ביניהם המשקף את דיוק תוצאות הריצה ומדד נוסף, g-menas, המעיד על המסווג אם הוא לוקה בחולשה בזהוי איום אמיתי או קוד תמים.

בשלב הראשון בניסוי זה, נבחרה חלוקה בת 10% קוד עויין לצרכי ריצה. לטענת עורכי המחקר, חלוקה זו מיצגת באופו מעשי את המצב בפועל. רמת הדיוק שהתקבלה הייתה מעל 95% במקרה בו אחוז הקוד הזדוני במדגם לבניית המאפיינים היה 16.7%. המסקנה הכללית שהתקבלה היא שאחוז קוד זדוני בתחום 10%-40% במדגם יספק את התוצאות האופטימליות בחלוקה אמיתית.

בשלב השני נבדקה השפעת שינוי אחוז הקוד הזדוני, במדגם ובקוד לצרכי הריצה, על דיוק התוצאות שהתקבלו. המסקנה שהתקבלה היא שבחירת אחוז דומה של קוד זדוני בשתי החלוקות תספק את התוצאות המדוייקות ביותר, וזאת על בסיס הפרמטרים האופטימליים שהתקבלו בניסוי הראשון.

פרק 4 שימוש במערכת מודלים סטטיסטיים על N-Grams כבסיס לזיהוי טיפוס קובץ

בפרק זה אציג שיטה ליצור חתימה מזהה, לטיפוסי קבצים שונים, המתבססת על ניתוח סטטיסטי כלשהו של מופעי N-Grams בקוד הקובץ הנבדק. בסיום הפרק אדגים את השיטה תוך התבססות על על ניתוח מספר המופעים של N-Gram בהשוואה למדגם המייצג של הטיפוס.

4.1 תקציר

זיהוי טיבו האמיתי של קובץ הוא מרכיב הכרחי במנגנון ההגנה על מערכת המחשב ולכן מושקע מאמץ רב בהסוואת אופיו של הקובץ הזדוני במטרה להערים על מנגנון זה. חברות העוסקות באבטחת מחשבים ותוכנה משקיעות משאבים רבים בפיתוח שיטות זיהוי תחת הכותרת " True Content Type analysis", אולם אלו מסתמכים לרוב על שיטות זיהוי קלות יחסית להערמה ונראה כי אין עדיין מענה מעשי לבעייה. רוב העיסוק בנושא כיום הוא אקדמי בעיקרו ועדיין אין תוצאות מעשיות משמעותיות. חלק זה של התזה מציג שיטה ליצירת חתימה מאפיינת לטיפוסי קבצים שונים, המבוססת על ניתוח סטטיסטי של מופעי N-Grams בגוף קבצי מדגם מאפיין לכל טיפוס. השיטה אינה מחייבת בחירה במדד סטטיסטי זו או אחר, ואף אינה מחייבת שימוש ב N-Gram בגודל מסויים, אלא מהווה כר לביצוע מספר נסיונות רב בשיטות שונות, במטרה למצוא תכונות יחודיות המבדילות בין המבנה האנטומי של טיפוס זה או אחר. במסגרת התזה מודגמת השיטה בעזרת בדיקה בסיסית של מספר המופעים של N-Grams עבור שישה טיפוסי קבצים שונים ומוצגות תוצאות הזיהוי המבטיחות שהושגו בעזרת בדיקה זו.

4.2 מבוא

בחלקים קודמים של תזה זו ראינו את הסיכון הגלום בהתקפה של גורמים עויינים על מערכות מחשב שונות, החל מגרימת נזק למערכת וכלה בגניבת מידע מודיעיני, כלכלי או אישי. הכלים המשמשים את התוקפים מגולמים, בדרך כלל, בקטעי קוד המבצעים פעילות ללא ידיעתו ובניגוד לרצונו של מנהל המערכת. על מנת להשיג את מטרתם, על התוקפים להחדיר קטעי קוד אלו למערכת המחשב. בשלב החדירה ינקוט הפורץ בכל דרך שבידו לגרום למערכת ההגנה של המחשב להאמין שהקובץ שמוריד , מתקין, מעתיק או פותח מפעיל המחשב הוא קובץ לגיטימי ללא כוונות זדון. אחת השיטות להערים על מערכת ההגנה היא "להעמיד פנים" שאתה קובץ מטיפוס אחד בעוד שהנך למעשה קובץ מטיפוס אחר.

מערכות הגנה רבות מסתמכות על טיפוס הקובץ, אותו הן בודקות במטרה להחליט על המשך הטיפול בו, למשל מניעת כניסתו באופן מוחלט בהתאם לרצון מנהל המערכת, סריקתו למציאת ווירוס או אישור כניסה ללא בדיקה נוספת. לדוגמה, מערכת עשויה להחליט כי לא תבדוק קבצים מטיפוס תמונה אולם תסרוק קבצים ביצועיים למציאת ווירוס. אם המערכת תסיק, בטעות, כי הקובץ הביצועי המנסה למצוא עצמו למערכת, הוא מטיפוס תמונה, היא תאפשר את כניסתו ללא סריקה. משיקולי יעילות, מתחילה האנליזה של הקוד הנבדק במנגנון זיהוי סוג הקובץ, וזאת למטרת חיסכון בפעולות מיותרות בהמשך.

אחת משיטות ההטעייה הבסיסיות ביותר, למשל, היא לשנות את הסיומת (ext) של הקובץ היות ובמערכת הפעלה חלונות הסיומת משמשת לזיהוי סוג הקובץ ולבחירת הישום שיפעיל אותו. אולם, מערכות אבטחה נוקטות, בדרך כלל, בדרכים מתקדמות יותר לזיהוי ובוחנות את תוכן הקובץ לצורך קביעת סוגו. אנאליזות המתבססות על ניתוח הקוד בגוף הקובץ נמצאות תחת הכותרת "True Content Type Analysis".

אולם נכון להיום, אנאליזות אלו מתבססות בעיקר על חתימת טיפוס דיגיטאלית סטטית קשיחה. מספר הבתים הראשון בכל קובץ מאפיינים אותו ומשמשים מערכות הפעלה שונות, לדוגמה UNIX, בהחלטה על סוג הישום שיפעיל אותו. בתים אלו נקראים מספרי קסם "Magic Numbers" או כותרת הקובץ "File Header" כשהשם נקבע על פי אורכו. לדוגמה, קובץ Java מקומפל ישא בראשיתו את הקוד "CAFEBABE" בעוד קובץ מטיפוס gif יזוהה על ידי הקוד "GIF89a" או הקוד "GIF87a".

שיטת זיהוי זו יעילה ביותר במקרה ותכולת הקובץ נשארה ללא שינוי, אולם תיכשל אם שאר תכולת הקובץ אינה תואמת לחתימתו. קל לראות ששינוי מספר הבתים הראשונים של הקובץ קל ביותר ואינו דורש מאמץ רב, ובכך חולשת שיטת זיהוי זו. שינוי של מספר בתים בראשיתו של הקובץ, לפני החדרתו למערכת, תכשיל את זיהויו המדויק ותאפשר, במקרים מסויימים, חדירה ללא הפרעה.

הבעייתיות הקיימת בשיטות הנקוטות כיום בזיהוי מדויק של אופי הקובץ דוחפת למחקרים רבים למציאת שיטות חדשות מדויקות יותר למציאת טיפוסי קבצים שונים, לצערנו לעיתים על חשבון ביצועים.

היעדים העיקריות אליהם יחתור מחקר בנושא:

- דיוק אבחון, מינימום טעויות מטיפוס I וטיפוס II.
- אבחון אוטומטי, ללא "מגע יד אדם".
- זמן אבחון מהיר.
- קושי במניפולציה של תוכן הקובץ, במטרה להערים על השיטה.
- מנגנון אוטומטי ליצירת חתימות טיפוס.

בהמשך, יעסוק המאמר בשיטה הנותנת מענה הולם לחלק מהדרישות לעיל ומניחה בסיס למחקרים עתידיים במטרה לתת מענה לאותן דרישות שלא יבואו על סיפוקן במאמר זה.

4.3 המחקר

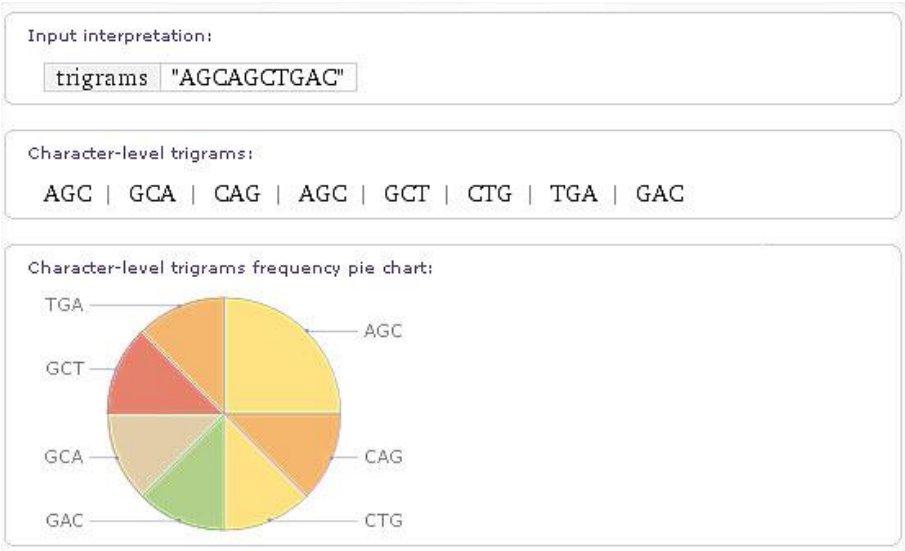
המחקר בא לתת מענה לשתי שאלות מהותיות שעלו ממחקרים קודמים ובסעיפים קודמים של תזה זו:

- האם ניתן למצוא קשר מובהק בין סוג הקובץ למבנהו האנטומי?
- אם התשובה לשאלה הקודמת חיובית, האם ניתן לאפיין שיטה אוטומטית לצורך יצירת חתימה דיגיטלית מאפיינת של טיפוס ללא התערבות אנושית?

בעבודות קודמות שראינו, הבחנו בקשר הקיים בין התפלגות הבתים לסוג הקובץ. מטרת המחקר זה היא להרחיב את היריעה ולהציע מסגרת (Framewok) כבסיס לביצוע ניסיונות שונים, תוך שמירה על גמישות גבוהה ככל האפשר של האפשרות להגדרות שונות בניסוי.

המכנה המשותף בכל הניסויים הוא בניסיון למצוא מאפיינים יחודיים לכל טיפוס קוד בעזרת שימוש ב N-Grams בגדלים שונים ובגודל חלונות שונה, תוך שימוש באנאליזות סטטיסטיות שונות במטרה למצוא את האנליזה שתניב את התוצאה הטובה ביותר.

"N-Gram" הוא למעשה רצף של N בתים בקוד הקובץ, למרות שהארכיטקטורה המוצעת מאפשרת גמישות בהגדרת הרצף על ידי הגדרת חלון גדול יותר מגודל ה N-Gram, ועל כך בהמשך. מציאת כל ה N-Grams בקובץ היא למעשה הזחה של חלון, שגודלו N, בית אחר בית בקוד הקובץ וקריאת הרצף הנמצא במסגרת החלון. בתרשים הבא, שיוצר בעזרת WolframAlfa, מוצגת המחרוזת "AGCAGCTGAC" והתוצאה של איתור כל ה N-Grams בגודל 3, כולל נתוני ההתפלגות.



תרשים 19: רשימת N-Grams בגודל 3 עבור מחרוזת

לצורך יצירת ה N-Grams מתכולת הקובץ עושה המחקר שימוש בחבילת קוד משותף שנועד במקור לניתוח קבצי טקסט במטרה למצוא קשרים בין מופעי מילים שונות.

4.3.1 חבילת NSP - Ngram Statistics Package [29]

חבילת NSP היא אסופה של תוכניות שונות הכתובות בקוד Perl בעלות מטרה משותפת, אנאליזה של קבצי טקסט. החבילה חינמית תחת רשיון GNU וניתנת להורדה חופשית מאתר http://ngram.sourceforge.net או אתר http://search.cpan.org/dist/Text-NSP.

בחבילה זו מוגדר ה N-Gram כרצף של N אסימונים, המופיעים בחלון שגודלו לפחות בגודל של N. הגדרת אסימון נתונה בידי המשתמש בחבילה אולם כברירת המחדל מתייחסת החבילה על מילה כאסימון ועל סימני הפרדה (White Spaces) כמפריד בין אסימונים.

החבילה מספקת את הפונקציונאליות הבאה:

- תוכנית count.pl שתפקידה לספור את כל מופעי ה – N-Gram בגודל מוגדר מראש וכן לציין את מספר המופעים של כל אחד מחלקי ה N-Gram. פלט התכנית הוא רשימת ה N-Grams בסדר יורד של מספר המופעים. התכנית מציעה גמישות רבה בבחירת גודל ה N-Gram, גודל החלון, הגדרת דפוסים מהם התכנית יכולה להתעלם, הגדרת תנאי עצירה בטקסט הנבדק והגדרת סף להתעלמות מ N-Grams המופיעים בתדירות נמוכה.
- תכנית statistic.pl מקבלת כקלט את אסופת קבצי הפלט של count.pl ומבצעת אנאליזה סטטיסטית, לפי בחירת המשתמש, על קבצים אלו. התכנית מעניקה ציון בהתאם לתוצאה של האנאליזה והפלט הוא רשימת ה N-Grams בסדר עולה של הציון. התכנית מקבלת כפרמטר את האנאליזה הסטטיסטית לביצוע ומאפשרת גמישות בקביעת הפורמט של הפלט. החבילה מספקת כ 20 סוגים שונים של אנאליזות סטטיסטיות מוכנות מראש, אולם מאפשרת למשתמש לייצר אנאליזות נוספות כאוות נפשו ולהוסיפם לחבילה.

הסיבה לבחירתי בחבילה זו היא הפונקציונאליות הבסיסית אותה היא מספקת למציאת כל מופעי ה N-Grams בגודל מוגדר ובחלון מוגדר והאנאליזות הסטטיסטיות שניתן לבצע עליהם. מבחינה מעשית הבחירה בחבילה זו אינה מוצלחת בגלל רמת הביצועים שלה, שאינה מספקת. מימוש מעשי של הפונקציונאליות, אותה החבילה מספקת, ידרוש כתיבה מחדש של הקוד.

4.3.2 שלבי המחקר הראשונים

הניסיון בשלבים הראשונים לאתר מאפיינים לטיפוסי קוד שונים בקבצים, הוביל בסופו של דבר למבנה הארכיטקטורה המוצעת. המחקר התחיל במספר נסיונות, בתחילה תוך התמקדות בטיפוס מסמך Word 2007 של מיקרוסופט – docx ולאחר מכן תוך הוספת קבצים בטיפוסים נוספים כ 'rtf, exe, dll, pdf ו zip. המטרה הראשונית היתה לבדוק אם ניתן למצוא דפוס N-Gram החוזר על עצמו עבור קבצים שונים, תוך בחינת החזרה בהסתמך על מבחנים סטטיסטיים שונים כגון Fisher Exact Test שמטרתו לבדוק עד כמה המצאות N-Gram מסוים בקוד היא מקרית מבחינה סטטיסטית ומבחן Chi-Square הבודק את טיב ההתאמה בין נתונים תאורטיים לבין אילו שהתקבלו על ידי התבוננות. בדיקות אלו מסופקות כחלק מחבילת NSP המוזכרת בסעיף הקודם עבור N-Gram בגודל 2 בלבד.

היות והחבילה מטפלת בקבצי טקסט, הוספתי תכנית שרות שמטרתה לקרוא קובץ בינארי ולכתוב את תכולתו כקובץ טקסט המכיל את הייצוג ההקסדצימלי של הקוד.

הבדיקות הראשונות לא העלו תוצאות מובהקות בולטות ולכן בחרתי בכוון אחר בו בחנתי את אחוז המופעים של N-Gram מסויים מתוך סך כל ה N-Grams תוך בדיקה אם אחוז זה עקבי עבור קבצים שונים. היות ומבחן זה לא דרש אנאליזה סטטיסטית שסופקה על ידי חבילת NSP, נפתחה בפני האפשרות להשתמש ב N-Grams בגדלים שונים מעבר ל 2. מרבית האנאליזות המסופקות עם חבילת ה NSP מוגבלות ל N-Grams בגודל 2 למעט מספר מבחנים מועט המאפשר N-Grams בגודל 3.

לצורך המבחן האחרון הוספתי תכנית שרות נוספת שמטרתה לקרוא קובץ המכיל את נתוני הספירה של ה N-Grams ולהוסיף עמודה נוספת, המכילה עבור כל N-Gram, את אחוז המופעים שלו מסך כל מופעי N-Grams בקובץ. לצערי, גם מבחן זה לא הניב את התוצאות המקוות היות ואחוז ניכר של ה N-Grams הופיע אך ורק פעם אחת בקובץ, עובדה שהובילה לתוצאות אחוז מופעים לא עקביות.

בשלב זה בחרתי לבדוק את מופעי ה N-Gram מכוון שונה. אם בתחילה התמקד המחקר בבדיקת הקשר בין כמות מופעי ה - N-Gram לבין הקובץ בו הם נמצאו, בחרתי כעת לבדוק את מופע ה N-Gram ביחס למופעו בקבצים אחרים באותו טיפוס בצורה עיקבית לעומת חוסר המופע שלו בקבצים המזוהים עם טיפוסים אחרים, דהיינו, למצוא בידול ל N-Grams מסויימים. לצורך הבדיקה הוספתי 2 תכניות שרות. האחת, שמטרתה היתה לאגד את נתוני הספירה מכל הקבצים של טיפוס מסויים, בקובץ אחד והשנייה, שמטרתה לאתר N-Grams המופיעים בצורה עקבית בטיפוס מסויים (100% הופעה) אולם אינם מופיעים בצורה עיקבית עבור כל שאר הטיפוסים. בדיקה זו החלה להניב פירות בשלב האיגוד, עת הבליטה את העובדה שאכן, עבור N-Grams מסויימים ניתן להבחין בכמות מופעים מובהקת עבור טיפוסים מסויימים. השלב השני, שלב היחוד, לא הניב תוצאות מוצלחות במיוחד כצפוי (בדיעבד) לאור העובדה שכל הוספת טיפוס נוספת מקטינה את כמות ה N-Grams הייחודיים, לעיתים עד לרמה של N-Gram אחד בלבד, עובדה שפגעה במעשיות כוון זה לאור הקלות בה ניתן לזייף טיפוס, על ידי הוספת N-Gram אחד לתכולת הקובץ.

כפי שציינתי, קיבלתי תוצאות מעניינות בבדיקה האחרונה, אולם זו בוצעה עבור N-Grams בגודל 2 והתוצאות היו טובות במיוחד עבור קבצים קטנים. ככל שנפח הקבצים גדל, צף ועלה חיסרון הבדיקה עבור גודל N-Gram זה. לאור העובדה שעבור N-Gram בגודל 2 קיימים לכל היותר $2^{16} = 65,536$ N-Grams שונים, ככל ההקובץ גדול יותר, כך מובטח מופע של כל N-Gram פעם אחת לפחות בתכולת הקובץ, עובדה שפגעה במעשיות הבדיקה זו.

בשלב זה בחנתי N-Grams גדולים יותר, עובדה שהובילה לבעיות ביצועי CPU ובמיוחד לבעיות ביצועי זיכרון, וזאת לאור כמות אפשרויות ה N-Grams השונות, למשל $2^{24} = 16,777,216$ עבור N-Grams בגודל 3. דווקא בעיית הזיכרון והניסיון לפתור אותה הובילו אותי לשיטה המוצעת בהמשך. בנסיונות הראשוניים התמקדתי במציאת N-Grams בגודל 3 המופיעים ב 100% מהקבצים של טיפוס. לצורך כך אגרתי בזיכרון כל N-Gram שהופיע. ככל שהתרבו הקבצים בבדיקה ונפחיהם גדלו, כך גדלה כמות הזיכרון שנדרשה על ידי תוכנית השרות עד לקריסת התוכנית.

כדי לפתור את הבעייה, סיננתי מראש N-grams שחסרו מקובץ זה או אחר. שינוי זה אכן פתר את בעיית הזיכרון אולם הציף בעייה אחרת. עבור טיפוסים מסויימים, כמות ה N-Grams הכרחיים היתה נמוכה ביותר, לעיתים 1 בלבד. אני מניח שבמקרים אלו בדרך כלל מדובר היה בחלקים מה-Magic Numbers של הטיפוס. בעייה זו הובילה למעשה לפתרון, כפי שהוא מוצג, לאחר שהוספתי סף כניסה לאחוז מופעי N-Gram בטיפוס.

4.4 ארכיטקטורת מערכת המודלים הסטטיסטיים

הארכיטקטורה המוצעת מורכבת משני מסלולים מקבילים, מסלול בניית החתימה ומסלול הבדיקה. כל מסלול מורכב ממספר שלבים, חלקם משותפים. המסלול המתואר כולל צעדים שמטרתם להתאים את התהליך לתכניות השרות ששימשו את הניסיון, לדוגמה, תרגום קובץ בינארי לקובץ טקסט. כמובן שבמימוש מעשי של השיטה המתוארת, מספר תהליכים יאוחדו או יבוטלו. לדוגמה, במימוש מעשי של מסלול הבדיקה, כל הצעדים יאוחדו לתהליך כולל אחד.

תרשים הארכיטקטורה מתואר בתרשים 21. חלקו הימני של התרשים מתאר את מסלול הכנת החתימות וחלקו הימני את מסלול הבדיקה של הקבצים.

4.4.1 מסלול הכנת חתימה

מסלול הכנת החתימות מתחיל בספריה הכוללת מדגם מייצג של קבצים עבור טיפוס מסויים. קבצים אלו מהווים את חומר הגלם ולכן חשוב להקפיד שאכן הם מהטיפוס הנדון.

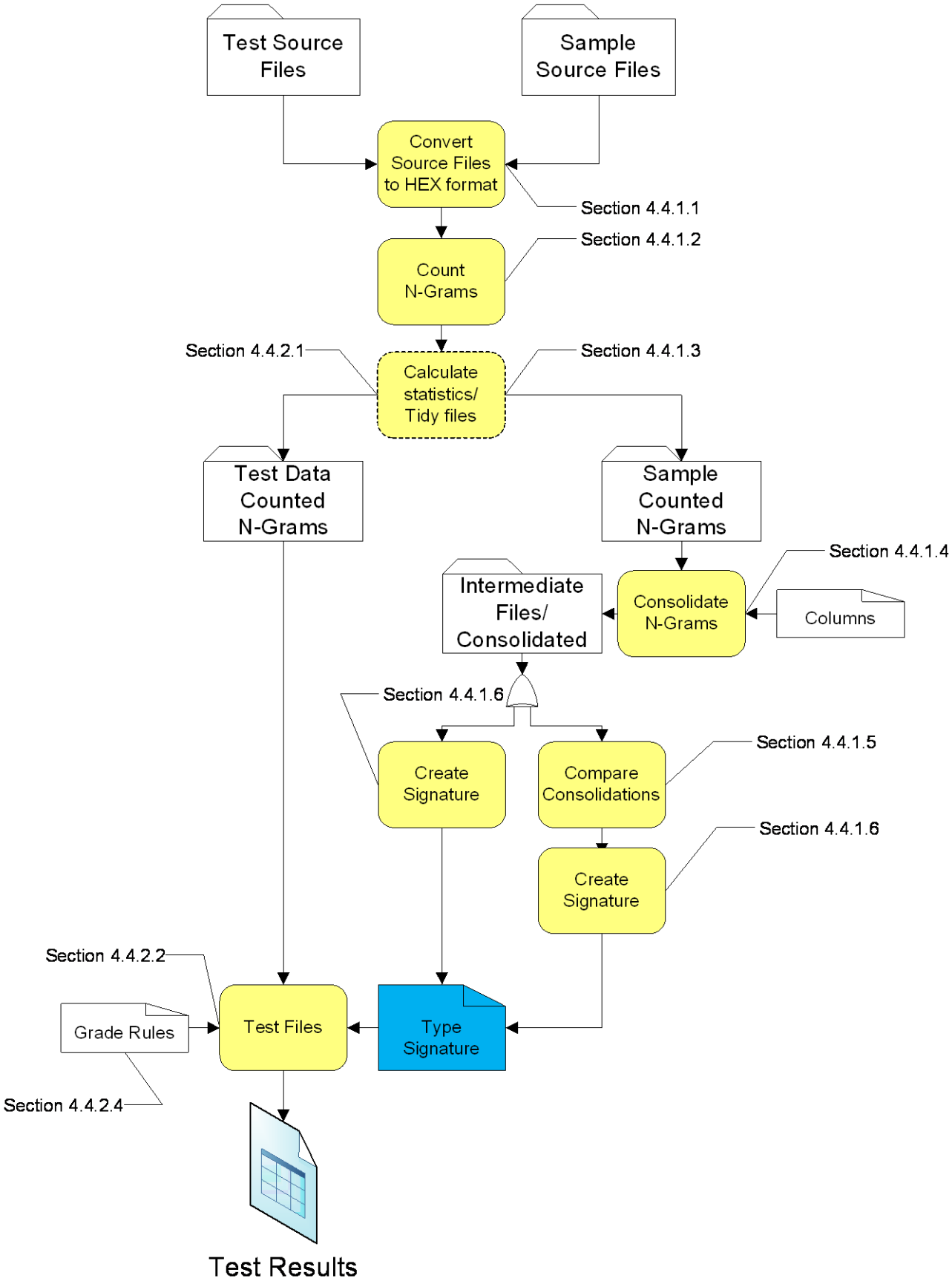
4.4.1.1 המרה לקובץ טקסט

שלב זה נועד להמיר את הקבצים הנבדקים לפורמט טקסטואלי אחיד, ללא קשר לפורמט המקורי שלהם, טקסט או בינארי. תוצאת ההמרה היא קובץ טקסט המכיל את תוכן קובץ המקור במבנה מספרים הקסדצימליים המופרדים בינהם בעזרת רווחים.

```
ff d8 ff e0 00 10 4a 46 49 46 00 01 01 01 00 9b 00
01 00 00 10 01 02 00 0a
00 00 00 1c 01 00 00 12 01 03 00 01 00 00 00 01 00
a4 03 00 01 00 00 00 00 00 30 6c 0c a4 03 00 01 00
03 00 00 4e 49 4b 4f 4e 20 43 4f 52 50 4f 52 41 54
00 01 00 00 00 ea 02 00 00 05 92 05 00 01 00 00 00
92 05 00 01 00 00 00 fa 02 00 00 90 92 02 00 03 00
00 00 00 c4 09 00 00 64 00 00 00 0a
00 00 00 32 30 30 34 3a 31 32 3a 30 31 20 30 39 3a
00 00 00 b4 00 00 00 0a
00 00 00 00 02 00 02 02 01 01 00 06 00 03 01 03 00
0b 11 15 0f 0c 0c 0f 15 18 13 13 15 13 13 18 11 0c
0b 01 00 01 05 01 01 01 01 01 01 00 00 00 00 00 00
0b 10 00 01 04 01 03 02 04 02 05 07 06 08 05 03 0c
6f fd 5e c8 cb fd ad 5d 4e b1 c1 96 35 e1 cd 1b 60
5f 64 c9 ff 00 44 ef f3 4f fe 92 4b eb 1f 56 c8 e9
af 16 e7 d5 f6 b8 75 7a 1e e6 3d ad 01 fe e7 b4 b4
e1 e7 e8 d0 ef fa 82 b5 5d 95 b6 75 26 3b 37 71 ff
dd 50 be 9d d7 52 f2 22 6a b1 cd ad ed b1 ae fa 5e
19 97 7d 8e aa 87 93 6e c0 21 fe 95 8c fc df 5e db
32 0d 6e 0e e4 02 0c 1e 09 0a
a6 4e 1d 57 db 75 e6 d7 9b 6c 71 79 06 08 d7 46 d7
```

תרשים 20: יצוג טקסטואלי של קובץ בינארי

מבנה טקסט זה מקל על חבילת NSP לבחון את תוכן הקובץ תוך התייחסות לטקסט המייצג של קוד בית אחד (8 ביטים) כמילה בטקסט. לדוגמה בתרשים 20, ערכו של הבית הראשון בקובץ הוא ff ששוה ערך ל 255 בבסיס 10 או 11111111 בבסיס 2.

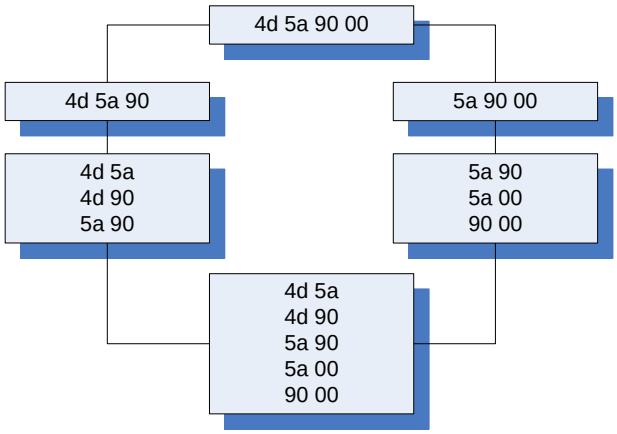


תרשים 21: ארכיטקטורת מערכת מודלים סטטיסטיים לזיהוי טיפוס בעזרת N-Grams

4.4.1.2 ספירת N-Grams

תהליך ספירת ה N-Grams מבוצע על ידי תכנית השרות Count.pl מחבילת NSP. התכנית מקבלת כקלט רשימת קבצי טקסט ASCII ומייצרת את רשימת ה - N-Grams המופיעים בקבצים אלו, בנוסף לתדירויות שלהם. הפלט הוא רשימת ה – N-Grams בסדר יורד של השכיחות שלהם. התכנית מאפשרת להגדיר את הפרמטרים הבאים:

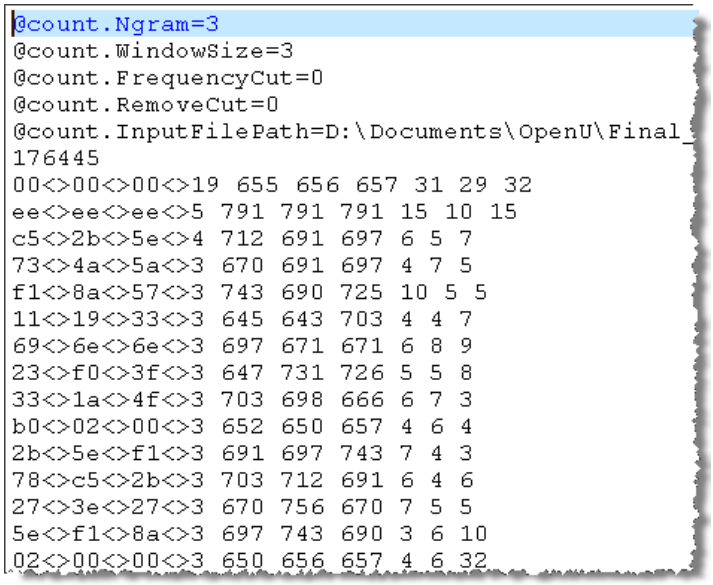
- גודל ה N-Gram. ברירת המחדל היא 2.
- גודל החלון. ברירת המחדל עבור גודל החלון הוא הערך שניתן ל N-Gram. פרמטר זה מאפשר הגדרה של חלון הגדול מגודל ה – N-Gram (אין משמעות להגדרת חלון קטן יותר). במקרה של חלון הגדול מגודל ה N-Gram, תחזיר התכנית את רשימת ה N-Grams, הן הרציפים והן הלא רציפים, הנמצאים במסגרת החלון. לדוגמה עבור הטקסט "4d 5a 90 00", התרשים הבא מתאר את תהליך מציאת כל ה - N-Grams בגודל 2 עבור חלון בגודל 3. חלקו השמאלי של התרשים מתאר את החלון הראשון המכיל את הטקסט "4d 5a 90". בתוך טקסט זה נמצאו ה -N-Grams "4d 5a", "4d 90", "5a 90". תהליך דומה עובר עבור החלון השני המתואר מצידו הימני של התרשים. יש לשים לב ל N-Gram "5a 90" המופיע בשני החלונות ולכן יופיע פעם אחת בלבד ברשימה המאוחדת של כל ה N-Grams.



תרשים 22: מציאת N-Grams בגודל 2 עבור חלון בגודל 3

- סף כניסה מינימלי של תדירות N-Gram. ניתן לקבוע סף תדירות כך ש - N-Grams המופיעים בתדירות נמוכה ממנו לא יוצגו או, לא יוצגו וגם לא יספרו, לפי בחירה.

התרשים הבא מתאר פלט דוגמה עבור N-Gram בגודל 3. בשורות הראשונות מוצגים הפרמטרים שנבחרו עבור פלט זה, לדוגמה גודל ה – N-Gram, גודל החלון, סף הכניסה ושם קובץ הקלט. הערך 176445 בשורת הפרמטר האחרונה מסמן שהיו 176445 N-Grams שונים בקובץ הקלט.



תרשים 23: תוצאת ספירת מופעי N-Grams בגודל 3 עבור חלון בגודל 3

השורות לאחר מכן מפרטות את ה – N-Grams שנמצאו ואת התדירויות שלהם. בראשית כל שורה נמצא את ה - N-Gram הנדון. הסימן <> מפריד בין חלקי ה N-Gram השונים ולכן בשורה הראשונה שבדוגמה נמצא את ה – N-Gram "00 00 00" מסומן כ – "00<>00<>00<>". המספר הראשון לאחר מכן מציין את מספר הפעמים ש N-Gram זה הופיע בקובץ הקלט, 19 פעמים במקרה זה. רשימת המספרים שלאחר מכן מציינת את מספר המופעים של כל חלק של N-Gram בקובץ הקלט. לדוגמה עבור השורה הרביעית בתרשים, "73<>4a<>5a<>3 670 691 697 4 7 5", הטבלה הבאה מפרטת את מספר המופעים עבור כל חלק של ה N-Gram. הסימון *xx*<> <> מייצג קוד כלשהוא.

חלק N-Gram	מספר מופעים
73<>4a<>5a<>	3
73<>*xx*<>*xx*<>	670
xx<>**4a**<>*xx*<>	691
xx<>*xx*<>**5a**<>	697
73<>4a<>*xx*<>	4
73<>*xx*<>**5a**<>	7
xx<>**4a<>5a**<>	5

טבלה 1: רשימת תדירויות עבור חלקי N-Gram בגודל 3

נתונים אלו ניתנים באופן אוטומטי על ידי חבילת ה – NSP ואינם משמשים בהדגמה למחקר זה, אולם הם עשויים בהמשך לשמש לצורך הוספת בדיקות נוספות למשל למציאת קשר בין מספר המופעים של N-Gram לבין מספר המופעים של התחילית שלו.

4.4.1.3 אנליזה סטאטיסטית / סידור קבצים

קבצי הפלט של תהליך הספירה מרוכזים בספרייה משותפת שמכילה את כל קבצי הפלט עבור טיפוס מסוים. קבצים אלו משמשים כקלט לתהליך האנליזה הסטאטיסטית. התוצאה נרשמת בקובץ פלט נפרד עבור כל קובץ קלט. בתהליך זה מבצעים אנליזה של נתוני הספירה, ומחשבים נתונים סטטיסטיים כנדרש.

חבילת ה – NSP מגיעה עם מספר יישומים סטטיסטיים עבור B-Grams (N-Grams בגודל 2) ומספר קטן יותר של ישומים עבור N-Grams בגודל 3. התכנית מקבלת את שם הבדיקה הסטטיסטית כפרמטר ומחשבת את הנתון הסטטיסטי עבור כל N-Gram בנפרד. פלט התכנית הוא רשימה דומה לזו שבסעיף הקודם הכוללת 2 עמודות נוספות. עמודה אחת הכוללת את הערך הסטטיסטי שחושב עבור ה – N-Gram והשנייה כוללת את דירוג ה- N-Gram ביחס לאחרים, בהתבסס על הציון שקיבל. רשימת ה - N-Grams בפלט התכנית ממויינת לפי הדירוג בסדר עולה. במקרה ושני N-Grams יקבלו ציון זהה, הם יקבלו את אותו הדירוג ולכן מצב בו מספר N-Grams יקבלו את אותו הדירוג אפשרי. אחד הפרמטרים לריצת התוכנית הוא רמת הדיוק הנדרשת בחישוב הסטטיסטי ולכן בעייה בה נתונים רבים מקבלים תוצאת חישוב זהה כתוצאה מעיגול נתונים וכתוצאה מכך דירוג זהה, ניתן לפתור על ידי דרישת דיוק גבוהה יותר.

```
@count.Ngram=2
@count.WindowSize=2
@count.FrequencyCut=0
@count.RemoveCut=0
@count.InputFilePath=jpeg_files.hex\Glass.hex
@statistic.StatisticName=Dice Coefficient
@statistic.Formatted=1
Total sample size = 3181

 N-gram  Rank  Dice Coefficient value  Frequency Values
-------------------------------------------------------
37<>37      1           0.7000              49 70 70
00<>00      2           0.3976              33 83 83
ff<>c4      3           0.2963              4 16 11
6e<>5f      4           0.2667              2 10 5
5f<>74      4           0.2667              2 5 10
ae<>b3      5           0.2353              2 7 10
07<>08      6           0.2222              3 14 13
ba<>73      7           0.2105              2 8 11
03<>01      8           0.2000              6 29 31
74<>6e      8           0.2000              2 10 10
dc<>68      8           0.2000              1 7 3
```

תרשים 24: תוצאת אנליזה סטטיסטית עבור קובץ ספירת B-Gram

לדוגמה בתרשים פלט הדוגמה, בדומה לקובץ הפלט של תהליך הספירה, בשורות הראשונות נמצא את הפרמטרים שנבחרו לשם הבדיקה הסטטיסטית, Dice Coefficient במקרה זה, גודל ה N-Gram וגודל החלון וסף הכניסה המינימלי של תדירות ה - N-Gram. בנתוני ה – N-Grams נמצא את ה – N-Gram בעמודה השמאלית. העמודה שלאחריה מציינת את דירוג ה – N-Gram לפי החישוב הסטטיסטי שבוצע. העמודה השלישית מציינת את הציון שקיבל ה – N-Gram בחישוב ולאחריה נתוני התדירות של חלקי ה N-Gram השונים, כפי שהופיעו בקובץ הספירה.

לחילופין, במקרה שלא נדרשת אנליזה סטטיסטית, כמו בדוגמה שבסעיף הוכחת נכונות הרעיון בהמשך, ניתן להשתמש בתכנית השירות countTidy אשר קוראת את קובץ הספירה של ה - N-Grams ומסדרת אותו בפורמט ראוי לקריאה. בנוסף לכך מוסיפה התכנית עמודה נוספת, מימין לעמודת ה N-Gram, ובה מצויין אחוז המופעים של ה N-Gram המסויים בשורה מסך כל ה N-Grams בקובץ הקלט. דוגמה לפלט תהליך זה נראית בתרשים הבא.

```
@count.Ngram=3
@count.WindowSize=3
@count.FrequencyCut=0
@count.RemoveCut=0
@count.InputFilePath=D:\Documents\OpenU\Final_paper_test_data\
Total sample size = 17524
N-gram          Percentage      Frequency Values
----------      ----------      ----------------------------
0a<>0a<>0a      0.29673591      52      106  106  106  55  52  55
01<>01<>01      0.08559690      15      73  73  73  20  19  20
00<>00<>00      0.05706460      10      181  181  181  16  17  16
02<>02<>02      0.05135814      9     64  64  64  12  10  12
f0<>ff<>00      0.02853230      5     129  116  181  5  5  105
00<>00<>01      0.02853230      5     181  181  73  16  13  9
01<>02<>03      0.02282584      4     73  64  54  9  6  4
ff<>c4<>00      0.02282584      4     116  83  181  4  10  4
```

תרשים 25: תוצאת הרצת תכנית Tidy על קובץ ספירה

4.4.1.4 איחוד תוצאות

הקלט לשלב זה הוא קבצי הנתונים הסטטיסטיים שהתקבלו כתוצאה מהשלב הקודם. שלב זה מאחד את התוצאות שהתקבלו עבור כל הקבצים השונים, השייכים לטיפוס מסויים כלשהו, לקובץ אחד.

קובץ זה מכיל את סיכום כל הנתונים שהתקבלו עבור כל N-Gram בקבצים השונים. בנוסף, שלב זה מאפשר לבצע, על כל עמודה בנתוני האיחוד, מספר חישובים סטטיסטיים בסיסיים כמציאת סכום, ממוצע, המינימום, המקסימום, סטיית תקן ואחרים. השלב מבוצע על ידי תוכנית, המקבלת כקלט מספר פרמטרים:

- ספריית הקלט - בספרייה זו תצפה התכנית למצוא את כל קבצי הפלט שנוצרו מתהליך האנליזה הסטטיסטית. לצורך קבלת תוצאות מייצגות ואמינות, כל הקבצים בספריה זו חייבים להיות מאותו טיפוס.
- שם הספרייה ושם הקובץ בו ירשמו תוצאות הריצה.
- אחוז סף הכניסה עבור מינימום מופעי N-Gram בקבצים שונים - בעזרת פרמטר זה ניתן להגדיר אילו N-Grams יכללו בתוצאות. ברירת המחדל היא 100%, דהיינו, בתוצאת הריצה יכללו אך ורק N-Grams שנמצאו בכל קבצי הקלט. N-Gram שחסר מקובץ זה או אחר לא יכלל בתוצאות הסופיות ונתוניו הסטטיסטיים לא יחושבו. קביעת הערך ל 70% למשל, תוודא שבתוצאה הסופית יכללו N-Grams שנמצאו לפחות ב 70% מקבצי הקלט. בהמשך נראה כי לפרמטר זה חשיבות רבה במקרה של טיפוסים בעלי מאפיינים יותר חלשים.
- שם הטיפוס - שם זה יוצמד לחתימה שתתבסס על הנתונים שיופקו על ידי ריצת תכנית זו.
- קובץ העמודות – קובץ זה מכיל את הגדרת העמודות שיופקו בקובץ הפלט של ריצת התכנית.

קובץ העמודות כתוב במבנה XML ומתאר את תוכן עמודות הפלט. דוגמה לקובץ זה ניתן לראות בתרשים הבא. הגדרת העמודות מתיחסת לעמודות קובץ הקלט, לפי סדר ההופעתן, תוך התעלמות מעמודת ה – N-Gram שתמיד מופיעה ראשונה. מסיבה זו הגדרת העמודה הראשונה בקובץ העמודות מתייחסת לעמודה השנייה בקובץ הקלט.

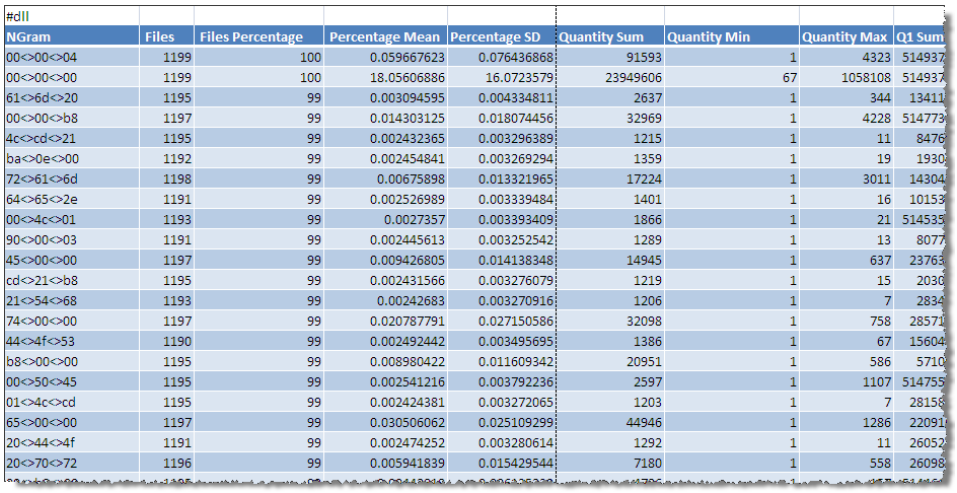
```xml
<?xml version="1.0" encoding="UTF-8"?>
<Columns>
    <Column>
        <Title>Percentage</Title>
        <Stats>
            <Stat type="mean">Mean</Stat>
            <Stat type="standard_deviation">SD</Stat>
        </Stats>
    </Column>
    <Column>
        <Title>Quantity</Title>
        <Stats>
            <Stat type="sum">Sum</Stat>
        </Stats>
        <Stats>
            <Stat type="min">Min</Stat>
        </Stats>
        <Stats>
            <Stat type="max">Max</Stat>
        </Stats>
    </Column>
    <Column>
        <Title>Q1</Title>
        <Stats>
            <Stat type="sum">Sum</Stat>
        </Stats>
    </Column>
    <Column>
```

תרשים 26: קובץ הגדרת עמודות לתהליך איחוד נתונים

התווית <Columns> מייצגת את רשימת העמודות וכוללת בתוכה הגדרת עמודה אחת או יותר. אילו מסומנות בעזרת התווית <Column>. כל הגדרת עמודה מכילה את הגדרת כותרת העמודה, המסומנת על ידי התווית <Title>, והגדרת החישובים הסטטיסטיים הנדרשים על עמודה זו. חישובים אלו מאוגדים על ידי התווית <Stats> שמכילה חישוב סטטיסטי אחד או יותר. הגדרת חישוב סטטיסטי בודדת מוגדרת על ידי התווית <Stat>. המאפיין "type" מייצג את סוג החישוב, לדוגמה "standard_deviation" עבור סטיית תקן, ותוכן התווית מציין את הכותרת שתוצמד לעמודת חישוב זו. משתמע מכך שלכל עמודה בקבצי הקלט, יהיו מספר עמודות בקובץ הפלט כמספר החישובים הסטטיסטיים הנדרשים על עמודה זו.

ניקח לדוגמה את פלט הרצת תכנית ה – Tidy אשר בתרשים 25. העמודה הראשונה לאחר עמודת ה – N-Gram מציינת את אחוז המופעים של N-Gram זה מסך כל ה – N-Grams השונים אשר בקובץ הקלט. הגדרת העמודה הראשונה בתרשים 26 אכן מתיחסת לעמודה זו וכותרתה "Percentage". על העמודה נדרשים חישובי הממוצע, "mean" וסטיית התקן, "standard_deviation". לכן בקובץ הפלט נקבל שתי עמודות תוצאה עבור עמודת ה – "Percentage" מקובץ הקלט, הראשונה "Percentage Mean" והשנייה "Percentage SD", בהתאם לכותרות המוגדרות בקובץ העמודות עבור כל חישוב נדרש.

תוצאת הריצה נרשמת בקובץ בעל מבנה CSV (Comma Separated Values) לנוחות הקריאה בעזרת Excel או כל תוכנה אחרת הקוראת פורמט זה. דוגמה לפלט קובץ מסוג זה ניתן לראות בתרשים הבא.



#dll

NGram	Files	Files Percentage	Percentage Mean	Percentage SD	Quantity Sum	Quantity Min	Quantity Max	Q1 Sum
00<>00<>04	1199	100	0.059667623	0.076436868	91593	1	4323	514937
00<>00<>00	1199	100	18.05606886	16.0723579	23949606	67	1058108	514937
61<>6d<>20	1195	99	0.003094595	0.004334811	2637	1	344	13411
00<>00<>b8	1197	99	0.014303125	0.018074456	32969	1	4228	514773
4c<>cd<>21	1195	99	0.002432365	0.003296389	1215	1	11	8476
ba<>0e<>00	1192	99	0.002454841	0.003269294	1359	1	19	1930
72<>61<>6d	1198	99	0.00675898	0.013321965	17224	1	3011	14304
64<>65<>2e	1191	99	0.002526989	0.003339484	1401	1	16	10153
00<>4c<>01	1193	99	0.0027357	0.003393409	1866	1	21	514535
90<>00<>03	1191	99	0.002445613	0.003252542	1289	1	13	8077
45<>00<>00	1197	99	0.009426805	0.014138348	14945	1	637	23763
cd<>21<>b8	1195	99	0.002431566	0.003276079	1219	1	15	2030
21<>54<>68	1193	99	0.00242683	0.003270916	1206	1	7	2834
74<>00<>00	1197	99	0.020787791	0.027150586	32098	1	758	28571
44<>4f<>53	1190	99	0.002492442	0.003495695	1386	1	67	15604
b8<>00<>00	1195	99	0.008980422	0.011609342	20951	1	586	5710
00<>50<>45	1195	99	0.002541216	0.003792236	2597	1	1107	514755
01<>4c<>cd	1195	99	0.002424381	0.003272065	1203	1	7	28158
65<>00<>00	1197	99	0.030506062	0.025109299	44946	1	1286	22091
20<>44<>4f	1191	99	0.002474252	0.003280614	1292	1	11	26052
20<>70<>72	1196	99	0.005941839	0.015429544	7180	1	558	26098

תרשים 27: דוגמת פלט של תהליך איחוד נתונים

השורה הראשונה מציינת את שם הטיפוס שהוכנס כפרמטר לריצת התכנית, dll במקרה זה. השורה שלאחריה מציינת את כותרות העמודות ולאחריה נתוני ה – N-Grams המחושבים. עמודה ראשונה מציינת את ה N-Gram הנדון. שורה שניה מציינת את כמות הקבצים מתוך המדגם בהם נמצא N-Gram זה. עמודה שלישית מציינת את אחוז הקבצים מתוך סך כל קבצי המדגם בהם נמצא ה - N-Gram. מהעמודה הרביעית ואילך נמצא את העמודות המחושבות לפי הגדרתם בקובץ העמודות. בהתייחס לדוגמה בסעיף הקודם ותרשימים 25 ו – 26, אכן העמודה הרביעית מציגה את ממוצע אחוז הופעות ה N-Gram בקבצים השונים והעמודה הבאה אחריה מציגה את סטיית התקן עבור נתון זה. שאר העמודות בהמשך מציינות אף הן את הנתונים בהתאם להגדרתם.

4.4.1.5 השוואת תוצאות איחוד

שלב אופציונלי זה מאפשר השוואת תוצאות איחוד של טיפוסים שונים במטרה למצוא N-Grams ייחודיים לטיפוס זה או אחר. בהמשך נראה כי ניתן להשתמש בנתון זה לצורך יצירת חתימה המבוססת על N-Grams ייחודיים עד רמה מסויימת, כפי שנקבעת על ידי המשתמש.

תכנית השרות להשוואת התוצאות מקבלת את הנתונים הבאים:

- נתוני ספריית הקלט – בספרייה זו מצפה התכנית לקבל את קבצי איחוד תוצאות של כל הטיפוסים אותם נרצה להשוות.
- נתוני ספריית ושם קובץ הפלט.
- סף מעבר ל N-Gram – אך ורק N-Grams שיהיו באחוז קבצים הגדול מערך זה יכללו בהשוואה. ברירת המחדל היא 100%.

- עמודות לאיסוף נתונים – מספרי עמודות הנתונים, אשר את נתוניהם נרצה ליבא מקבצי הקלט. הספירה מתחילה מעמודת הנתונים הראשונה, שמסומנת במספר 1 וממוקמת במקום הרביעי משמאל לאחר עמודות ה N-Gram, כמות הקבצים ואחוז מופע בקבצים. עמודות אלו משמשות בהמשך לצורך יצירת החתימה.

הפלט של התכנית הוא בפורמט CSV. דוגמה לפלט זה ניתן לראות בתרשים הבא:

NGram	X of Y	pdf_cons_79.csv	dll_cons_99.csv	exe_cons_94.csv	rtf_cons_84.csv	docx_cons_100.csv
1c<>00<>08	1 of 5	<no data>	<no data>	<no data>	<no data>	500^100
6c<>2e<>72	1 of 5	<no data>	<no data>	<no data>	<no data>	500^100
65<>6d<>65	1 of 5	<no data>	<no data>	<no data>	<no data>	500^100
73<>2e<>78	1 of 5	<no data>	<no data>	<no data>	<no data>	500^100
02<>28<>a0	1 of 5	<no data>	<no data>	<no data>	<no data>	500^100
73<>65<>74	1 of 5	<no data>	<no data>	<no data>	<no data>	500^100
13<>00<>08	1 of 5	<no data>	<no data>	<no data>	<no data>	500^100
00<>14<>00	1 of 5	<no data>	<no data>	<no data>	<no data>	500^100
00<>00<>11	1 of 5	<no data>	<no data>	<no data>	<no data>	500^100
00<>06<>00	1 of 5	<no data>	<no data>	<no data>	<no data>	500^100
70<>72<>6f	1 of 5	<no data>	500^100	<no data>	<no data>	<no data>
61<>72<>65	1 of 5	400^100	<no data>	<no data>	<no data>	<no data>
6f<>63<>50	1 of 5	<no data>	<no data>	<no data>	<no data>	500^100
66<>31<>5c	1 of 5	<no data>	<no data>	<no data>	500^100	<no data>
50<>4b<>01	1 of 5	<no data>	<no data>	<no data>	<no data>	500^100
6e<>73<>69	1 of 5	<no data>	<no data>	<no data>	500^100	<no data>
70<>61<>72	1 of 5	<no data>	<no data>	<no data>	500^100	<no data>
69<>6f<>6e	1 of 5	400^100	<no data>	<no data>	<no data>	<no data>
08<>00<>00	1 of 5	<no data>	<no data>	<no data>	<no data>	500^100
a2<>04<>02	1 of 5	<no data>	<no data>	<no data>	<no data>	500^100
61<>74<>65	1 of 5	400^100	<no data>	<no data>	<no data>	<no data>

תרשים 28: קובץ השוואה בין קבצי איחוד לטיפוסים שונים

העמודה השמאלית מציינת את ה – N-Gram. העמודה שלאחריה מציינת בכמה טיפוסים נמצא N-Gram זה מתוך סך כל הטיפוסים. לאחר מכן נמצא עמודה עבור כל טיפוס בספריית הקלט. תוכן התא מציין <no data> אם ה N-Gram לא נמצא באיחוד התוצאות של טיפוס זה. במקרה וכן נמצא, יופיעו הנתונים בתא. נתונים אלו מורכבים מרשימת ערכים המופרדים ביניהם בסימן '^'. הערך הראשון, שתמיד יופיע, מציין את מספר הקבצים בטיפוס המסויים בהם נמצא ה N-Gram. שאר הנתונים מבוססים על עמודות הנתונים שנלקחו מקבצי האיחוד לפי הגדרת הפרמטר לריצת התכנית.

4.4.1.6 ייצור חתימה

שלב זה הינו האחרון במסלול זה ובסיומו נקבל קובץ המכיל את חתימה של טיפוס אחד או יותר. חתימת טיפוס היא למעשה אוסף ה – N-Grams שנמצאו במדגם הקבצים מאותו הסוג של הטיפוס המאופיין, אשר עמדו בסף המופעים כפי שנקבע. קיימות שתי אפשרויות ליצור החתימה, האחת מתבססת על קבצי איחוד התוצאות שהתקבלו בסעיף 4.4.1.4 והשנייה מתבססת על קובץ ההשוואה שהתקבל בסעיף 4.4.1.5 קובץ החתימה הסופי יהיה זהה במבנהו בשני המקרים.

ייצור החתימה דורש את הפרמטרים הבאים:

- נתוני ספריית הקלט – בספרייה זו מצפה התכנית לקבל את קבצי איחוד תוצאות של כל הטיפוסים אותם נרצה להוסיף לחתימה.
- נתוני קובץ הפלט.

- במקרה והחתימה מתבססת על קבצי איחוד - עמודות לאיסוף נתונים, בדומה למתואר בסעיף 4.4.1.5.
- במקרה והחתימה מתבססת על קובץ השוואה - סף מעבר למספר טיפוסים. פרמטר זה מאפשר סינון של N-Grams המופיעים ביותר ממספר הטיפוסים הנקוב. הפרמטר מאפשר, לדוגמה, לבחור N-Grams, שנמצאים אך ורק בטיפוס אחד אולם לא באחרים, לחתימה. בעזרת אפשרות זו ניתן ליצור חתימה מדוייקת יותר, אם כמות ה N-Grams הייחודיים לטיפוס תאפשר זאת.

התרשים הבא מציג דוגמה של קובץ חתימה במבנה CSV. החתימות, עבור הטיפוסים השונים, נמצאות ברצף אחת לאחר השנייה.

pdf	9206		
6e<>74<>65	400	100	1^547
6e<>66<>6f	398	99	1^73
65<>69<>67	395	98	1^160
69<>64<>74	395	98	1^157
65<>61<>72	376	94	1^68
74<>69<>6f	400	100	1^150
73<>74<>72	398	99	1^636
74<>73<>20	386	96	1^117
69<>67<>68	395	98	2^160
30<>32<>32	335	83	1^132
dll	10299		
21<>b8<>01	498	99	1^2
00<>00<>b8	499	99	1^355
00<>00<>00	500	100	67^610569
00<>03<>00	499	99	1^6661
00<>00<>04	500	100	1^4323
2e<>0d<>0a	496	99	1^1

תרשים 29: דוגמה לקובץ חתימה

כל טיפוס מתחיל בשורה בה העמודה הראשונה מציינת את שם הטיפוס, כפי שנרשם בשלב איחוד התוצאות, והשנייה מציינת את סך כל ה N-Grams הנמצאים בחתימה של טיפוס זה. הנתון השני משמש בהמשך לצורך חישוב המשקל שניתן לכל N-Gram בעת חישוב הציון הכללי של קובץ נבחן עבור טיפוס זה.

השורות הבאות לאחר הכותרת מפרטות את ה N-Grams השונים של החתימה. העמודה הראשונה לאחר ה – N-Gram מציינת את מספר הקבצים בהם נמצא N-Gram זה מתוך המדגם. אם נציב סף כניסה של 100% בעת איחוד הנתונים הסטטיסטיים, נצפה שהערך בעמודה זו יהיה אחיד עבור כל ה – N-Grams השייכים לחתימת הטיפוס, והוא יהיה שווה למספר הקבצים שהיו במדגם.

העמודה השנייה מציינת את אחוז הקבצים בהם נמצא ה – N-Gram מתוך סך כל הקבצים שבמדגם. גם ערך זה משמש לצורך חישוב המשקל שיינתן ל – N-Gram זה בעת חישוב הציון. העמודה השלישית מכילה את כל הערכים הסטטיסטיים והחישוביים שנאספו בשלב האיחוד. הערכים מופרדים ביניהם בעזרת הסימן '^'.

4.4.2 מסלול בדיקה

מסלול זה משמש לבדיקת וחשיפת הטיפוס עבור קבצים לא מוכרים. יש לשים לב כי השלבים המתוארים משמשים לצורך תיאור הרעיון, וכל מימוש מעשי של התהליך יחייב מציאת דרך יעילה יותר לטיפול בקוד הבינארי ואיחוד כל השלבים תחת תהליך מאגד אחד.

4.4.2.1 הכנת קבצים לבדיקה

הליך הכנת הקבצים לבדיקה הוא למעשה אותו התהליך המתואר בשלושת השלבים הראשונים בתהליך ייצור החתימה. יש להקפיד שאותם הפרמטרים ששימשו את תכניות השירות לצורך הכנת קבצי המדגם לתהליך ייצור החתימה, ייבחרו גם בשלב הכנת הקבצים לבדיקה. הכוונה בעיקר היא לגודל ה N-Gram, גודל החלון וסוג הבדיקה הסטטיסטית שנבחרה. כל מקרה של בחירה שונה בפרמטרים אלו תפגע בתוצאות המבחן ותחזיר תוצאות שגויות, אם בכלל.

4.4.2.2 בדיקת הקבצים

התכנית testFiles משמשת לחישוב הסבירות של הקבצים הנבדקים להיות מטיפוס מסויים. התכנית מקבלת מספר פרמטרים:

- קובץ או ספריית הקלט – קובץ זה או הקבצים הכלולים בספריה יבחנו לצורך קביעת הסוג שלהם.
- קובץ הפלט, אליו יירשמו תוצאות הבדיקה.
- קובץ החתימה – קובץ התוצאה של מסלול ייצור החתימה (ראה סעיף 4.4.1.6).
- קובץ חוקי הציון - קובץ זה מכיל מערכת חוקים ותנאים לקביעת הציון המוענק לכל N-Gram בכל טיפוס (ראה סעיף 4.4.2.4).
- התכנית מפעילה את התהליך הבא:
- טעינת קובץ החתימה ושמירתו בזיכרון במבנה נתונים מתאים.
- טעינת קובץ חוקי הציון לזיכרון.
- עבור כל קובץ...
    - עבור כל טיפוס הנמצא בחתימה...
        - חשב את ציון הקובץ.
    - בחר בטיפוס בעל הציון הגבוה ביותר.

תוצאת הריצה נרשמת לקובץ הפלט בפורמט CSV, כדוגמה שמוצגת בתרשים הבא. העמודה הראשונה מציגה את שם הקובץ. העמודה שלאחריה מציגה את הטיפוס/ים שקיבל/ו את הציון הגבוה ביותר. העמודות הבאות מציינות, עבור כל טיפוס בחתימה, את הציון שקיבל הקובץ עבור טיפוס זה.

יש לשים לב, לדוגמה בשורה המודגשת הראשונה, שלמרות שהקובץ הוא מטיפוס dll נבחרה ההתאמה הטובה ביותר ל exe. אופיים הדומה של שני טיפוסים אלו עשוי להסביר את התופעה. בשורה המודגשת השניה ניתן לראות שלקובץ ה dll שנמצאה התאמה זהה לטיפוס dll ו – exe ובשתי המקרים הציון הוא 100, הציון המקסימאלי האפשרי.

File	Best Match	dll	docx	exe	pdf	rtf	zip
102_381.rtf	rtf	0	1	0	5	60	0
102_694.rtf	rtf	0	1	0	8	69	0
102_517.rtf	rtf	0	1	0	7	71	0
102_607.rtf	rtf	0	1	0	7	71	0
102_383.rtf	rtf	0	2	0	8	61	0
102_137.rtf	rtf	0	1	0	7	69	0
102_268.rtf	rtf	0	1	0	10	70	0
102_290.rtf	rtf	0	1	0	12	78	0
102_485.rtf	rtf	1	1	0	6	57	0
102_378.rtf	rtf	0	2	0	10	61	0
102_483.rtf	rtf	1	1	0	4	66	0
102_708.rtf	rtf	20	12	18	76	99	0
102_718.rtf	rtf	25	14	20	65	96	0
L_53_2003.zip	zip	5	8	7	0	0	81
Microsoft.PowerShell.Commands.Utility.resources.ni.dll	exe	67	32	81	68	37	52
102_588.rtf	rtf	21	18	17	85	100	0
102_707.rtf	rtf	28	18	23	81	87	0
inetres.dll	dll	100	32	97	42	19	52
MFC71CHT.DLL	dll	100	20	94	3	0	52
102_595.rtf	rtf	4	7	6	51	98	0
102_675.rtf	rtf	8	13	9	69	99	0
MFC80CHT.dll	dll	100	19	94	3	0	52
102_723.rtf	rtf	25	19	24	83	100	0
102_599.rtf	rtf	22	14	19	64	96	0
php_mcrypt.dll	dll;exe	100	17	100	39	15	44
102_565.rtf	rtf	23	13	20	65	96	0

תרשים 30: תוצאות בדיקת טיפוס עבור קבצים לא ידועים

4.4.2.3 חישוב ציון ההתאמה לטיפוס

השלב המהותי ביותר באבחון הקובץ, הוא תהליך הענקת ציון ההתאמה לטיפוס מסויים, הקובע את דירוג הקובץ עבור טיפוס זה ביחס לאחרים. בשלב ההחלטה על הציון נלקחים בחשבון ה - N-Grams שנמצאו, אחוז המופעים של N-Grams אלו מתוך סך כל הקבצים ששימשו כמדגם לחתימה וקיום תנאי זה או אחר, כפי שהם מופיעים בקובץ חוקי הציון.

ראשית, נזכור כי בקובץ החתימה נמצא, בצמוד לשם הטיפוס, את סך כל ה – N-Grams המהווים חלק מהחתימה עבור קובץ זה. בעת חישוב הציון, כל N-Gram שנמצא בקובץ נבדק להמצאות בחתימה. אם אינו נמצא, הוא אינו תורם מאום. אם ה – N-Gram נמצא בחתימה, הוא יקבל ציון בתחום 0% – 100%, וזאת בהתאם לתנאי הראשון שיתקיים בקובץ חוקי הציון עבור ה - N-Gram וטיפוס זה. ציון ה – N-Gram יוכפל במשקלו ביחס לסך כל ה – N-Grams שנמצאו עבור טיפוס זה.

הציון הסופי יחושב לפי הנוסחה:

$$G = \frac{\sum G_n W_n}{\sum W_n}$$

כאשר נתוני המשוואה הם:

- G_n – הציון שקיבל N-Gram מסוים.
- W_n – המשקל שניתן לציון N-Gram זה, כמות הקבצים בהם הופיע ה – N-Gram במדגם.

קל לראות שבמקרה שכל ה – N-Grams בחתימת הטיפוס יהיו בקובץ הנבדק וכל N-Gram יקבל ציון 100%, הציון הסופי של הקובץ עבור הטיפוס יהיה 100. תרומתו של כל N-Gram לציון שקולה לאחוז מופעיו בקבצים ששימשו למדגם. כך מובטח ש – N-Gram שהופיע ביותר קבצים במדגם, ומכאן הוא אופייני יותר לטיפוס, יתרום לציון הסופי יותר מ – N-Gram שהופיע בפחות קבצים. לדוגמה, אם נקבע שסף הכניסה של N-Grams לחתימת הטיפוס הוא הופעה לפחות ב 50% מקבצי המדגם ואכן יימצא N-Gram שנמצא רק ב 50% מהקבצים, גם אם יימצא בקובץ הנבדק, הוא יתרום לציון חצי מתרומתו של N-Gram שנמצא בכל קבצי המדגם וגם בקובץ הנבדק.

4.4.2.4 קובץ חוקי הציון

הציון, אותו יקבל כל N-Gram עבור טיפוס מסוים, יקבע בהתאם לחוקים והתנאים הנמצאים בקובץ חוקי הציון. הקובץ במבנה XML ומכיל חוק עבור כל טיפוס שנרצה לבדוק, כשכל חוק מורכב ממספר תנאים, אלו או אחרים.

מהי משמעות התנאי? זו למעשה הדרך של של הארכיטקטורה לקשר בין הנתונים הסטטיסטיים שנאספו בעת ייצור החתימה, לבין נתוני ה – N-Gram, כפי שנמצאו בקובץ אותו אנו בוחנים. אם נחזור בקצרה על שלבי בניית החתימה תוך התמקדות באיסוף הנתונים:

- בשלב האנליזה סטאטיסטית/סידור קבצים, בחרנו את הבדיקה הסטטיסטית שנרצה לערוך על כל ה – N-Grams שנמצאו.
- בשלב איחוד תוצאות בחרנו את העמודות, אותן נרצה לאסוף, ואת החישובים הסטטיסטיים אותם נרצה לבצע עבור כל עמודה.
- בשלבי השוואת תוצאות איחוד ויצור החתימה בחרנו את העמודות "המעניינות" שנרצה להוסיף לחתימה, מתוך כל העמודות המחושבות בשלב איחוד התוצאות.

מכאן שבשלב זה זמינות כל העמודות שאספנו לצורך השוואה עם נתוני הקובץ, אותו אנו בוחנים.

תבנית קובץ החוקים מתוארת במבנה BNF הבא:

<rules_def>	::=	"<Rules>" <rules> "</Rules>"
<rules>	::=	<rules> <rule> | ""
<rule>	::=	"<" <type_text> ">" <tests> "</" <type_text> ">"
<type_text>	::=	"exe" | "dll" | "docx" | "pdf" | ……
<tests>	::=	<tests> <test> | <test>
<test>	::=	"<Test grade=" <grade> ">" <condition> "/Test"
<grade>	::=	"[0 - 100]"
<condition>	::=	<cond_part> | <cond_part> <bin_log_oper> <cond_part>
<bin_log_oper>	::=	"And" | "Or"
<cond_part>	::=	"true" | <lhs> <rel> <rhs>
<lhs>	::=	"file_field[" <field_num> "]"
<rhs>	::=	"signature_field[" <field_num> "]"
<rel>	::=	"<" | "<=" | "=" | "=>" | ">"
<field_num>	::=	"[0 - …]"

כפי שציינתי, כל חוק מכיל תנאי אחד או יותר. סדר התנאים בתוך החוק משמעותי היות והתנאי הראשון המתברר כנכון מחזיר את הערך שמוצמד לו ובדיקת החוק מסתיימת. התנאים מתבססים על הערכים שנמצאו בעת סריקת הקובץ עבור המדדים הסטטיסטיים השונים. ערכים אלו זמינים לבחינת התנאים בתוך שני מערכים שונים. מערך אחד מכיל את הערכים הסטטיסטיים שהתקבלו עבור ה – N-Gram הנדון בסריקת הקובץ והשני את הערכים הסטטיסטיים כפי שנמצאו בחתימת הטיפוס עבור אותו ה – N-Gram. כך שלדוגמה ניתן לקבל את מספר מופעי ה – N-Gram שנמצאו בקובץ באיבר השני של מערך נתוני הסריקה ואת מספר המופעים המינימאלי של ה – N-Gram, בסריקת כל הקבצים המייצגים של הטיפוס הנבדק, במקום הראשון במערך נתוני החתימה.

זמינות נתונים אלו מאפשרת לקבוע תנאים כגון: "אם מספר ה – N-Grams שנמצאו בסריקה קטן מהמספר המינימאלי של ה – N-Grams בחתימה, החזר את הציון 50."

במימוש הדוגמה השתמשתי בקוד PERL. המערך המייצג את נתוני הסריקה מסומן כ - sf (**S**ignature **F**ields) בו העמודה הראשונה נמצאת במקום 0 ושאר העמודות, לפי סדר הבחירה בעת יצור החתימה, נמצאות לאחריה. מכאן שאם בחרנו, ביצור החתימה, בעמודות המתיחסות למינימום המופעים של N-Gram בקבצים, מקסימום המופעים וממוצע המופעים, נקבל לפי אותו הסדר את ערך המינימום ב – sf[0], ערך המקסימום ב – sf[1] ואת ערך הממוצע ב – sf[2].

המערך המייצג את ערכי החתימה עבור ה – N-Gram מסומן כ - ff (**F**ile **F**ields). גם כאן נמצא את העמודה הראשונה משמאל ולאחריה, לפי סדר ההופעה, את שאר העמודות. יש לשים לב כי העמודה הראשונה, אליה אנו מתייחסים, היא העמודה הראשונה לאחר עמודת ה – N-Gram. מכאן שעמודה ff[0] תכלול בדרך כלל תוצאת החישוב הסטטיסטי עבור ה – N-Gram ועמודה ff[1] תכלול את מספר המופעים של N-Gram זה בקובץ.

היות והתנאי כתוב בקוד PERL כחלק מקובץ XML, הקוד עבר קידוד מתאים על מנת שיעמוד בתנאים של קובץ מסוג זה. מכאן שהתנאי

$ff[1] >= $sf[0] && $ff[1] <= $sf[1]

שקול למעשה ל:

ff[1] >= sf[0] && ff[1] <= sf[1]

שהוא בתורו שקול לשאלה, האם, עבור ה – N-Gram הנבחן כעת, הערך בעמודה השנייה של הקובץ גדול או שווה לערך בעמודה הראשונה בחתימה וגם האם הערך בעמודה השנייה של הקובץ קטן או שווה לערך בעמודה השנייה בחתימה. אם התנאי נכון, התכנית תעניק ל – N-Gram את הציון הצמוד לו.

```
<?xml version="1.0" encoding="UTF-8"?>
<Rules>
    <exe>
        <Test grade="100">$ff[1] &gt;= $sf[0] &amp;&amp;  $ff[1] &lt;= $sf[1]</Test>
        <Test grade="80">$ff[1] &lt;= $sf[0]</Test>
        <Test grade="50">1</Test>
    </exe>
    <dll>
        <Test grade="100">$ff[1] &gt;= $sf[0] &amp;&amp;  $ff[1] &lt;= $sf[1]</Test>
        <Test grade="50">1</Test>
    </dll>
    <docx>
        <Test grade="100">$ff[1] &gt;= $sf[0] &amp;&amp;  $ff[1] &lt;= $sf[1]</Test>
        <Test grade="50">1</Test>
    </docx>
    <rtf>
        <Test grade="100">$ff[1] &gt;= $sf[0] &amp;&amp;  $ff[1] &lt;= $sf[1]</Test>
        <Test grade="50">1</Test>
    </rtf>
    <pdf>
        <Test grade="100">$ff[1] &gt;= $sf[0] &amp;&amp;  $ff[1] &lt;= $sf[1]</Test>
        <Test grade="50">1</Test>
    </pdf>
    <zip>
        <Test grade="100">$ff[1] &gt;= $sf[0] &amp;&amp;  $ff[1] &lt;= $sf[1]</Test>
        <Test grade="50">1</Test>
    </zip>
</Rules>
```

תרשים 31: קובץ חוקי הציון

בתרשים הקובץ למעלה, לדוגמה, מודגש החוק עבור טיפוס מסוג dll. שם התווית מסמן את שם הטיפוס אליו מתייחס החוק. החוק מכיל תנאים שונים בסדר מוגדר מראש. עבור טיפוס מסוג exe למשל, מודגש התנאי הראשון. התווית המייצגת תנאי היא <Test>. בתור התווית נמצא את המאפיין "grade" המציין את הציון שיקבל ה – N-Gram אם יתקיים התנאי המופיע בתוכן התווית. כזכור, סדר התנאים משמעותי והתנאי הראשון שמתקיים הוא הקובע היות והבדיקה מסתיימת מיד אחריו, לכן גם אם N-Gram עומד בשני תנאים שונים, הראשון שייתקיים יקבע את ציונו.

יש לשים לב שהתנאי האחרון הוא '1' המייצג "true" שתמיד מתקיים. בצורה זו ניתן להגדיר את הציון עבור N-Gram שנמצא, אולם לא התקיים לגביו אף תנאי אחר בעל משמעות. באופן טרוויאלי ניתן להגדיר קובץ ובו כל החוקים יהיו מסוג זה וכל הציונים יהיו 100%, כך שעצם הימצאות ה – N-Gram תתרום אך ורק את משקלו לציון, ללא התחשבות בבדיקה סטטיסטית זו או אחרת.

האלגוריתם הבא מתאר את פעולת חישוב תרומת N-Gram מסוים לציונו של קובץ עבור טיפוס מסויים על בסיס החוק:

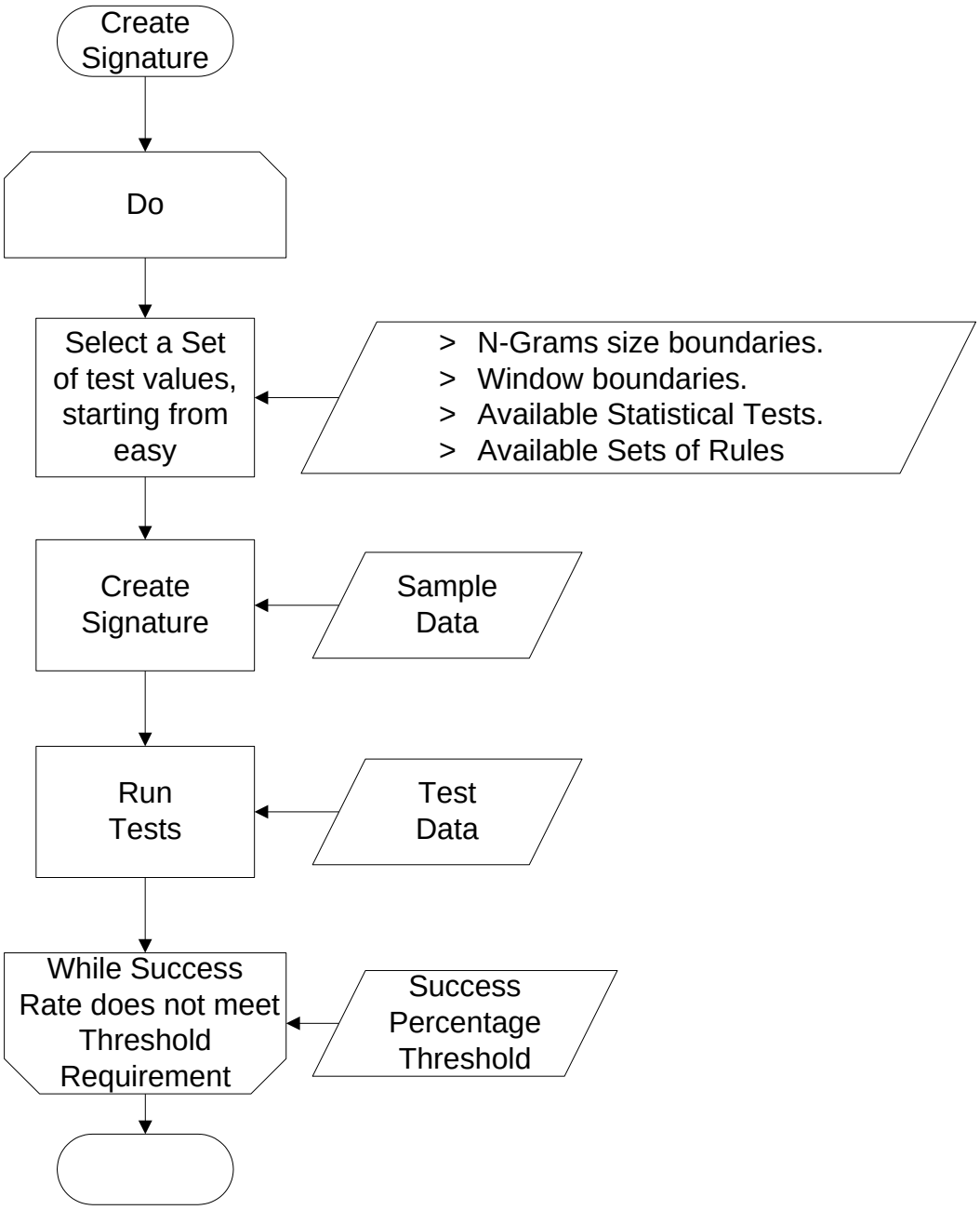
For each $r \in Rules$
  If $r.Type$ matches current type
    For each $c \in r.Conditions$
      If c is true
        Add $(c.Grade * NGram\ weight)$ to the file type total score
        Break

ניקח לדוגמה את החוקים המופיעים בתרשים. העמודות שנלקחו עבור החתימה הן, עבור כל N-Gram, מינימום המופעים בעמודה הראשונה ומקסימום המופעים בעמודה השנייה. בנתוני הקובץ מופיע נתון ממוצע המופעים בעמודה הראשונה ומספרם בשנייה. אם נבחן את התנאים עבור קובץ מסוג exe נראה שמשמעותו של החוק עבור הטיפוס:

- אם מספר המופעים של ה – N-Gram בקובץ נמצא בתחום המתאים בין המינימום והמקסימום שבחתימה, הענק את הציון 100% ל – N-Gram.
- אחרת, אם מספר המופעים של ה – N-Gram קטן ממספר המופעים המינימלי שלו בחתימה, הענק את הציון 80% ל – N-Gram.
- אחרת, הענק את הציון 50% ל – N-Gram (מעצם העובדה שהופיע בקובץ הנבדק).

4.5 מתודולוגיה

הארכיטקטורה מגדירה מסגרת עבודה, המשמשת לאיתור הפרמטרים המתאימים ביותר לצורך יצירת חתימה מדויקת כנדרש. בסעיף זה נתמקד בשיטה בה ינקוט החוקר, בבואו לחקור טיפוס חדש לצורך יצירת חתימה עבורו. מתודולוגיה זו עשויה אף לשרת תכנון של מערכת אוטומטית ליצירת חתימות, אשר תדע לקבל את תחומי ערכי הפרמטרים המתאימים כקלט, להריץ נסיונות באופן אוטומטי ולהחזיר את החתימה שתניב את התוצאות המדויקות ביותר, או החתימה הראשונה שתעמוד בדרישות הסף. בנוסף, המערכת יכולה להחזיר את הערכים המדוייקים של הפרמטרים שהביאו לתוצאה המקווה.



תרשים 32: מתודולוגיה למציאת חתימה לטיפוס קובץ

התרשים מתאר סדר פעולות מוצע לצורך איתור חתימה מתאימה לטיפוס, אשר תעמוד בדרישות האיכות הנדרשות, במינימום מאמץ. התהליך המוצע הוא חזרתי (Iteration) עד לקבלת התוצאה המקווה, או כישלון. שלבי התהליך מתוארים להלן.

4.5.1 קביעת סף הצלחה לזיהוי

בחירת הערך באחוזים לזיהוי קבצים מתוך המדגם לריצה תקבע את הסף בין הצלחה וכישלון. הצלחה נמדדת על פי כמות הניחושים המוצלחים שניחשה המערכת לגבי שייכות קובץ לסוג הטיפוס הנבדק. נקודת ההנחה היא שעבור קבצי מדגם השייכים לטיפוסים אחרים, כבר קיימת חתימה ונעשה בה שימוש במבחן. ככישלון ייחשבו זיהוי שגוי של קובץ מטיפוס אחר, כשייך לטיפוס הנבדק (טעות מטיפוס I), או זיהוי של קובץ השייך לסוג הנבדק, כטיפוס אחר (טעות מטיפוס II).

4.5.2 בחירת מדגם מייצג לטיפוס ולמבחן

המדגם המייצג של הטיפוס חייב להכיל קבצים שהגיעו ממקורות מוכרים ונבדקו על מנת לוודא את שייכותם לסוג הנבדק, אם על ידי הפעלתם הפיזית על ידי היישום לו הם מיועדים, או בחינת תוכנם, במקרה של קבצים טקסטואליים. למותר לציין את הצורך לוודא את נקיונם של הקבצים מקוד זדוני וזאת על ידי שימוש בתוכנות סריקה כאנטי וירוס ותוכנות הגנה אחרות. לכמות הקבצים במדגם חשיבות מכרעת היות וכמות קטנה יותר תניב תוצאות מדוייקות פחות. כמו כן, היות והמטרה ביצירת החתימה היא למצוא מכנה משותף בין הקבצים עד רמת דיוק מסויימת, אין צורך להשתמש בקבצים בעלי גודל פיזי גדול היות ואלו לא יתרמו לרמת דיוק החתימה, אולם עלולים להשפיע על ביצועי המערכת בעת יצירתה.

הקבצים המשמשים במדגם לריצת המבחן חייבים להיות מטיפוסים שונים. חלקם הגדול מהטיפוס הנבחן ואחרים מטיפוסים, להם כבר קיימת חתימה. בעת המבחן ייעשה שימוש בכל החתימות הידועות עבור הטיפוסים האחרים. מטרת השימוש בקבצים מטיפוסים שונים היא לאתר טעויות מסוג I ו – II. שימוש אך ורק בקבצים מהסוג הנבדק או אך ורק בחתימה הנבדקת לא יאפשרו השוואה מעשית של תוצאות ריצת המבחן.

4.5.3 קביעת תחומי הערכים האפשריים לפרמטרים שונים

קביעת תחומי הערכים האפשריים עבור החזרות במבחנים נוגעת לפרמטרים הבאים:

- גבולות גודל ה – N-Gram. הערכים שייבחרו הם הגודל המינימלי והמקסימלי של N-Gram שייבחן. בעת בחירת התחומים יש לקחת בחשבון שבחירת ערכי סף נמוכים מדי, לדוגמה 2, לא תניב תוצאות מדוייקות שכן מספר האפשרויות השונות לקבלת N-Gram בגודל זה מוגבל ל 2^{16} . בחירת ערכי סף גבוהים מדי, לעומת זאת, עשויה לגרום למערכת לקרוס בשל חיסרון במשאבים. בעת הריצה, ייבחרו ראשית הערכים הנמוכים במטרה להשיג זמן ריצה מינימלי.
- כגודל ה N-Gram, כך גודל החלון ישפיע על ביצועי הבדיקה, אולם עשוי לשפר את תוצאות מבחן הריצה. הגודל המינימאלי שיבחר יהיה בדרך כלל גודל ה – N-Gram. גם במקרה זה בחירת הערכים תחל מהערך הנמוך.

- סט המבחנים הסטטיסטיים הזמינים. סדר הבחירה במבחנים יחל מהמבחן הקל ביותר בצריכת משאבים.
- סטים שונים של חוקים, החל מהחוק הטרוויאלי המחזיר ציון של 100% עבור המצאות N-Gram וכלה בחוקים מורכבים יותר, הבנויים ממספר תנאים שונים. חוקים אלו ימויינו לפי רמת המורכבות, מהקל לכבד, וייבחרו על פי סדר זה.

4.5.4 תהליך הבדיקה

תהליך הבדיקה הוא חזרתי, עד לקבלת תוצאות שייספקו את סף ההצלחה לזיהוי. כל חזרה בתהליך מורכבת מהצעדים הבאים:

4.5.4.1 בחירת סט ערכים לריצה הבאה

בשלב זה ייבחרו הערכים המתאימים לפרמטרים השונים של הריצה. כפי שציינתי בסעיפים הקודמים, בחירת הערכים תיעשה בסדר עולה, לפי השפעתם על צריכת המשאבים. המטרה בכך היא להשיג את התוצאה המקווה במינימום מאמץ. נסמן את תחומי הערכים האפשריים לפרמטרים כ – A, B, … .
הקוד הבא מתאר את תהליך בחירת הערכים לפרמטרים:

For each $\in A$, in ascending complexity order
  For each $b \in B$ in ascending complexity order
  .
  .
    Run Test with (a, b, …), On Success, Stop.

לצורך ייעול התהליך, נוכל לבחור את סדר הפרמטרים כך שאלו המשפיעים פחות על צריכת המשאבים, ימצאו בלולאות הפנימיות של הקוד, כך שנגיע לבחירת הערכים המשפיעים ביותר על המשאבים, לקראת סוף תהליך הבחירה..

4.5.4.2 יצירת חתימה

בשלב זה נשתמש במדגם המייצג לטיפוס ובערכי הפרמטרים שנבחרו בסעיף הקודם על מנת לייצר חתימה לפי התהליך המתואר בסעיפים המוקדמים. חתימה זו תשמש בריצת המבחן. ניתן לבחון את איכות קבוצת המדגם על ידי פיצול אוכלוסייתה ל K קבוצות שונות. בכל פעולה נבחרת אחת הקבוצות כקבוצת מבחן ושאר K-1 הקבוצות משמשות לייצור החתימה. הפעולה תחזור K פעמים. יתרון השיטה הוא בכך שהמדגם משמש הן לייצור החתימה והן לאישורה. שיטה זו, המכונה K-Fold Cross Validation, מקלה על איתור החריגים בקבוצת המדגם והסרתם. דוגמה לשימוש בשיטה ניתן לראות בסעיף 4.6.4.

4.5.4.3 ריצת מבחן

בריצת המבחן נעשה שימוש בחתימה מהסעיף הקודם, בצירוף כל החתימות הקיימות עבור הטיפוסים האחרים אשר במדגם המייצג למבחן. תוצאות הריצה יבוטאו בציון התאמה שיינתן לכל קובץ עבור כל טיפוס. קובץ ייחשב מטיפוס מסויים אם יקבל את הציון הגבוה ביותר עבור טיפוס זה.

4.5.4.4 השוואת תוצאות הריצה עם סף ההצלחה

בשלב זה בוחנים את תוצאות הריצה עבור כל קבצי המדגם. הצלחה נחשבת כזיהוי של קובץ מהטיפוס הנבחן ושלילת קובץ שאינו מהטיפוס הנבחן, קרי, תשובה של כן, אם שייך, או לא במקרה ואינו שייך. היות והמטרה היא מציאת חתימה מתאימה לטיפוס מסוים, לא נראה כטעות זיהוי שגוי של קובץ מטיפוס שאינו נבדק, כטיפוס אחר שאינו נבדק. אחוז ההצלחה יחושב בעזרת מספר התוצאות המדוייקות ומספר הקבצים במדגם. תוצאה זו תושווה לסף ההצלחה. אם תתקבל תוצאה טובה יותר מהסף, הדבר ייחשב כהצלחה והמערכת תחזיר את החתימה שהתקבלה והפרמטרים ששימשו לקבלתה. במקרה של כישלון במעבר הסף, יבחר סט פרמטרים חדש והתהליך יבוצע שנית.

4.6 הוכחת הרעיון (POC)

לצורך הוכחת יעילות הרעיון המוצג בארכיטקטורה, בחרתי בבדיקה בסיסית ביותר, וזאת במטרה לשמור על פשטות ההדגמה ועם זאת, להדגים את חוזק השיטה. ההוכחה בוצעה על מערכת הפעלה חלונות אולם חשוב לציין כי הרעיון המוצג בתזה זו אינו מוגבל למערכת הפעלה זו או אחרת.

4.6.1 נתוני מדגם

האתגר הראשון בהוכחת הרעיון היה מציאת מקורות מספקים ואמינים לקבצי מדגם ומבחן. על מנת להקטין את האפשרות למקריות בקבלת התוצאות שאפתי לכמות מספקת של קבצים מכל טיפוס אותו בחנתי. בין מקורות הקבצים מהם שאבתי ניתן למנות מאגר של חברה העוסקת בתחום האבטחה, חיפוש ב – Google עבור טיפוסים מסויימים וחיפוש על המחשב האישי, בעיקר אחר קבצים מסוג exe ו – dll.

הטיפוסים וכמות הקבצים ששימשו בהוכחת הרעיון מפורטים בטבלה הבאה.

טיפוס הקובץ	כמות לחתימה	כמות מדגם	כמות כללית
exe	500	200	700
dll	500	699	1199
docx	500	182	682
pdf	400	19	419
rtf	500	255	755
zip	120	45	165
סה"כ	2520	1400	3920

טבלה 2: טיפוסי קבצים ששימשו למדגם ונתוני כמות

4.6.2 מהות הבדיקה

כפי שציינתי, הבדיקה לצורך הוכחת נכונות הינה בסיסית ביותר במהותה. ה – N-Grams שנבדקו היו בגודל 3 עם חלון בגודל זהה. בשלב איסוף הנתונים, לא בוצעה אף בדיקה סטטיסטית ונעשה שימוש אך ורק בתכנית השירות Tidy. במהלך ייצור החתימה נאספו נתוני המינימום והמקסימום עבור כל N-Gram שנכלל בחתימה, ועבור כל N-Gram שנמצא בקובץ נבדק:

- אם אינו נמצא בחתימה, הענק את הציון 0.
- אחרת, אם מספר המופעים שלו בקובץ נמצא בתחום המתאים בין המינימום והמקסימום שבחתימה, הענק את הציון 100%.
- אחרת, הענק את הציון 80%.

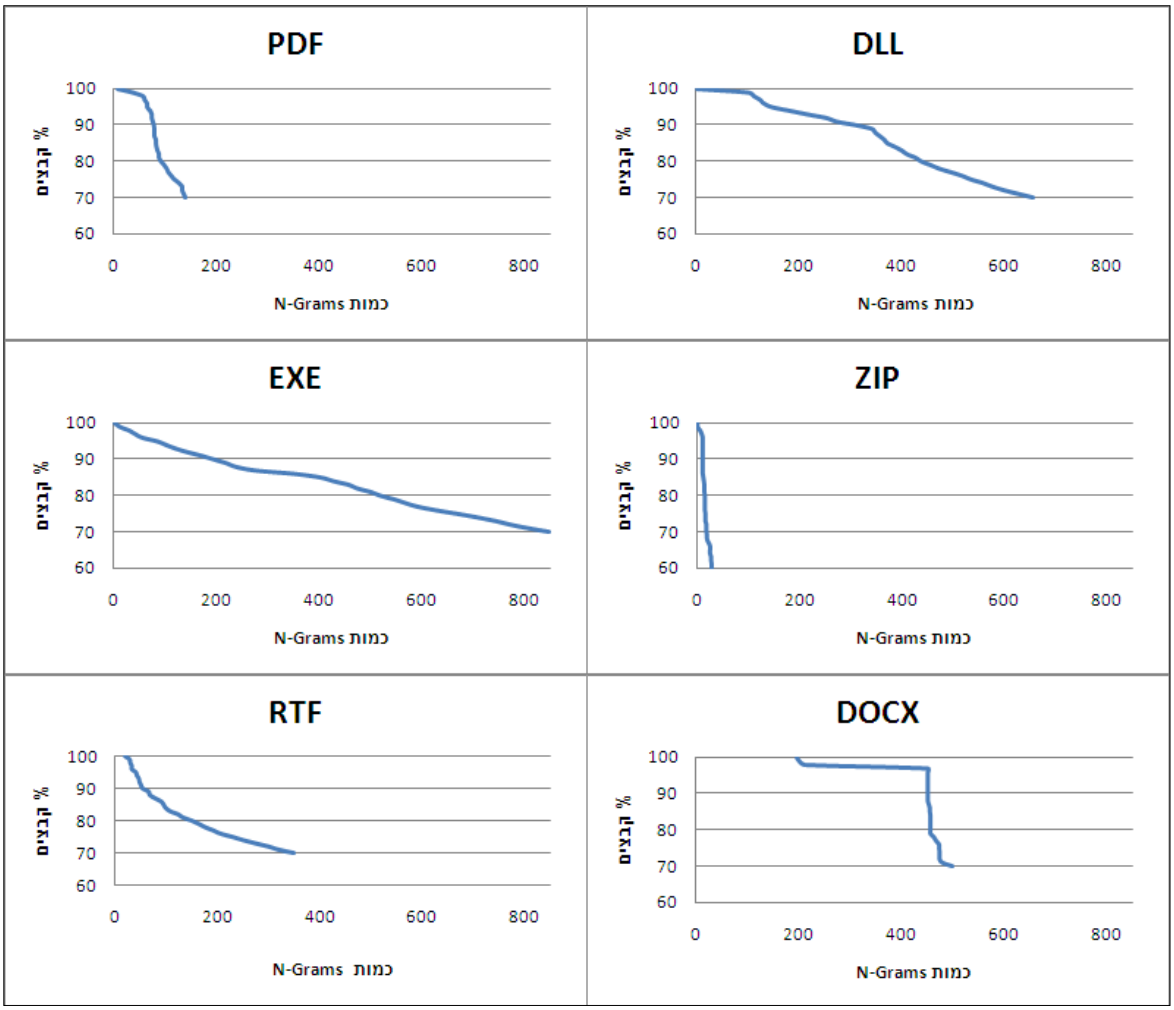
כמובן שבמימוש מעשי, נעשה שימוש נרחב יותר בנתונים סטטיסטיים שייאספו ובמגוון עשיר יותר של תנאים וזאת על מנת לקבל תוצאות אופטימליות. לדוגמה, נוכל לאסוף את נתוני סטיית התקן ולבדוק את התחום בו נופל הערך הנבדק, על מנת לתת ציון בהתאם. חוקים אילו נבחרו בצורה אחידה עבור כל הטיפוסים בהדגמה. כמובן שקיימת האפשרות לבחור חוקים ותנאים שונים עבור טיפוסים שונים.

4.6.3 מציאת סף כניסה עבור N-Grams

את סף הכניסה של N-Grams ניתן לקבוע בשתי דרכים:

- הגבלת אחוז מופע בקבצים – בדרך זו קובעים סף מינימום לאחוז הקבצים בהם יופיע ה - N-Gram על מנת שיכלל בחתימה. החיסרון העיקרי בשיטה זו הוא שבמקרה של טיפוסים בעלי מבנה פחות מובהק, אנו עשויים לקבל חתימה בעלת מספר קטן של N-Grams. במקרה זה קיים החשש לזיוף קל יותר של הקובץ על ידי שתילת אותם N-Grams בתוכנו. בעיה נוספת תתעורר כאשר נבחן קובץ גדול במיוחד, אז גדלה האפשרות שאותו מספר מועט של N-Grams יימצא בו, למרות שלא יהיה מהטיפוס הנדון.
- הגבלת מספר מינימאלי של N-Grams בחתימה – במקרה זה המשתנה יהיה מספר הקבצים המינימלי, בהם יופיע ה – N-Gram. יתרונות שיטה זו על הקודמת ברורים אולם חסרונה העיקרי הוא שמציאת ה – N-Grams דורשת יותר משאבי זיכרון ודורשת טיפול מיוחד, במקרה של מעבר על מגבלה זו.

מהשיקולים שצויינו למעלה בחרתי בשיטה השנייה לקביעת סף הכניסה. הגבול שבחרתי, בצורה שרירותית, עמד על מינימום 100 N-Grams לחתימה של כל טיפוס. למעט קבצים מסוג zip, בהם מגבלות זיכרון מנעו ממני מציאת 100 N-Grams יחודיים, הצלחתי לעשות זאת לגבי כל השאר הטיפוסים.



תרשים 33: כמות N-Grams לפי סף כניסה של קבצים עבור טיפוסים שונים

התרשים למעלה מציג את כמות ה – N-Grams החוזרים בקבצים השונים עבור כל טיפוס שנבחן, בהשתנות סף הכניסה של אחוז הקבצים, בהם נמצא ה N-Gram. הנתונים מתייחסים כמובן ל - N-Gram וחלון בגודל 3. אם נבחן לדוגמה את הגרף המייצג את הטיפוס exe, במקרה של סף כניסה בן 90% עבור מופעי N-Grams בקבצים, נקבל כ – 200 N-Grams שונים העומדים בסף זה. לצורך השוואה בין מאפייני הטיפוסים השונים, כל הגרפים קיבלו את אותם ערכי אורך עבור הצירים.

השוואה בין הגרפים השונים מגלה הבדל ניכר בין הטיפוסים השונים. קל לראות שלטיפוסים dll, exe ו – docx רמת מובהקות גבוהה יותר. במיוחד בולטים הטיפוס exe, בו ניתן למצוא מעל 800 N-Grams שונים במקרה של סף בן 70%, והטיפוס docx, בו יש מספר גבוה במיוחד של כ – 200 N-Grams שונים שיופיעו בכל הקבצים. לעומתם, הטיפוסים pdf, rtf ו – zip מכילים מאפיינים בולטים פחות. בולט לרעה הטיפוס מסוג zip המכיל מספר קטן ביותר של N-Grams, מן הסתם בשל אופי הנתונים בתוכו, נתוני ארכיון מכווצים. במקרה של הטיפוס zip, נאלצתי להוריד את סף הכניסה ל 55% על מנת לאתר לפחות 30 N-Grams שונים. החדשות הטובות הם שדווקא בקובץ מסוג ארכיון קיימים כלים נוחים לבדוק אם אכן הקובץ מטיפוס זה, למשל על ידי נסיון לפתוח אותו בעזרת תוכנה יעודית.

על מנת לקבל לפחות כ - 100 N-Grams לחתימה, נבחרו ערכי הסף כמפורט בטבלה הבאה.

טיפוס	סף כניסה
DLL	99%
PDF	79%
EXE	94%
ZIP	55%
RTF	84%
DOCX	100%

טבלה 3: סף כניסה נבחר עבור הטיפוסים השונים

4.6.4 בדיקה פנימית עבור כל טיפוס

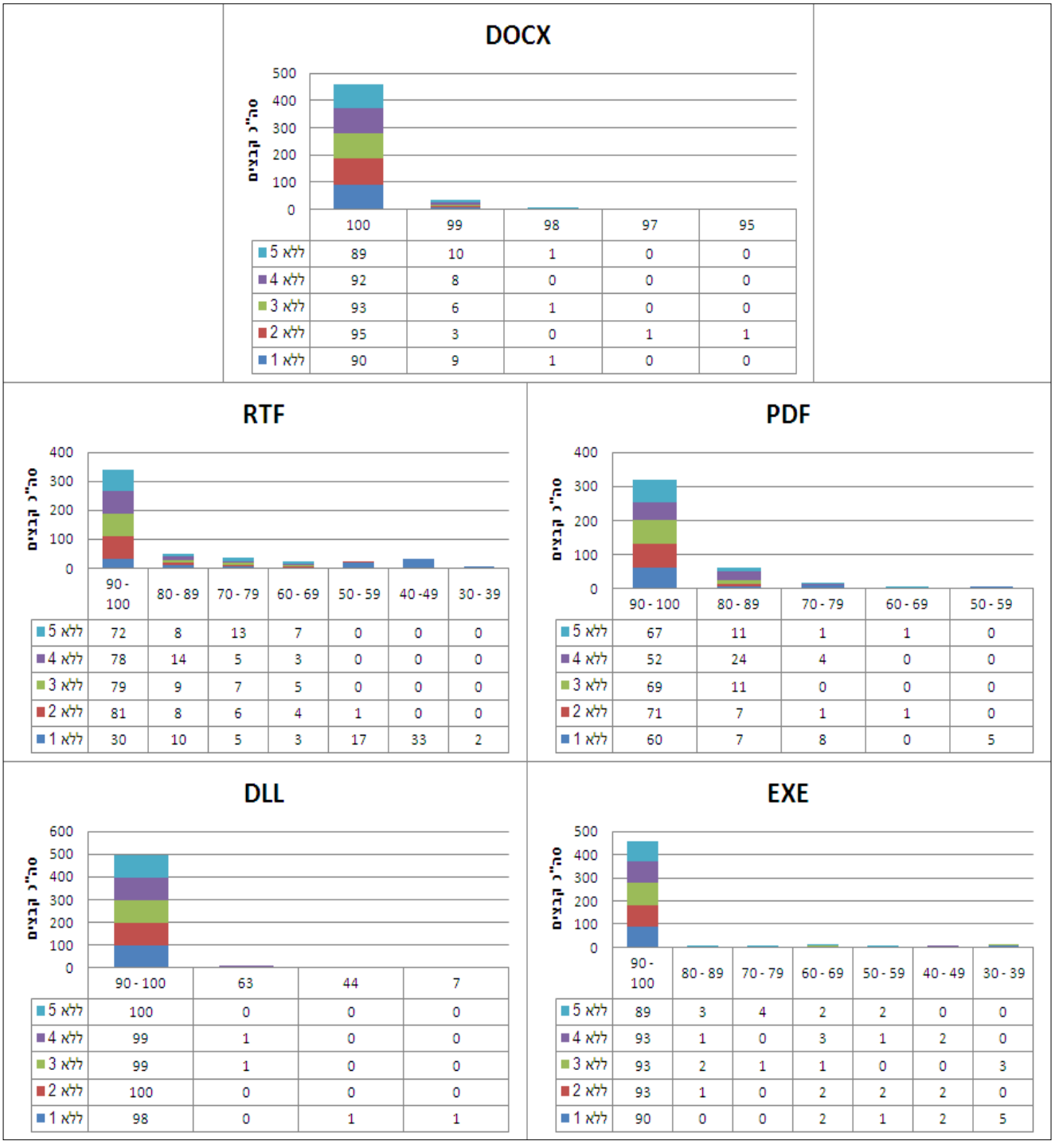
השלב הראשון בהוכחת היכולת היה לבצע בדיקת K-Fold Cross Validation כשעבור K נבחר הערך 5. ראשית נבחרו בצורה שרירותית אותם קבצים שישמשו לחתימה המייצגת של הטיפוס. הקבצים הנבחרים חולקו לחמש קבוצות שונות, אחידות בגודלן, לצורך בדיקה עבור כל קבוצה ביחס לאחרות, לקבלת הציון עבור קבצי הקבוצה. לדוגמה, עבור הטיפוס dll נבחרו 500 קבצים לשמש בחתימה. הקבצים הנבחרים חולקו ל – 5 קבוצות שונות בנות 100 קבצים בכל אחת. עבור כל קבוצה יוצרה חתימה מ – 400 הקבצים שהיוו את ארבע הקבוצות האחרות ולבסוף נבחנו קבצי הקבוצה לקבלת ציון ביחס לחתימה זו.

הבדיקה נערכה עבור כל טיפוסי המדגם, למעט קבצים מטיפוס zip, ותוצאותיה מפורטות בתרשים הבא. עבור כל טיפוס, הטבלה מציינת את התוצאות שהתקבלו עבור הקבוצות החלקיות. לדוגמה, עבור טיפוס RTF, בעזרת חתימה שנוצרה על ידי תת הקבוצות 5 – 2, קיבלו 30 מקבצי הקבוצה הראשונה ציון בתחום 100 – 90, 10 קיבלו ציון בתחום 89 - 80, 5 קיבלו ציון בתחום 79 – 70 וכן הלאה. בנתונים המוצגים, החלוקה לקבוצות ציונים שונות נבחרה על פי פיזור הציונים. במקרה של פיזור רב, נבחרו תחומים בני 10 נקודות לאיגוד הנתונים. במקרים אחרים, כמו לדוגמה בקבצים מטיפוס docx, מוצגים הציונים המסויימים בשל הפיזור הנמוך של הנתונים.

בחינת התוצאות מעלה תוצאה מעניינת, אך צפויה. טיפוסים, להם רמת מובהקות בולטת יותר, השיגו ציונים טובים יותר לעומת אלו בעלי המאפיינים הפחות בולטים. לדוגמה, הקבצים מטיפוס docx השיגו כולם, תוצאה בתחום 100 – 95, מתוכם כ – 92% השיגו את הציון המלא 100. כך גם קבצים מסוג dll ו – exe השיגו תוצאות טובות, למעט קובץ אחד בלבד מסוג exe, שקיבל את הציון 7. לעומתם, קבצים מסוג rtf ו – pdf, השיגו תוצאות פחות טובות, במיוחד קבצי ה – rtf.

אותם קבצים שקיבלו ציונים נמוכים יותר בולטים לרעה לעומת האחרים היות והצפיה היא שכולם יקבלו ציון גבוה. אולם, יש לזכור כי לא כל הקבצים נבדקו לאימות טיפוסם האמיתי וקיים חשש שקובץ מטיפוס אחר השתרבב לתוך המדגם. מצד שני, ככל שהמדגם גדול יותר, כך קטן משקלו היחסי של קובץ מסוג שונה.

דבר נוסף שיש לזכור הוא שבמבחן האמיתי, נבחן הקובץ הנבדק ביחס לטיפוסים השונים ולכן, גם אם לא יקבל ציון מלא, מה שיקבע את הטיפוס, על פי השיטה המוצעת, הוא ציונו ביחס לאחרים.



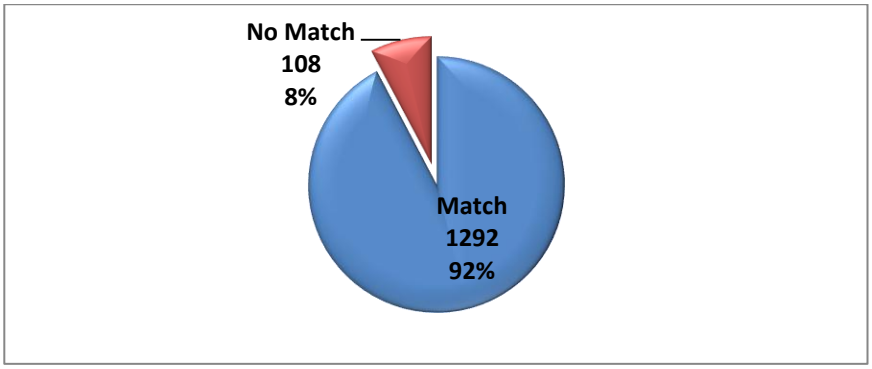
תרשים 34: תוצאות בדיקות פנימיות לטיפוסים עבור קבוצות חלקיות

4.6.5 בדיקת קבצי מדגם

החתימות למבחן הסופי בוצעו בעזרת הקבצים שנבחנו בסעיף הקודם. הכמויות לכל חתימה מפורטות בטבלה 2. בסך הכל נכללו 2520 קבצים ביצור החתימות. בקבוצת הקבצים למדגם נכללו כל אותם קבצים שלא היוו חלק מחתימה כלשהי. סך הכל 1400 קבצים נכללו במדגם זה.

התוצאות שהתקבלו מעודדות. מתוך 1400 הקבצים שנבחנו, אובחן המדויק הטיפוס של 1292 שהם כ – 92% מהקבצים. שאר 108 הקבצים המהווים כ- 8% מסך כל הקבצים לא אובחנו במדויק.

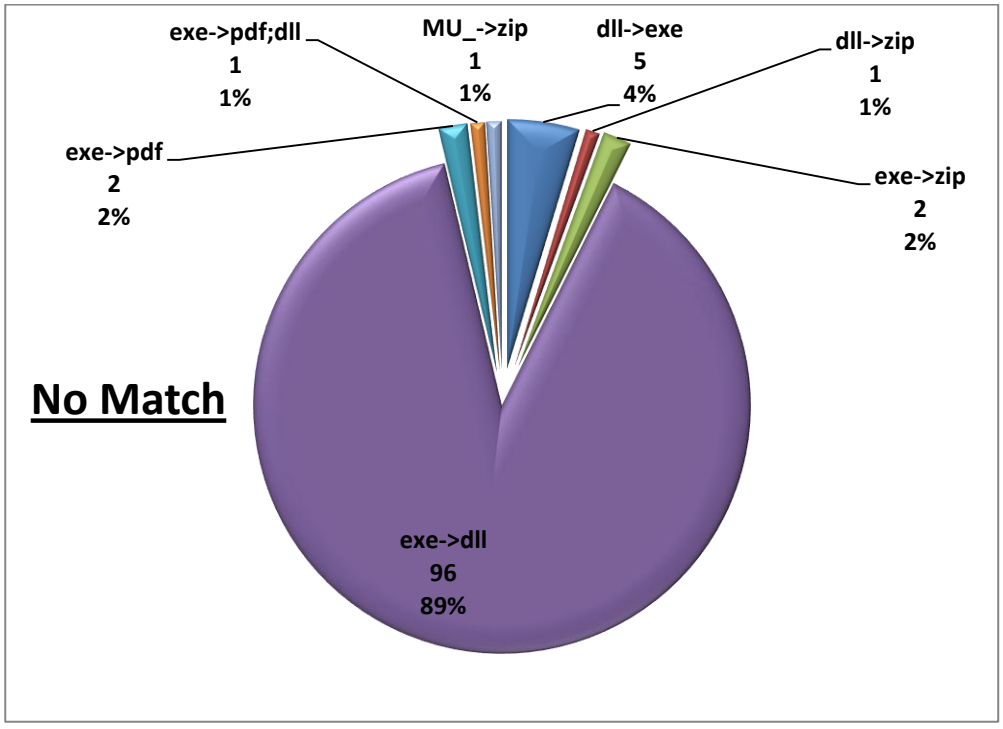
התוצאות שנתקבלו מומחשות בתרשים הבא.



תרשים 35 : תוצאות התאמת טיפוס עבור 1400 קבצים

4.6.6 ניתוח תוצאות

כאשר בוחנים את הקבצים, עבורם לא נמצאה התאמה, עולה תמונה מעניינת לגבי סוג הטעויות הנפוץ באנאליזה. התרשים הבא מעמיק באותם 8% מהקבצים שאינם תואמים.



תרשים 36: התפלגות קבצים עבורם לא התקבלה התאמה

מתוך 108 הקבצים, עבורם לא התקבלה התאמה, 97 קבצי exe זוהו כקבצי dll ו – 5 קבצי dll זוהו כקבצי exe. טעות זו סבירה היות וקבצי dll הם ספריות הרחבה המכילות קוד ביצועי, הדומה במהותו ומבנהו לקבצי exe, בשינויים קלים. זה מותיר אותנו עם 6 קבצים, לגביהם עדיין אין הסבר.

2 מקבצים אלו הם הקובץ eraserd.exe, שאובחן כקובץ מטיפוס zip. קובץ זה קיבל ציונים נמוכים עבור כל הטיפוסים, dll = 6, docx = 3, exe = 7, pdf = 3, rtf = 3, zip = 8. בדיקה של הקובץ מעלה שהוא מטיפוס ביצועי עבור מערכת ההפעלה DOS, עובדה המצריכה, כנראה, יצירת חתימה מיוחדת עבור קבצים מסוג זה. הקובץ השני הוא SCTASKS.EXE.MU_ שאובחן כקובץ מטיפוס zip, אולם, למעשה טיפוס זה הוא סוג של ארכיון לקבצי התקנה של Windows. גם הקובץ האחרון קיבל ציונים נמוכים במיוחד, מה שמרמז אולי על הצורך לאתר קבצים בעלי ציון נמוך בכל הטיפוסים ולהשתמש בהם כדי לאתר טיפוסים שלא אובחנו עד כה, במטרה לייצר עבורם חתימה. כוון נוסף שיש לבדוק עוסק במקרים של שילוב טיפוסים שונים באותו קובץ, לדוגמה השתלת קוד ביצועי בקובץ מסוג pdf, על מנת לבצע ניתוח מדויק על מרכיביו השונים של הקובץ.

סך הכל, אם בוחנים את הסטיה המתקבלת באבחון טיפוס הקבצים, מתקבל הרושם שהיא קטנה ביותר. אם נתעלם מהחלפה בין קבצי exe ו – dll ומהמקרים המוזכרים לעיל, נקבל 6 קבצים מתוך 1400 שלא אובחנו כראוי, טעות דגימה של כ – 0.4%. נתון זה מעודד במיוחד לאור העובדה שבהדגמה זו לא נעשה מאמץ מיוחד לבצע אופטימיזציה בפרמטרים של הניסוי, על מנת לקבל תוצאות מדויקות יותר.

מדד נוסף, בו נוכל להשתמש למדידת איכות התוצאות, הוא F-Measure. מדד זה מודד את דיוק התוצאות ולוקח בחשבון שני נתונים.

- דיוק התוצאות המוחזרות (Precision) - הנתון המתקבל מחלוקת מספר התוצאות המדויקות שהוחזרו בסך כל התוצאות המוחזרות.

$$Precision = \frac{True\ Positive}{(True\ Positive + False\ Positive)}$$

- מדד ההחזרה (Recall) - הנתון המתקבל מחלוקת מספר התוצאות המדויקות שהוחזרו במספר התוצאות שהיו אמורות לחזור.

$$Recall = \frac{True\ Positive}{(True\ Positive + False\ Negative)}$$

F-Measure מתקבל על ידי מעין ממוצע משוקלל בין השניים.

$$\mathrm{F} - Measure = \frac{2 * Precision * Recall}{(Precision + Recall)}$$

ערכי שלושת הנתונים נעים בין 0 ל – 1, כאשר 0 מייצג את התוצאה הגרועה ביותר ו – 1 את התוצאה הטובה ביותר. הטבלה הבאה מסכמת את התוצאות שהתקבלו עבור בדיקת קבצי המדגם.

F-Measure	Recall	Precision	FN	FP	TP	סה"כ קבצים	טיפוס הקובץ
1	1	1	0	0	182	182	DOCX
1	1	1	0	0	255	255	RTF
0.927	1	0.864	0	3	19	19	PDF
0.931	0.991	0.877	6	97	693	699	DLL
0.651	0.495	0.952	101	5	99	200	EXE
0.968	1	0.938	0	3	45	45	ZIP
0.923	0.924	0.923	107	108	1293	1400	סה"כ
0.997	0.993	1	6	0	894	899	קובץ ביצועי (EXE+DLL)
0.996	0.996	0.996	6	6	1394	1400	כל הקבצים

טבלה 4: F-Measure עבור תוצאות הניסוי

בטבלה, השורה האחרונה המסכמת את כל הקבצים מתייחסת לקבצי DLL ו – EXE כקבוצה אחת של קבצים ביצועיים. הערך 6 בשורה זו עבור FP מתקבל היות ובמקרה אחד מזוהה קובץ מטיפוס EXE כ – PDF או DLL. עבור קבצים מטיפוס DOCX ו– RTF התקבלו תוצאות מושלמות. תוצאות טובות מעל 0.92 התקבלו אף עבור שאר הטיפוסים למעט קבצים מטיפוס EXE, עבורם התקבלה תוצאה מאכזבת של 0.651. כזכור, מירב הטעויות עבור טיפוס זה ייחסו לו שייכות לקבוצת הקבצים מטיפוס DLL. אם נתייחס לעובדה שבשני המקרים מדובר בקבצים בעלי מבנה דומה של קוד ביצועי ונתייחס אליהם כקבוצה אחת, נקבל תוצאה מצויינת של 0.997. בסך הכל מתקבלת תוצאה כללית של 0.996 במדד ה F-Measure עבור כל הקבצים.

התוצאות המתקבלות בולטות לטובה גם בהשוואה למחקרים קודמים. הטבלה הבאה משווה בין תוצאות מספר מחקרים שסוקרו במסמך זה לעומת התוצאות שהתקבלו כאן, המצויינות בשורה האחרונה. יש לציין שבמסגרת המחקרים השונים נעשה שימוש במדגמים שונים, עובדה העשויה להשפיע על טיב תוצאותיהם.

מאמר	טיפוסים במדגם	מספר קבצים		הצלחה	הערות
		מדגם	בדיקה		
Content Based File Type Detection Algorithms [18]	30		120	27.50%	אלגוריתמBFA
				45.83%	אלגוריתם BFC
				95.83%	אלגוריתם FHT, מתבסס על ניתוח כותרת/סיומת תוכן הקובץ
Fileprints: Identifying File Types by n-gram Analysis [25]	6	640	160	80.4% - 100% (סריקה חלקית) 62.7% - 88.3% (סריקה מלאה)	טביעת קובץ עבור כל קבוצת קבצים שייצגו טיפוס
				90.0% - 100% (סריקה חלקית) 76.8% - 94.5% (סריקה מלאה)	בחירת תת הקבוצה הטובה ביותר עבור כל טיפוס
		6	794	86.9% - 100% (סריקה חלקית) 77.1% - 98.9% (סריקה מלאה)	בחירת קובץ מייצג עבור כל טיפוס
Using Artificial Neural Networks For Forensic File Type Identification [26]	5	13,000	1,300	1% - 50%	Raw Filtering
				0% - 60%	Character Code Frequency
Content based file type identification using cosine similarity and a Divide-and-Conquer Approach [23]	10	1,000	1,000	90.19%	Divide and Conquer + NN
מחקר זה	6	2,520	1,400	99.6%	לאחר איחוד קבצי DLL ו - EXE לטיפוס אחד

טבלה 5: השוואה בין תוצאות מחקר זה למחקרים קודמים

4.7 מסקנות

בהוכחת היכולת ראינו מימוש מעשי והדגמה של הארכיטקטורה לצורך אבחון של כ – 2500 קבצים בשלב אחד ו – 1400 קבצים בשלב ההוכחה. התוצאות שהתקבלו בבדיקה, הפשוטה יחסית, מעודדות ביותר ומראות שקיים בסיס להמשך המחקר בכוון שהותווה על ידי מאמר זה.

הארכיטקטורה המוצעת משמשת כבסיס למימוש מספר רב של ניסויים, תוך התבססות על הגמישות הרבה אותה היא מציעה. בין היתר ניתן לבחור את:

- גדלי N-Grams וגדלי חלון.
- סף הכניסה עבור N-Gram בקובץ מסויים.
- האנאליזה הסטטיסטית עבור כל N-Gram בקובץ מסויים.

- כל האנאליזות הסטטיסטיות השונות עבור כל N-Gram וקבוצת קבצים מאותו טיפוס.
- סף הכניסה עבור כל N-Gram וקבוצת קבצים מאותו טיפוס, לפי אחוז הקבצים או כמות ה – N-Grams שיכנסו לחתימה.
- העמודות השונות שישמשו לחתימה.
- המשקל שיוצמד לכל נתון בעזרת קובץ חוקי הציון.

אחד היתרונות העיקריים, אותם אני רואה בארכיטקטורה, הוא חוסר הצורך או המוטיבציה למצוא הגיון במבנה הנתונים של טיפוס זה או אחר. אם מרבית השיטות האחרות דורשות ידע במבנה הקובץ עבור טיפוס מסויים, לצורך ניתוחו ואפיון חתימה עבורו (למשל, הכרת מספר הקסם, magic number, עבור הטיפוס), הרי שבמקרה דנן, כל שנדרש הוא מדגם מייצג וגדול מספיק על מנת למצוא, באופן אוטומטי, תכונות חוזרות אותן ננצל לייצור קובץ חתימה מאפיין לטיפוס, שישמש להתאמת אותן התכונות בעת בדיקת הטיפוס של כל קובץ נבדק.

מסקנה מתבקשת נוספת היא שיעילות הארכיטקטורה אינה זהה עבור כל טיפוס וטיפוס. בטיפוסים מסויימים כקבצי docx מצאנו בנקל דפוסים מאפיינים ואכן, הדבר ניכר בתוצאות הפנימיות ותוצאות הוכחת היכולת, שם נתקבלו תוצאות מדויקות של 100% הצלחה. לעומתם ראינו כי בקבצים מסוג zip, בהם פיזור הבתים יותר אקראי, כביטוי של אופי הטיפוס, קשה יותר למצוא מספיק מאפיינים ייחודיים על מנת לייצר חתימה יעילה, דבר הבא לידי ביטוי בתוצאות הפחות מדוייקות אותם קיבלנו.

נתון מעניין שעלה תוך כדי הניסויים קשור לציונים שנתנו לקבצים מטיפוס docx. למרות שכל הקבצים מטיפוס זה אובחנו בהצלחה, היו קבצים רבים שלא קיבלו את הציון המלא. כאשר התמקדתי בקבצים שקיבלו ציון גבוה, התברר שרובם ככולם היו בשפה האנגלית. אלו שקיבלו ציונים נמוכים במיוחד היו ברובם בשפות זרות, רוסית, איטלקית או אחרת. הדבר מעלה את האפשרות שמוטיבים מסויימים של השפה, אנגלית במקרה זה, נכנסו כחלק מהחתימה ועזרו בזיהוי הטיפוס. סביר להניח שגם לחתימה עצמה השתרבבו מסמכים בשפות אחרות, אולם משקלם בחתימה היה שולי יחסית לאור השכיחות הגבוהה במיוחד של המסמכים באנגלית. גילוי זה פותח פתח מעניין למיקוד נוסף בקבצי טקסט, תוך בניית חתימות שיתבססו גם על שפת המסמך וישמשו לזיהוי השפה, בנוסף לזיהוי סוג המסמך.

זיהוי הטיפוס האמיתי של קובץ משמש, בדרך כלל, לצורך החלטה על המשך הטיפול בו. אם המטרה היתה נטולת התחשבות בשיקולים זדוניים, יכולנו להסתפק בסיומת הקובץ (במקרה של Windows) או במספרי הקסם שלו לצורך הזיהוי. אולם, החשש לזיוף סוג הקובץ, אם במטרה להבריחו לתוך הארגון כקוד זדוני או מחוץ לארגון כמידע סודי, מחייב בדיקה, עליה יהיה קשה להערים. מסיבה זו יש להבטיח שחתימה עבור טיפוס תהיה קשה מספיק לזיוף. עבור קבצי docx למשל, נוכל לקבל חתימה אמינה יחסית, לעומת קבצי zip, אותם ניתן לזייף בנקל. דרך נוספת להערים על המערכת היא למשל לבצע מניפולציה על תכולת הקובץ, לדוגמה, הצפנתו על פי מפתח או קידוד הבתים על פי פעולה מסויימת. דרך זו אפשרית אולם יש לזכור מספר דברים. לצורך שימוש מעשי בקובץ, יש להשתמש בתכנית שתבצע בו את פעולת ההסוואה, ותכנית נוספת שתבצע את פעולת הפיענוח. בכל כוון שלא נבחר, אם מהארגון החוצה או פנימה לתוך הארגון, אחת מפעולות אלו תדרש להתבצע בתוך הארגון. הדבר יחייב את הכנסת תכנית ההצפנה/פיענוח לתוך הארגון, תוך מעקף כל מנגנוני ההגנה שלו. דבר נוסף שיש לזכור הוא שעדיין קיימת בידינו האפשרות לייצר חתימה גם עבור הקובץ המוצפן או המומר, כך שאפשר יהיה לאתר אותו כקובץ מטיפוס מסויים שעבר המרה, עובדה שתגרור את העלאת סף החשדנות כלפיו.

המימוש לצורך ההדגמה אינו יעיל, לפיכך אני רואה בארכיטקטורה המוצעת פתרון לביצוע נסיונות שונים עד לקבלת פרמטרים אופטימליים, שיספקו תוצאות מספיק מדויקות. פרמטרים אלו יכולים לשמש בסיס לאחר מכן למימוש פתרון מעשי ויעיל המבוסס עליהם.

4.8 כוונים עתידיים

הגמישות אותה מציעה הארכיטקטורה בונה בסיס רחב לביצוע נסיונות שונים בעתיד. להערכתי, הכוון של הגדלת ה – N-Gram לא יניב תוצאות טובות יותר היות והגדלת ה – N-Gram עשויה להקטין את כמות ה – N-Grams החוזרים בקבצים השונים של הטיפוס ובכך להחליש את החתימה שלו. לעומת זאת, הגדלת החלון עשויה להיות כוון מעניין שעשוי להרחיב את כמות ה – N-Grams הזמינים. צעד נוסף בכוון זה, שעשוי להיות מעניין, הוא לסמן עבור כל N-Gram בחלון המורחב את המרחק בין רכיביו ולהשתמש במידע זה בזמן האנאליזה כאינדיקטור נוסף במתן הציון. במרחק בין רכיביו כוונתי לסכום הפערים בין הבתים השונים. לדוגמה עבור N-Gram בגודל 3 וחלון בגודל 5, עבור רצף הבתים 04 03 02 01 00 וה – N-Gram 04<>02<>00 נקבל מרחק 2 עבור הפערים שתורמים הבתים 01 ו 03. אולם, התפתחות בכוון זה אינה באה ללא תג מחיר שבמקרה זה עשוי להיות בתשלום כבד על ביצועים ובמיוחד, צריכת זיכרון גבוהה.

לאור הפתיחות של חבילת ה – NSP לקבלת אנאליזות סטטיסטיות נוספות, ניתן לשקול להרחיב את ההיצע שלה ולהעשיר את האפשרויות העומדות בפני מבצעי הנסיונות. הרחבה נוספת לארכיטקטורה עשויה להיות על ידי שינוי צורת מתן הציון ל – N-Gram. בארכיטקטורה הקיימת, הציון נקבע לפי אך ורק לפי הטיפוס הנבדק. ניתן להוסיף אפשרויות ציון נוספות כגון חוק מיוחד עבור N-Gram מסוים של טיפוס מסוים. למשל, הדרישה שכדי שהקובץ יזוהה כטיפוס מסוים, המצאות N-Gram מסוים היא חיונית, אחרת הקובץ אינו מטיפוס זה למרות שהוא עשוי להשיג ציון גבוה. חוק זה כמובן שונה מהמקרה של N-Gram שמופיע ב 100% מהקבצים אולם תורם רק את חלקו היחסי.

כמובן שאחד הכוונים החשובים ביותר, אותם יש לפתח על מנת להפוך את השימוש בארכיטקטורה לישים הוא לכתוב מימוש יעיל, לפחות עבור תהליך האיבחון. מעבר לדיוק האיבחון, אפשרות שימוש מעשית בארכיטקטורה תקום או תיפול על בסיס מהירות הביצוע ועלות משאבי ריצה במחשב.

כוונים אפשריים נוספים:

- שימוש בארכיטקטורה לצורך איתור אנומאליות בקבצים. לדוגמה, מציאת קבצים שאינם מנפקים ציון גבוה עבור שום טיפוס מוכר, למרות שעשויים להיות מזוהים כטיפוס מסויים לאור הציון היותר הגבוה אותו קיבלו עבור טיפוס זה. קבצים אלו עשויים להפוך למועמדים לבדיקה מעמיקה יותר שבעקבותיה יאובחנו כקבצים לגיטימיים אשר סוגם טרם אובחן ותוייג, או כקבצים עויינים.
- שימוש בארכיטקטורה לצורך Text Classification. מיון מסמכים באופן אוטומטי לקטגוריות שונות. לתחום זה יישומים שונים כגון מיון אוטומטי של מסמכים מדעיים, קטלוג אוטומטי של מקורות Web ועוד.
- שימוש בארכיטקטורה ככלי עזר במחקרים בתחום Bioinformatic, ביולוגיה מולקולארית ו - DNA.
- לאור התוצאות שהתקבלו עבור קבצים מטיפוס docx, ניתן להרחיב בכוון של איתור השפה בה נכתב המסמך. בנוסף, במקרה של מימוש של איבחון יעיל ניתן אולי לעשות שימוש לצורך איתור טקסט מסויים זה או אחר, למניעת דליפת מידע. תפקיד אותו ממלאות היום מערכות DLP (Data Loss Prevention).
- בדיקת אפשרות יצירת חתימות אוטומטיות עבור קבצים בהם נעשתה המרה לצורך הערמה על מערכות ההגנה.
- שימוש בארכיטקטורה לצורך זיהוי מרכיבים מטיפוסים שונים בתוך קובץ אחד, לדוגמה קבצי docx המכילים בתוכם קוד תמונה או קוד ביצועי.

רשימת מקורות

[1] M. E. Kabay and E. Whyne, "A Brief History of Computer Crime," in *Computer Security Handbook, 5th Edition, Volume I*. Wiley, 2008, ch. 2.

[2] C. Fleizach, M. Liljenstam, P. Johansson, G. M. Voelker, and A. Méhes, "Can You Infect Me Now? Malware Propagation in Mobile Phone Networks," in *ACM Workshop on Recurring Malcode*, 2007.

[3] Websense, "State of Internet Security," Websense Security Labs, Q1 - Q2 Quarterly Report, 2008.

[4] Sophos, "Security threat report," Sophos Plc., Security threat report, 2009.

[5] P. A. Taylor, *Hackers: Crime in the Digital Sublime*, 1st ed. Routledge, 1999.

[6] D. E. Denning, *Information warfare and security*, 1st ed. Addison-Wesley Professional, 1998.

[7] C. Macleod, "Is that a hacker next to you?," *Communications Engineer*, vol. 5, 2007.

[8] N. Chantler, "Risk: The Profile of the Computer Hacker. ," Ph.D. dissertation, Curtin University of Technology, Australia, 1995.

[9] N. Idika and A. P. Mathur, "A Survey of Malware Detection Techniques," Department of Computer Science, Purdue University, Research Report, 2007.

[10] McAfee, "A Brief History of Malware," McAfee, Inc White Paper, October 2005.

[11] P. Mell, K. Kent, and J. Nusbaum, "Guide to Malware Incident Prevention and Handling," Computer Security Division , National Institute of Standards and Technology NIST Special Publication, 2005.

[12] Microsoft, "Microsoft Security Intelligence Report," Microsoft Corporation, Key Findings Summary report, June, 2008.

[13] J. Aycock, *Computer Viruses and Malware*. Springer Science Business Media, 2006.

[14] P. Szor, *The Art of Computer Virus Research and Defense*. Addison Wesley Professional, 2005.

[15] E. Filiol, "Strong Cryptography Armoured Computer Viruses Forbidding Code Analysis: the Bradley Virus," in *Best Paper Proceedings*. EICAR, 2005.

[16] A. E. Stepan, "Defeating Polymorphism," in *Virus Bulletin Conference*, 2005.

[17] D. Bilar, "Opcodes as predictor for malware," *International Journal of Electronic Security and Digital Forensics*, vol. 1, 2007.

[18] M. McDaniel and M. H. Heydari, "Content Based File Type Detection Algorithms," in *Proceedings for the 36th Hawaii International Conference on System Sciences*, 2002.

[19] M. Christodorescu and S. Jha, "Static Analysis of Executables to Detect Malicious Patterns," Proceedings of the 12th USENIX Security Symposium (Security'03), USENIX Association, 2003.

[20] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*. Addison Wesley, 2001.

[21] J. Z. Kolter and M. A. Maloof, "Learning to Detect Malicious Executables in the Wild," in *Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004.

[22] K. S. Dash, S. R. K. Dubba, and K. A. Pujari, "New Malicious Code Detection Using Variable Length n-grams," in *Algorithms, Architectures and Information Systems Security*. World Scientific, 2008, ch. 14, pp. 307-323.

[23] A. Irfan, L. Kyung, S. Hyunjung, and H. ManPyo, "Content-Based File-type Identification Using Cosine Similarity and a Divide-and-Conquer Approach," *IETE Technical Review*, vol. 27, no. 4, Jul. 2010.

[24] M. Karresand and N. Shahmehri, "File Type Identification of Data Fragments by Their Binary Structure," in *Proceedings of the 2006 IEEE Workshop on Information Assurance United States Military Academy*, West Point, NY, 2006.

[25] W.-J. Li, S. J. Stolfo, and B. Herzog, "Fileprints: Identifying File Types by n-gram Analysis," in *2005 IEEE Workshop on Information Assurance*, West Point, NY, 2005.

[26] R. M. Harris, "USING ARTIFICIAL NEURAL NETWORKS FOR FORENSIC FILE TYPE IDENTIFICATION," MSc. Thesis, Purdue University, West Lafayette, Indiana, 2007.

[27] R. F. Erbacher and J. Mulholland, "Identification and Localization of DataTypes within Large-Scale File Systems," in *Proceedings for the Second International Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE'07)*, 2007, pp. 55-70.

[28] R. Moskovitch, et al., "Unknown malcode detection and the imbalance problem," *Journal in Computer Virology*, vol. 5, no. 4, pp. 295-308, 2009.

[29] T. Pedersen, S. Banerjee, A. Purandare, B. T. McInnes, and Y. Liu. (2009) NSP - Ngram Statistics Package.

Abstract

In today's connected environment, most businesses, large and small, increasingly rely on the Internet as a source of information and a platform for communication and Electronic commerce.

At the same time that the Internet infrastructure becomes an important element in increasing the productivity for all modern businesses, it exposes them to a new type of threat, one that was unknown before the Internet era. The threat is on the infrastructures of the organization using digital communication channels as a mediator.

One of the main motivating factors driving the increased use of the net is the ability to use technologies based on active content (Such as Active-x, Java applets, Java Script and Executable files) in order to implement Web Based Applications. The flexibility of these technologies conveys great benefits, but at the same time, they serve as Malware carriers, capable of harming the organization by damaging its infrastructures, stealing information or performing other illegal activities.

Traditional protection methods like Anti viruses, Intrusion prevention systems and URL Categorization, which are widely used today, do not provide sufficient protection against modern threats, threats which are already prevalent today. Mostly, they are based on Reactive protection, meaning, from the time that the threat is known (for instance, a new type of Virus), the software vendors creates some kind of a signature and updates its customer's machines. The problem with this approach is that it leaves the business vulnerable from the time when the Virus started to spread, up to the time that the protection software gets the update from the vendor. These protection methods do not supply sufficient protection for unknown or complex attacks.

An approach of blocking all access to active content might work well in protecting, but most of the infrastructures, such as CRM, ERP and Electronic commerce systems are based on active content, so this approach will not do either.

For these reasons and others, the modern protection approach is Proactive, meaning the infrastructure will allow the use of active content, but not before inspecting its behavior in an isolated place, where it can do no harm. This inspection might be achieved by using semantic analysis of the code or by allowing the code to run in some kind of "Sand Box", while inspecting its behavior.

This approach is not perfect and there are ways to work around it. For instance, a code might randomly activate its malicious part, in the hope that it will not be active while it is under inspection. Another approach is designing the code in such a way that it will act normal, unless it is in the presence of a specific file.

In the last mentioned attack approach, there is an attack method, which is constructed from several stages and components. The combination of all these activates the attack. For instance, one file can contain a type of "Command Interpreter", which does not perform any suspicious behavior, unless it is in the presence of another file, which contains a set of related commands.

Most of the time, the most difficult part of the attack is to penetrate and infuse the code into the system. Attackers develop new approaches to disguise the true nature of the file content, if by using naive methods such as changing File Extensions or injecting Malware code into an otherwise, harmless and legitimate code such as a document, an image or a music file.

Recent research in this area was undertaken in order to find efficient ways to identify the true nature of code in a file, without relying on external characteristics. In this thesis I am reviewing the progress in this area, while analyzing some of the methods and directions taken. Also, I am looking at new approaches, which are based on Byte Distribution Analysis.

The main contribution of this thesis is by presenting a Framework that supply, in a comfortable way, methods for finding the most efficient parameters such as N-Gram size, Statistical classifier and other qualifiers, for the creation of characteristic signature of different file types. The Framework provides an infrastructure for automatic creation of a signature for every file type, based on sampling data constructed from files of that type. Moreover, the Framework can scan unknown files and determine their type based on that signature.

TABLE OF CONTENTS

The Open University of Israel
Department of Mathematics and Computer Science

Code Nature Revealing
Using Static Analysis

Thesis submitted as partial fulfillment of the requirements
Towards an M.Sc. degree in Computer Science
The Open University of Israel
Computer Science Division

By
Raam Sharon

Prepared under the supervision of Prof. Ehud Gudes

August 2011

VII