**טבת תשע"ז** **ינואר 2017**



***עזריאלי***

***www.JCE.ac.il***

המחלקה להנדסת תוכנה

**אלגוריתם להתאמת מחרוזות על תעבורת רשת מכווצת**

**חיבור זה מהווה חלק מהדרישות לקבלת**

**תואר שני בהנדסה**

**מאת**

**עילי זידמן**

**300634524**

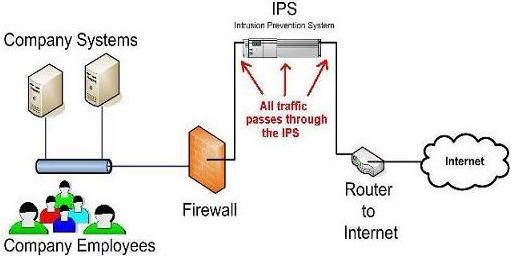
**izeidman@cisco.com**

**מנחה**: **ד"ר צור-דוד שמרית**

# תקציר הפרויקט

מערכות איתור חדירות (IDS) פותחו כדי לזהות ולדווח תקיפות ברשת בסוף 1990, כאשר תקיפות האקרים ותולעים ברשת החלו להשפיע על האינטרנט. טכנולוגיות IDS מסורתיות יודעות לזהות תנועה עוינת ולשלוח התראות אבל לא יודעות לעשות דבר כדי לעצור את ההתקפות.

מערכות מניעת חדירות רשת (NIPS) מותקנות אונליין עם קטע הרשת המוגן. כאשר התנועה עוברת דרך המערכת, נבדקת האם יש התקפה. כמו וירוסים, לרוב החדירות יש איזשהו סוג של חתימה. לכן, אלגוריתם-התאמת מחרוזות הוא החלק העיקרי במערכת מניעת חדירות רשת. כאשר התקפה מזוהית, NIPS חוסמת את התקיפה. ישנו trade-off בין הדיוק של הזיהוי ויעילות האלגוריתם. שניהם הם מאוד חשובים כדי להבטיח שתנועה לגיטימית לא מתעכבת או משובשת כאשר היא זורמת ברשת. מסיבה זו, אלגוריתם התאמת המחרוזות חייב להיות מסוגל לפעול במהירות גבוהה, תוך גילוי עיקר החדירות. עם מהירות הרשת המכפילה את עצמה מדי שנה, זה הופך קשה יותר ויותר עבור פתרונות מבוססי תוכנה להיות כאלה מהירים.



כיום כלי אבטחה לא יודעים להתמודד עם תנועה מכווצת, אשר הופכת להיות יותר ויותר נפוצה ב HTTP. פרוטוקול HTTP משתמש בכיווץ GZIP, אשר בהתחלה דורש סוג כלשהו של decompression לפני ביצוע משימת התאמה המחרוזות. לפיכך, זה מאוד לא יעיל להריץ אלגוריתם התאמת מחרוזות על מידע מכווץ.

בפרויקט זה אציג אלגוריתם התאמת מחרוזות על HTTP מכווץ(GZIP). האלגוריתם משתמש ב Ternary Content Addressable Memory (TCAM) והוא מסוגל להתאים מחרוזות רבות בפעולה אחת. האלגוריתם משיג מהירות קו בשיעור של כמה סדרי גודל יותר מהר מאשר עבודות נוכחיות, תוך השגת דיוק דומה של זיהוי. הפרויקט הוא הרחבה של פרויקט שעשתה המרצה שלי בשיתוף עם עוד סטודנטים מהאוניברסיטה העברית שבו האלגוריתם עובד על HTTP לא מכווץ.

האלגוריתם משתמש בעובדה שהחבילות מכווצות ובצורת המימוש של GZIP, ושומר נתונים תוך כדי ריצת האלגוריתם כדי שהוא לא יצטרך לבצע decompression של החבילה.

תוצאות המדידות מראות שהאלגוריתם משפר את הביצועים ביחס לאלגוריתם המקורי גם בכמות הבייטים שהאלגוריתם מדלג וגם בכמות הבדיקות ב TCAM. בנוסף כדי שהאלגוריתם המקורי יעבוד על חבילות מכווצות, הוא צריך לבצע decompression שזו פעולה מאוד יקרה. האלגוריתם שמומש לא מבצע decompression של המידע.

תוכן עניינים

[תקציר הפרויקט 2](#_Toc478538776)

[הצהרה 5](#_Toc478538777)

[תודות 6](#_Toc478538778)

[רקע 7](#_Toc478538779)

[תאור הבעיה 8](#_Toc478538780)

[אלגוריתם GZIP 8](#_Toc478538781)

[תאור הפתרון 10](#_Toc478538782)

[אלגוריתם 10](#_Toc478538783)

[אתחול ה TCAM 10](#_Toc478538784)

[ריצת האלגוריתם 10](#_Toc478538785)

[שינויים כדי לטפל במצביעים 12](#_Toc478538786)

[גבול שמאלי 12](#_Toc478538787)

[גבול אמצעי 13](#_Toc478538788)

[גבול ימני 14](#_Toc478538789)

[מימוש 15](#_Toc478538790)

[תוצאות שהתקבלו 17](#_Toc478538791)

[השפעת רוחב ה TCAM על ממוצע הדילוגים 17](#_Toc478538792)

[השפעת גודל החבילה על ממוצע החיפושים ב TCAM 18](#_Toc478538793)

[פתרונות דומים בספרות 19](#_Toc478538794)

[רשימת ספרות 20](#_Toc478538795)

# הצהרה

העבודה נעשתה בהנחיית ד"ר צור-דוד שמרית,

במכללה האקדמית להנדסה ירושלים – עזריאלי

המחלקה להנדסת תוכנה.

החיבור מציג את עבודתי האישית

ומהווה חלק מהדרישות לקבלת תואר שני

M.Sc. בהנדסת תוכנה.

# תודות

תודה רבה למנחה הפרויקט ד"ר צור-דוד שמרית על העזרה וההדרכה בפרויקט.

תודה למכללה להנדסה ולכל צוות המכללה.

תודה לאשתי היקרה על התמיכה והעזרה בכל, הכל בזכותך.

# רקע

פרויקט זה בא ליישם חלק ממחקר בתחום אבטחת המידע של סגל של האוניברסיטה העברית בירושלים Yaron Weinberg, Shimrit Tzur-David, Danny Dolev ו- Tal Anker שהבינו כי הטכנולוגיה והסכנות ברשת הולכות וגוברות בימינו ולכן פיתחו אלגוריתם חדש עבור NIPS שיאפשר רמת אבטחה גבוהה תוך שימור הדיוק ובמהירות ביצוע גבוהה. האלגוריתם לחיפוש חתימות שהציגו במאמר: High Performance String Matching Algorithm for a NIPS[[1]](#footnote-1) נקרא RTCAM. האלגוריתם מציע שימוש ברכיב חומרה הנקרא TCAM.

היתרון הראשון של הרכיב הוא מהירות הביצוע, מכיוון שזיכרון זה מסוג TCAM , זה מאפשר גישה בO(1) לכל מקום בזיכרון.

היתרון השני הוא יכולת הרכיב לשמור שלושה ערכים בתוכו (1,0, don’t care) יתרון זה יהיה הבסיס של הלוגיקה באלגוריתם RTCAM . שני היתרונות אלה יהיו גורמים משמעותיים במהירות הסריקה. לאחר ניסויים רבים, הגיעו לתוצאות כי שימוש באלגוריתם RTCAM ב NIPS מאפשר מהירות בדיקה מאוד גבוהה. עוד יתרון גדול שהמחקר הקפיד עליו, הוא התאמה בין האלגוריתם RTCAM למערכת SNORT שהפך להיות גורם מרכזי במערכות הגנה כמו IDS או NIPS . פרויקט גמר הזה תופס חלק מעשי בהתקדמותו של המחקר מפני שבא לממש את האלגוריתם RTCAM על מנת שהוא יתאים לסריקה של חבילות מכווצות ברשת.

# תאור הבעיה

כיום ישנה עלייה בשימוש בכיווץ בפרוטוקול HTTP. עובדה זו מחייבת פיתוח פתרון לבעיות אבטחה שיכולות להיות במערכות תקשורת, אשר צריכות לבדוק חתימות בתוכן חבילות מכווצות על מנת להחליט האם לתת לחבילה לעבור או להגן על מערכת ולזרוק אותה. תהליך זה, של בדיקות תוכן החבילה המגיעה למערכת תקשורת, נקרא Deep Packet Inspection. בנוסף לזיהוי חתימות של וירוסים, ספאמים, או חדירות כלשהן, DPI מאפשר גם צנזורה על הפצת מידע על מנת להגן על קניין רוחני.

כאשר החבילה מגיעה למערכת תקשורת כגון נתב, הוא יצטרך לפענח אותה על מנת לסרוק את התוכן. תהליך זה כבד ודורש 32 KB עבור כל חיבור מה שגורם למערכת לשרוף כמות משמעותית של מאות מגבייט או גיגבייט של זיכרון כאשר עליה לבדוק מאות חיבורים. במקרים אחרים, משתמשים בתוכנות אבטחה, שנמצאות בין הלקוח לבין השרת, שלא תומכות בכיווץ. במקרים אלה התוכנות אחראיות לכתוב מחדש את הפתיח של החבילה ולהתריע בפני השרת שהדפדפן לא תומך בכיווץ מידע. תהליך זה מקשה על תעבורה מהירה כי תופס הרבה רוחב הפס.

חסרונות אלה הינם משמעותיים ולכן מערכות רבות מתעכבות בגלל בדיקות של חבילות מכווצות ומחליטות להגביל את אחוז החבילות המכווצות או פשוט מזניחות את הבדיקות של חבילות אלה. תוקפים הבינו, שבאותו אופן ההתקפות שלהם יכולות להסתובב ברשת כקבצים מכווצים וכי מערכות מסוג NIPS עכשוויות לא תומכות לגמרי בטיפול של קבצים מסוג זה.

## אלגוריתם GZIP

פרוטוקול HTTP מאפשר כיווץ של הנתונים והאלגוריתם הנפוץ ביותר הוא GZIP. GZIP היא תוכנה בקוד פתוח שפותחה על ידי Jean-Loup Gailly ו- Mark Adler בשנת 1992 המתבסס על האלגוריתם הנקרא DEFLATE שמכווץ את הקובץ בעזרת שילוב של שתי שיטות כיווץ.

תחילה הקלט מכווץ על ידי האלגוריתם **LZ77** ובשלב שני הפלט מכווץ על ידי **קוד הופמן**. הרעיון של האלגוריתם LZ77 הוא לאתר שכפולים אפשריים במחרוזת ולהחליף אותם על ידי מצביעים כזוג סדור המציין ראשית את המרחק בבייתים מהמצביע הקורא אל הבייט הראשון של המחרוזת החוזרת, ושנית את גודלו של השכפול.

השיטה הבאה שמופעלת בתהליך הכיווץ GZIP היא שימוש בקוד הפמן. השיטה מתבססת על אורך משתנה לסימנים על פי שכיחותם, כך שסימן נפוץ יוצג באמצעות מספר קטן של סיביות. לכן כאשר השיטה מקבלת כקלט את הפלט של האלגוריתם LZ77 , הוא מכיל מלא סימנים "]", "," ו-"[" בתדירות מאוד גבוהה ולכן בעזרת קוד הפמן, הם יוצגו עם כמה שפחות ביטים. השילוב בין שני האלגוריתם ב- GZIP מאפשר הורדה במשקלו של קובץ בזכות המצביעים וייצוג במספר מינימלי של סיביות, לרוב ניתן לחסוך באמצעות שיטה זו בין 20 ל 90 אחוז משטח האחסון, עבור סימנים בשכיחות גבוהה.

# תאור הפתרון

הפתרון לבעיה זו, הוא לפתח אלגוריתם שידע לזהות ולטפל בחוקי הכיווץ כגון שימוש במצביעים על מנת לא לבצע decompression של המידע ולדלג על כמה שיותר בייטים, אשר בהם האלגוריתם ידע בוודאות שאין חתימה של וירוסים.

כלומר שה- NIPS יהיה מסוגל לסרוק קובץ מכווץ כמו שהוא כאשר הוא מגיע אליו מבלי להרחיב אותו. מיתרון זה, נוכל להרוויח זמן בשלב הזיהוי של הווירוס ולכן בזמן התגובה של המערכת כאשר היא תחליט לפעול. על מנת לשמור על איכות הזיהוי של חתימות הווירוסים, האלגוריתם יצטרך להיות מסוגל להשתלב עם המערכת SNORT. לב פתרון הבעיה הוא שימוש בזיכרון חומרתי הנקרא TCAM . רכיב זיכרון זה הוא מסוג Content-addressable וזמן חיפוש בתוכו הוא O(1) .

## אלגוריתם

### אתחול ה TCAM

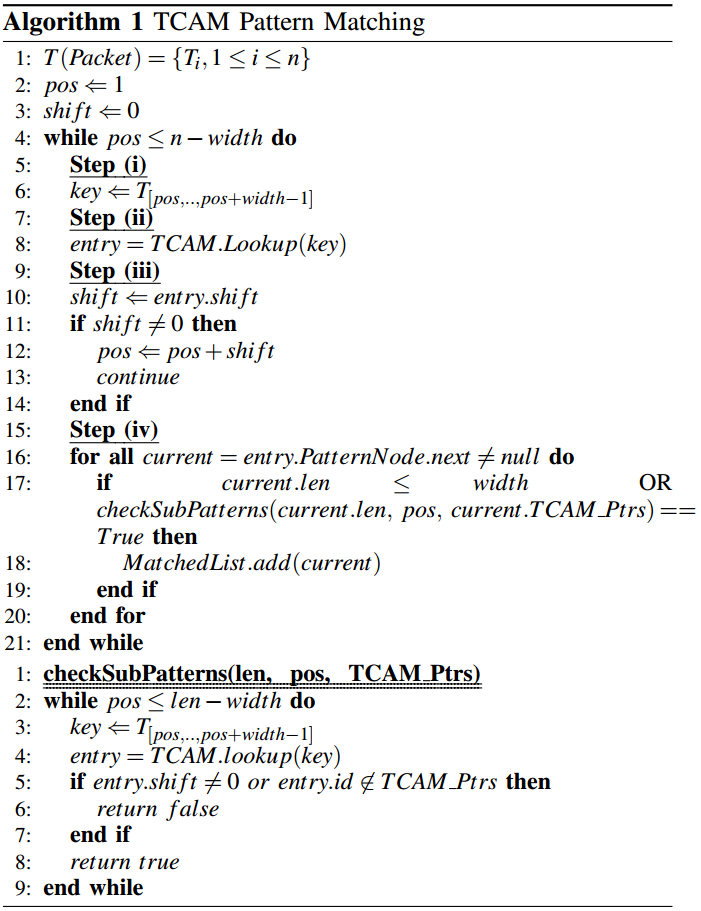
TCAM בגודל M יכול להחזיק בתוכו M/w שורות, כאשר w הוא רוחב ה TCAM. ה TCAM מאותחל בתהליך של שני שלבים. בשלב הראשון שקורה מראש אנחנו מחלקים את החתימות כך שיתאימו לרוחב ה TCAM, w. חתימות אשר יותר גדולות מ w יפוצלו ליותר משורה כאשר על כל שורה שומרים metadata מהו גודל החתימה, מה האינדקס בתוך החתימה וכו'... לדוגמא אם נרצה להכניס ל TCAM ברוחב 3 את החתימה abcdef יכנסו שתי שורות: abc, index=0, length=6, shift=0,sigNum=1

ו- def, index=1,length=6, shift=0,sigNum=1

לכל שורה יש גם shift שאומר כמה בייטים אנחנו יכולים לדלג כאשר יש התאמה בחיפוש ב TCAM. בשלב הראשון כל ה shifts הם 0, בשלב השני אנחנו יוצרים מכל שורה, שורה חדשה עם הזזה של בייט אחד לצד ימין כאשר מאבדים את התו הכי ימני של השורה המקורית, הזזה כזאת מעלה את ערך ה shift ב 1 והתו השמאלי ביותר יהיה don’t care. בדוגמא שמעל זה יהיה: ?ab, index=0, length=6, shift=**1**,sigNum=1 ו ?de, index=1,length=6, shift=**1**,sigNum=1 וכן הלאה עד שהשורה האחרונה תהיה כולה don’t care והיא בעצם השורה שמוחזרת אם אין התאמה לאף שורה. הסדר של ההכנסה ל TCAM הוא ממוין לפי ערך ה shift בגלל התכונה של ה TCAM שמחזיר את השורה הראשונה שמתאימה.

### ריצת האלגוריתם

להלן פסאודו קוד של האלגוריתם:



האלגוריתם פועל בארבעה שלבים:

בשלב 1 בונים מהחבילה את ה key בגודל w שאותו אנחנו הולכים לחפש ב TCAM.

בשלב 2 אנחנו מחפשים ב TCAM את השורה שמתאימה ל key שבנינו קודם.

בשלב 3 בודקים מה ה shift שהחזיר ה TCAM, אם ה shift גדול מ 0, זאת אומרת שה key לא התאים לאף חתימה ואפשר לדלג shift בייטים ולחזור ל שלב 1.

אם ה shift שווה 0 זאת אומרת שיש התאמה לתחילית של חתימה ואז עוברים שלב 4.

בשלב 4 אני בודק לפי metadata אם באמת יש התאמה של חתימה ואם כן מוסיף את המיקום לרשימת החתימות שהאלגוריתם מצא.

### שינויים כדי לטפל במצביעים

כדי לטפל במצביעים הייתי צריך לשנות את האלגוריתם המקורי שהסברתי לעיל והוספתי שתי רשימות:

1. SPMB- שומר ערך בוליאני עבור כל ביט, אם באינדקס מסוים הערך הבוליאני הוא true זה מסמן לנו שתת חתימה נמצאה באינדקס הזה והיא מתחילה באינקס פחות רוחב ה TCAM.
2. PMB- שומר ערך בוליאני עבור כל ביט, אם באינדקס מסוים הערך הבוליאני הוא true זה מסמן לנו שחתימה מלאה נמצאה באינדקס הזה.

### גבול שמאלי

מקרה ראשון- הביט דולק ב SPMB:

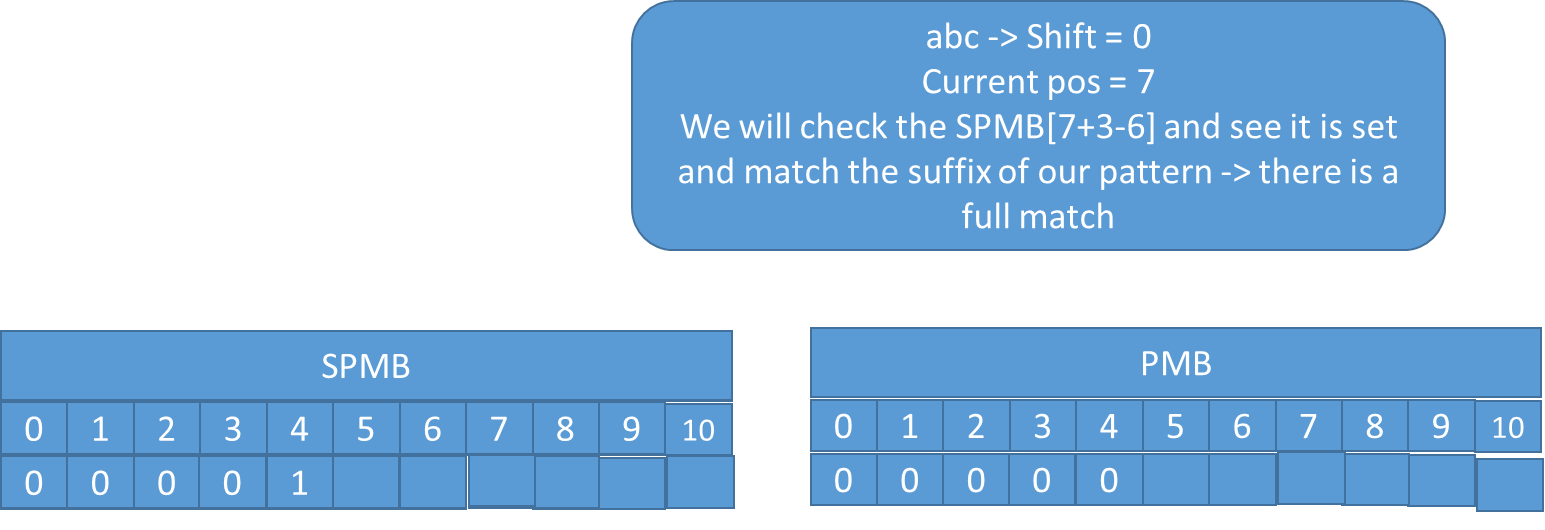
דוגמא:

Pattern set:

{“abcdef”}

W = 3

Input: bcdefa{6,5}



מקרה שני- הביט לא דולק בגבול: סורקים עד w-1.

### גבול אמצעי

נבדוק את הערכים ב PMB אם אחד מהם דולק נבדוק ב SPMB במקומות i-w, i-2w... נבדוק כך עד אורך החתימה ואם כולם דולקים מצאנו חתימה בגבול אמצעי.

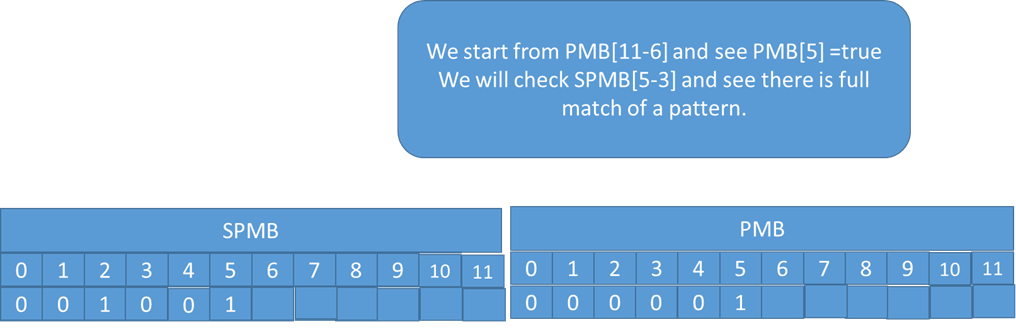
דוגמא:

Pattern set:

{“abcdef”}

W = 3

Input: abcdef{6,6}



### גבול ימני

בדומה לגבול השמאלי נתחיל מ w-1 עד סוף הגבול ואם יש ביט דולק נלך אחורה כדי לראות אם יש התאמה.

דוגמת הרצה עם מקסימום shift:

W = 5

Full Packet: pppppeeeeerrrrrttttt

Compressed: ppp[2,3]eee[2,2]rrrrrttt[2,2]

############# TCAM entries for ‘abcde’ pattern in HEX #############

6162636465

?61626364

??616263

???6162

????61

?????

####################### Algorithm lookups ( <lookup key> <shift> ) #########

Search 1: ppppp 5

Search 2: eeeee 5

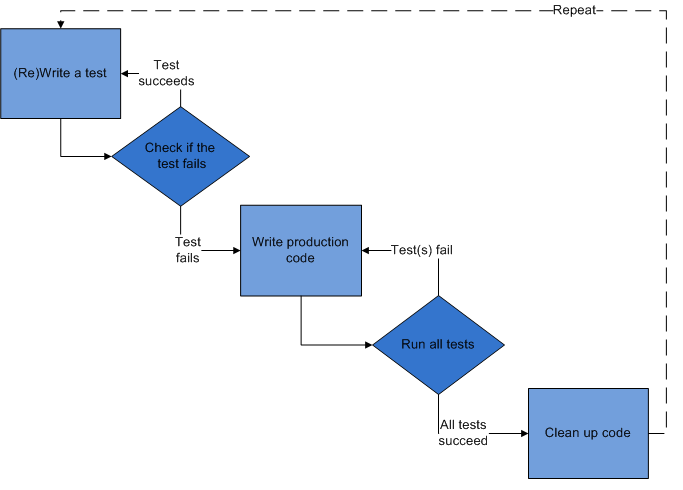
Search 3: rrrrr 5

Search 4: ttttt 5

בדוגמא זאת ניתן לראות שהחיפוש הראשון החזיר shift 5 לכן דילגנו על 5 אותיות של p אותו דבר קרה בחיפושים הבאים וזה מראה מקרה אופטימלי שגודל החבילה היה 20 ותוך 4 חיפושים ב TCAM סיימנו לסרוק את החבילה.

## מימוש

מימשתי את הפרויקט בשפת Scala במתודולוגיית פיתוח מונחה-בדיקות (test driven development ובקיצור TDD) השתמשתי ב Scala-test עבור כתיבת והרצת בדיקות היחידה. כך נראה תהליך של כתיבת קוד ב TDD:



היתה לשימוש בשיטה זאת יתרון משמעותי בכך שכמה פעמים במהלך הפרויקט אחרי שהמנחה עשתה לי code review הייתי צריך לעשות שינוי רוחבי בקוד, ואז הטסטים גילו לי רגרסיה בכל מיני מקרי קצה. דוגמא אחת כזאת היתה כשלא פירססתי נכון את החתימות של Snort, חתימה של Snort יכולה להראות כך לדוגמא:

**content:"|0A|Array|0A 28 0A 20 20 20 20 5B|"** אני בטעות התייחסתי לזה כ string למרות שבאמת מה שבין ה || צריך להיות אקסה, לכן החלטנו עם שימרית שנמיר את זה במקום ל string לאקסה וגם את החבילות נמיר לאקסה ונעשה השוואה באקסה. שינוי זה גרר כמה שינויים בקוד, ואחרי השינויים כשהייתי בטוח שהכל עובד מצוין הרצתי את הטסטים ושישה מהם נכשלו! דיבגתי את הטסטים וגיליתי שהיו כמה regressions בכמה מקרים, תיקנתי את הקוד ואז כל הטסטים עברו והייתי בטוח שכל המקרים שאני אמור לתמוך בהם עדיין נתמכים.

הקוד משותף ב GitHub בקישור: <https://github.com/ilayze/RTCAM-Compressed-HTTP>

המחלקות שמימשתי הם:

* **tcamSimulator.scala –** סימולטור המדמה את רכיב ה TCAM. המחלקה מכילה שיטות לאתחול הרכיב, ולקבלת מטהדטה עבור מפתח מסוים.
* **gzipPacket.scala –** המחלקה מייצגת חבילה בודדת ומכילה פונקציה לאתחול החבילה ולקבלת חלק ממנה (לפי הרוחב של ה TCAM).
* **runtimeMeasurements –** המחלקה אחראית על שמירת נתונים מההרצה של האלגוריתם כגון כמות הפניות ל TCAM אורך החבילה וכו'...
* **rtcamCompressedHttp –** מימוש האלגוריתם, מקבל כקלט אוביקט מסוג **gzipPacket** ואוביקט מסוגtcamSimulator ומריץ את האלגוריתם על החבילה תוך שימוש ב TCAM.
* **algorithmResult –** האוביקט שמוחזר אחרי ריצת האלגוריתם, מכיל רשימה של ההתאמות שהאלגוריתם מצא ואת המדידות של זמן הריצה שהאלגוריתם מדד.

**דוגמת הרצה של האלגוריתם:**

* אתחול ה TCAM

**val** tcamSimulator = **new** tcamSimulator(width = 10)  
 tcamSimulator.initialize("path/to/snort/file")

* אתחול החבילה

**val** gzipAscii = Converter.*ToGzipAscii*(tcamPackage)  
 **val** gzipPacket = **new** gzipPacket(gzipAscii)

* הרצת האלגוריתם

**val** rtcamCompressedHttp = **new** rtcamCompressedHttp(packet = gzipPacket, tcam = tcamSimulator)  
 **val** algorithmResult = rtcamCompressedHttp.execute()  
   
}

# תוצאות שהתקבלו

מימשתי סימולציה של TCAM בכמה רוחבים שונים ועבור כל אחד הרצתי את האלגוריתם עבור דפי אינטרנט מכווצים ולא מכווצים , ותוך כדי האלגוריתם מדדתי מדידות לגבי כל ריצה לדוגמא: כמות הפניות ל TCAM, על כמה בייטים דילגתי תוך כדי הריצה וכו'...

### השפעת רוחב ה TCAM על ממוצע הדילוגים

במדידה זאת השוואתי איך משפיע השינוי של רוחב ה TCAM על כמות הדילוגים בממוצע. ההשוואה היא בין המימוש שלי שעובד על דפים מכווצים(קו כחול) מול המימוש הנאיבי שקודם כל עושה decompression ואז מריץ את האלגוריתם(קו אדום).

ניתן לראות שבמימוש שלי ממוצע הדילוגים עולה משמעותית ככל שרוחב ה TCAM גדל יחסית לאלגוריתם הנאיבי ששם הקו עולה בצורה יותר מתונה.

### השפעת גודל החבילה על ממוצע החיפושים ב TCAM

במדידה זאת השוואתי איך משפיע אורך החבילה על כמות החיפושים ב TCAM.

ההשוואה היא בין המימוש שלי שעובד על דפים מכווצים(קו כחול) מול המימוש הנאיבי שקודם כל עושה decompression ואז מריץ את האלגוריתם(קו אדום).

ניתן לראות שכאשר החבילה יחסית קטנה כמות החיפושים כמעט זהה, אבל כאשר החבילות הם יותר גדולות יש הפרש משמעותי בכמות החיפושים ב TCAM והאלגוריתם הנאיבי מבצע הרבה יותר חיפושים. זה נובע מכך שיש הרבה יותר דילוגים במימוש שלי בגלל הפוינטרים ולכן במימוש שלי הוא סורק הרבה פחות בייטים.

# פתרונות דומים בספרות

צוות ישראלית של שלושה אנשי מדעי המחשב מה- IDC בהרצליה ומאוניברסיטת תל אביב מציעים במאמרים שלהם “Accelerating Multi-Pattern Matching on Compressed HTTP Traffic” ו- “Shift-based Pattern “Matching for Compressed Web Traffic אלגוריתמים שמאפשרים סריקה של חבילות מכווצות. הרעיון של שני האלגוריתמים התבסס על כך כי אפשר, בעזרת מערכת אבטחת רשתות אמיתית, לדלג בסריקה על יותר מ- 70% מהחבילה.

האלגוריתם הראשון, שענת ברמלר בר וירון קורל היצעו ב 2009 , נקרא ACCH(Aho-Corasick-based algorithm for Compressed HTTP) שמתבסס על האלגוריתם חיפוש של Aho-Corasick שבונה מראש אוטומט סופי כי בעזרתו נוכל לזהות חתימות ב- O(n). בתחילתו של מערכת הפעלת Unix , הפקודה grep השתמשה באלגוריתם זה.

לאחר מכן, אותו צוות ואיש מדעי המחשב, ויקטור זיגדון מה- IDC , פיתחו אלגוריתם חיפוש הנקרא SPC , רשאי תיבות של שם המאמר. האלגוריתם בא לשפר את האלגוריתם Wu-Manber לזיהוי חתימות. הרעיון הוא לשמור בתוך וקטור של ביטים, שנקרא Partial Match bit Vector , מידע על תחילת התאמה עם חתימה מסוימת. אכן נדליק את הביט ה- j עם באינדקס j בחבילה נמצאה התאמה של m ביטים ) m גודלו של חלון הסריקה(. כאשר נתקלים במצביע בחבילה, אנחנו נוכל בקלות לדעת, בעזרת וקטור ביטים, האם יש אפשרות להתאמה במצביע הסרוק. אם אין ביט דלוק בביט הראשון אז נוכל לעשות קפיצה בטוחה בסריקה וכך להתקדם יותר מהר עד סוף הסריקה. במקרה שנמצאה התאמה, אז נצטרף לבצע בדיקה מלאה של המחרוזת של עליה מצביע המצביע.

# רשימת ספרות

• *"High Performance String Matching Algorithm for a Network Intrusion Prevention System",* Yaron Weinsberg, Shimrit Tzur-David, Danny Dolev, Tal Anker.

• *"Shift based Pattern Matching for Compressed Web Traffic"*, Anat Bremler-Brarr, Yaron Koral, Victor Zigdon.

• *“High Performance Deep Packet Inspection”*, Yaron Koral

• *“Accelerating Multi-Patterns Matching on Compressed HTTP Traffic”*, Anat Bremler-Barr, Yaron Koral

• “*Alexa – The Web Information Company* “, [www.alexa.com](http://www.alexa.com)

• *“Usage of Gzip Compression for websites”*, web3Techs website.

*• Snort - network intrusion detection system for UNIX and Windows:* [*www.snort.org*](http://www.snort.org)

1. http://www.cs.huji.ac.il/~shimritd/publications/hpsr06.pdf [↑](#footnote-ref-1)